unSP Programmer's Guide

V1.0 November 26, 2007

No.19, Industry E. Rd. IV, Hsinchu Science Park, Hsinchu City 30077, Taiwan, R.O.C.
Tel: +886-3-666-2118 Fax: +886-3-666-2117 Web: www.generalplus.com

Important Notice

GENERALPLUS TECHNOLOGY INC. reserves the right to change this documentation without prior notice. Information provided by GENERALPLUS TECHNOLOGY INC. is believed to be accurate and reliable. However, GENERALPLUS TECHNOLOGY INC. makes no warranty for any errors which may appear in this document. Contact GENERALPLUS TECHNOLOGY INC. to obtain the latest version of device specifications before placing your order. No responsibility is assumed by GENERALPLUS TECHNOLOGY INC. for any infringement of patent or other rights of third parties which may result from its use. In addition, GENERALPLUS products are not authorized for use as critical components in life support devices/ systems or aviation devices/systems, where a malfunction or failure of the product may reasonably be expected to result in significant injury to the user, without the express written approval of Generalplus.

Table of Content

Revision History

Revision	Date	Revised By	Remark
V1.0	2007/11/26	Summer Yi	Original

1 Introduction

1.1 General Description

The *unSP*, pronounced as micro-n-S-P, is the first 16-bit microprocessor developed by Generalplus. Not only does the *unSP* perform general operations such as addition, subtraction and other logical operations, but it also supports multiplication and inner-product operations for digital signal processing. Now, the *unSP* has a series of version named *unSP* 1.0, *unSP* 1.1, *unSP* 1.2, *unSP* 1.3 and *unSP* 2.0.

1.2 Pin Diagram

1.2.1 Pin Diagram and Description of unSP-1.2

- **Pin Diagram**

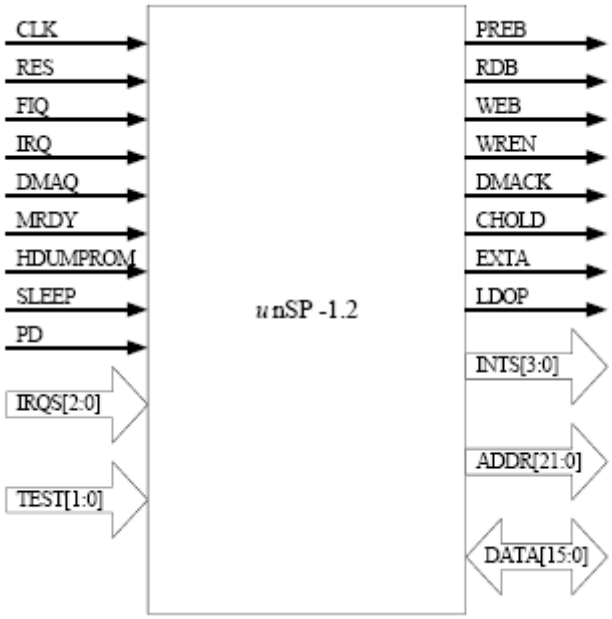


Figure 1.1

- **Pin Description**

Table 1.1

Name	I/O	Description	Number
CLK	INPUT	External clock	1
RES	INPUT	External reset	1
FIQ	INPUT	Fast interrupt request	1
IRQ	INPUT	Normal interrupt request	1
IRQS	INPUT	Normal interrupt request select	3
DMAQ	INPUT	DMA request	1
MRDY	INPUT	Memory data ready signal	1
TEST	INPUT	Test mode select pins	2

Name	I/O	Description	Number
HDUMPROM	INPUT	Dump internal ROM to memory bus	1
SLEEP	INPUT	Sleep mode	1
PD	INPUT	Power down mode	1
ADDR	OUTPUT	Address bus	22
PREB	OUTPUT	Memory pre-charge signal	1
RDB	OUTPUT	Memory read enable signal	1
WEB	OUTPUT	Memory write enable signal	1
WREN	OUTPUT	Memory accessing mode	1
DMACK	OUTPUT	DMA acknowledge	1
CHOLD	OUTPUT	CPU stall signal	1
EXTA	OUTPUT	Access data with DS	1
LDOP	OUTPUT	Fetch instruction into CPU	1
INTS	OUTPUT	CPU operation mode	4
DATA	INOUT	Data bus	16

1.2.2 Pin Diagram and Description of unSP-1.3

■ **Pin Diagram**

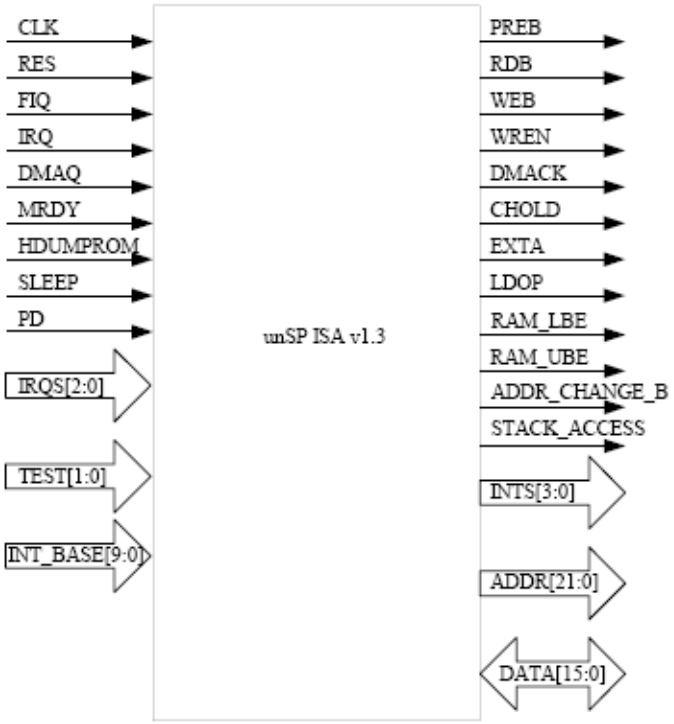


Figure 1.2

- Pin Description

Table 1.2

Name	I/O	Description	Number
CLK	INPUT	External clock	1
RES	INPUT	External reset	1
FIQ	INPUT	Fast interrupt request	1
IRQ	INPUT	Normal interrupt request	1
IRQS	INPUT	Normal interrupt request select	3
DMAQ	INPUT	DMA request	1
MRDY	INPUT	Memory data ready signal	1
TEST	INPUT	Test mode select pins	2
INT_BASE	INPUT	Interrupt vector base address	10
HDUMPROM	INPUT	Dump internal ROM to memory bus	1
SLEEP	INPUT	Sleep mode	1
PD	INPUT	Power down mode	1
ADDR	OUTPUT	Address bus	22
RAM_LBE	OUTPUT	Upper byte Enable	1
RAM_UBE	OUTPUT	Lower byte Enable	1
ADDR_CHANGE_B	OUTPUT	Address change signal	1
STACK_ACCESS	OUTPUT	Indicate that CPU is accessing stack	1
PREB	OUTPUT	Memory pre-charge signal	1
RDB	OUTPUT	Memory read enable signal	1
WEB	OUTPUT	Memory write enable signal	1
WREN	OUTPUT	Memory accessing mode	1
DMACK	OUTPUT	DMA acknowledge	1
CHOLD	OUTPUT	CPU stall signal	1
EXTA	OUTPUT	Access data with DS	1
LDOP	OUTPUT	Fetch instruction into CPU	1
INTS	OUTPUT	CPU operation mode	4
DATA	INOUT	Data bus	16

1.2.3 Pin Diagram and Description of *unSP*-2.0

- **Pin Diagram**

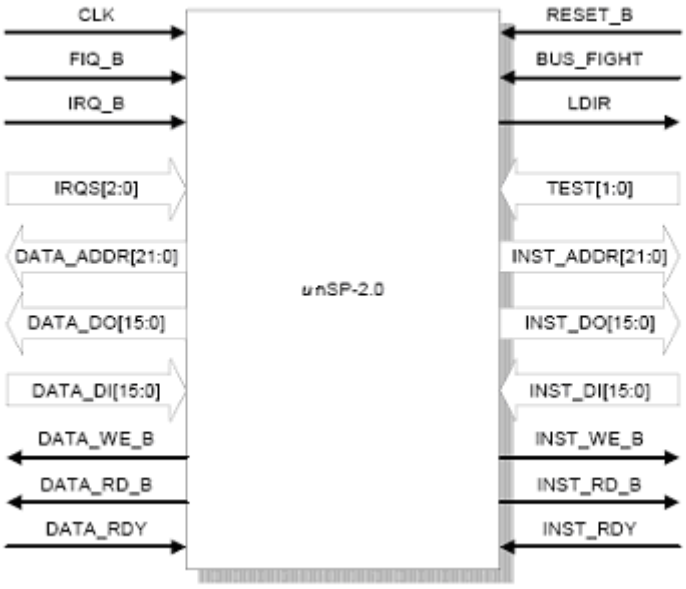


Figure 1.3

- **Pin Description**

Table 1.3

Name	I/O	Description	Number
CLK	Input	CPU clock	1
RESET_B	Input	CPU reset	1
FIQ_B	Input	Fast interrupt request (level trigger)	1
IRQ_B	Input	Interrupt request (level trigger)	1
IRQS	Input	Interrupt request source	3
TEST	Input	Test mode select	2
INST_ADDR	Output	Instruction Bus address	22
INST_DI	Input	Instruction Bus data input	16
INST_DO	Output	Instruction Bus data output	16
INST_RD_B	Output	Instruction Bus read enable signal	1
INST_WE_B	Output	Instruction Bus write enable signal	1
INST_RDY	Input	Instruction Bus ready signal	1
DATA_ADDR	Output	Data Bus address	22
DATA_DI	Input	Data Bus data input	16
DATA_DO	Output	Data Bus data output	16
DATA_RD_B	Output	Data Bus read enable signal	1

DATA_WE_B	Output	Data Bus write enable signal	1
DATA_RDY	Input	Data Bus ready signal	1
BUS_FIGHT	Input	Instruction Bus & Data Bus access same	1
LDIR	Output	Load next instruction	1

1.3 Features

1.3.1 Features of unSP-1.0 and unSP-1.1

- 16-bit micro controller with DSP function
- Memory bus interface
  - Address width: 22-bit
  - Data width: 16-bit
  - 4M words (8M bytes) memory space
  - 64 banks/ 64k words per bank
- 8*16-bit registers
  - 4 general registers (R1~R4)
  - 4 system registers (SP, BP, SR, PC)
- 10 interrupts
  - 1 fast interrupt (FIQ)
  - 8 normal interrupt (IRQ0-IRQ7)

    Software interrupt (BRK)
- 6 addressing modes
  - Register Direct(R)
  - Register Indirect ([R])
  - Immediate (IM6/IM16)
  - Memory Absolute Address ([A6]/ [A16]/

    [A22]) Indexed Address ([BP+IM6])
  - PC Relatively (PC+IM6)
- 16-bit multiplication
  - 2 operation modes: signed*signed, unsigned*signed
  - 16-levels inner product operation
  - 2 operation modes: signed*signed, unsigned*signed
  - 4 guard bits to avoid overflow

Difference between *unSP* -1.0 and *unSP*- 1.1

UnSP - 1.1 is an enhanced version of *unSP* -1.0. The behaviors of CS and DS registers are changed to facilitate large program execution and large data access.

Table 1.4

	UnSP **1.0**	*unSP* **1.1**
GOTO instruction	The target address is limited in current page (16-bit, implemented as PC=target_addr).	The target address can be any address in the 4M words memory (22-bit).
CS register auto-increment/decrement	N/A	1. New feature 2. During program execution, PC will be incremented by one continuously. If a carry takes place, CS will be incremented by one. 3. Branch instruction versus PC is based on the combination of CS and PC. The result will be stored back to the CS and PC.
DS register auto-increment/decrement	N/A	1. New feature. 2. In indirect addressing mode, if the D:[++Rs] / D:[Rs++] / D:[Rs--] are used, these operations are executed based on the 22-bit register arithmetic and the final result is stored back to DS and RS.
Instruction cycles	Longer	Generally speaking, most instruction cycles are faster than *unSP* 1.0.

1.3.2 Features of unSP-1.2

- 16-bit micro controller with DSP function
- Memory bus interface
  - Address width: 22-bit
  - Data width: 16-bit
  - 4M words (8M bytes) memory space
  - 64 banks/ 64k words per bank
- 13*16-bit 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-bit 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-bit dividend and 16-bit 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-bit shift operation

  Support 32-bit shift operation by combining 2 shift instructions
- Support DMA function
- Support power down/sleep mode

1.3.3 Features of unSP-1.3

The most significant difference between *unSP* 1.2 and *unSP* 1.3 is the byte addressing modes in *unSP*1.3.

- 16-bit micro controller with DSP function
- Memory bus interface
  - Address width: 22-bit
  - Data width: 16-bit
  - 4M words (8M bytes) memory space
  - 64 banks/ 64k words per bank
  - Byte accessing-mechanism with new addressing mode
- 14*16-bit and 1*6-bit registers
  - 4 general registers (R1~R4)
  - 4 secondary registers (SR1~SR4)
  - 6 system registers (SS, MDS, 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)

- 9 addressing modes
  - Register
  - Immediate
  - Direct
  - Indirect
  - Multi-indirect
  - Displacement
  - Byte register indirect
  - Byte indexed address
  - Byte register indexed address
- 16-bit 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-bit dividend and 16-bit divisor
  - 16 continuous operations (DIVS, DIVQ) or only one operation (DIVSS, DIVUU) to generate correct quotient.
- Bit-operation
  - Bit test/ set/ clear/ inverse operation
  - Destination can be register or memory
  - 2 address mode (direct, indirect) for memory access.
- Effective-exponent detect operation
- 16-bit shift operation

  Support 32-bit shift operation by combining 2 shift instructions
- Support DMA function
- Support power down/sleep mode

1.3.4 Features of *unSP*-2.0

- 16-bit micro controller with DSP function

- *unSP 1.*2 binary compatible
- Modified Harvard architecture
- Instruction memory bus (IM): addr: 22-bit / data: 16-bit
- 64 banks / 64k words per bank
- Data memory bus (DM): addr: 22 bits / data: 16-bit

  64 banks / 64k words per bank
- 2 configurations
  - IM/DM share 4M words memory space (default)
  - IM/DM own separate 4M words memory space (Not recommended. Assembler and linker do not support overlapped address of IM and DM.)
- 4-stage pipelined architecture
  - IF (Instruction Fetch)
  - DE (Decode)
  - MR (Memory Read Access)
  - EX/MW (Execution/Memory Write)
- 21*16-bit registers
  - 4 general registers (R1-R4)
  - 4 secondary-bank registers for interrupt (SR1-SR4)
  - 1 base address register (BP)
  - 3 system registers (SP, SR, PC)
  - 1 inner flag register (FR)
  - 8 extended registers (R8-R15)
- 10 interrupt sources
  - 1 fast interrupt (FIQ)
  - 8 normal interrupts (IRQ)
  - 1 software interrupt (BRK)
  - Support IRQ nested mode with priority
- 6 addressing modes
  - Register
  - Immediate (I6/I16)
  - Direct (A6/A16)
  - Indirect + auto indexing address (DS indirect)
  - Displacement (BP+IM6)
  - Multiple indirect (PUSH/POP)
- 16-bit multiplication
  - 3 operation modes: signed*signed, unsigned*signed, unsigned*unsigned
  - Integer / Fraction modes

- 16-levels inner product operation
  - 1 cycle MAC (multiplier/accumulator) unit
  - 3 operation modes: signed*signed, unsigned*signed, unsigned*unsigned
  - 4 guard bits to avoid overflow
  - Integer / Fraction modes
- Non-restoring division
  - 32-bit dividend and 16-bit divisor
  - Need 16 continuous operations (DIVS, DIVQ) to generate correct quotient
- Effective-exponent detect operation
- Bit operations
  - Support 4 operations: test, set, clear, inverse.
  - Destination can be register or memory.
  - 2 address mode (direct, indirect) for memory access.
- 16-bit shift operation
  - 1 cycle log-shifter
  - Support 32-bit shift operation by combining 2 shift instructions.

1.4 Architecture

1.4.1 Architecture of unSP-1.0 and unSP-1.1

The design goal of *unSP* 1.0 and *unSP* 1.1 is to achieve high performance with low cost. It uses the traditional multi-cycle architecture. The organization of *unSP* 1.0 and *unSP* 1.1 is illustrated as below. The principal components are:

- The general registers (R1-R4) and the system registers (SP, BP, SR, PC) in the register bank.
- The data register (DR), which store the data.
- The address generator unit (AGU), which selects and holds all memory address and generate sequential address when required.
- The shifter, which can shift or rotate one operand by any number of 4-bit.
- The ALU, which performs the arithmetic and logic functions required by current instruction.

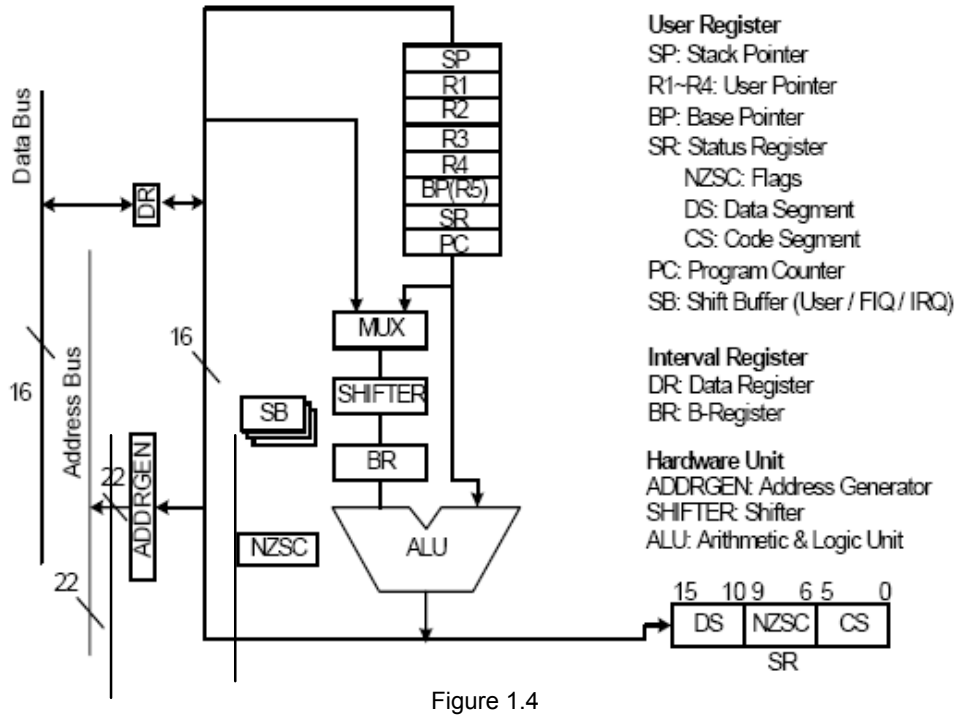


Figure 1.4

1.4.2 Architecture of unSP-1.2

The design goal of *unSP* 1.2 is to achieve high performance with low cost. It uses the traditional multi-cycle architecture. The organization of *unSP* 1.2 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 selects and holds all memory address and generate sequential address when required.
- The shifter, which can shift or rotate one operand by any number of 4-bit.
- The ALU, which performs the arithmetic and logic functions required by current instruction.

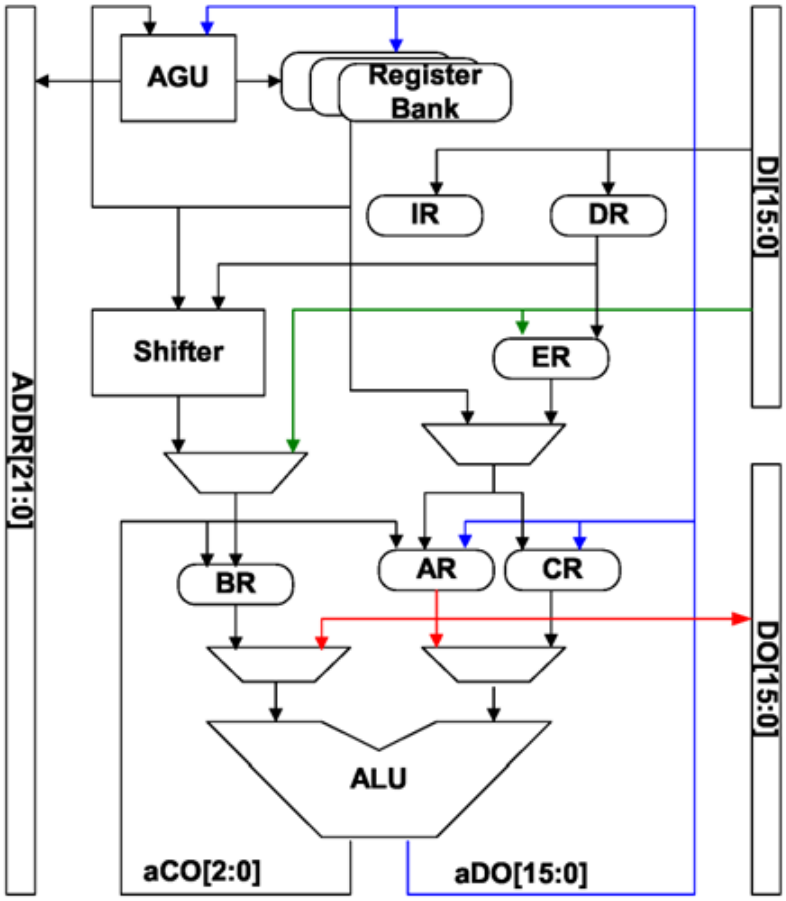


Figure 1.5

1.4.3 Peripheral Interface of unSP-1.2

- **DMA Timing**

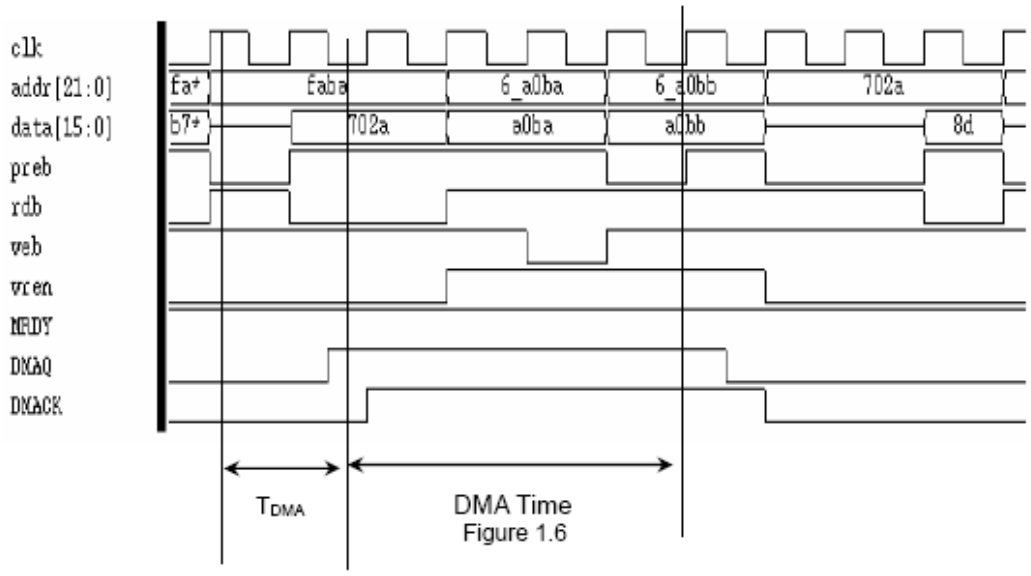


Figure 1.6

- DMAQ: DMA request, active high.
- DMACK: when CPU accept DMA request from DMAQ, it will reply DMACK signal and release address, data bus, preb, web, rdb, wren signals after last memory access.

The longest delay without memory waiting cycle from DMAQ request to DMACK reply T_{DMA} <= 3 clock cycles (max cycles for instruction to access memory)

■ **SLEEP Timing**

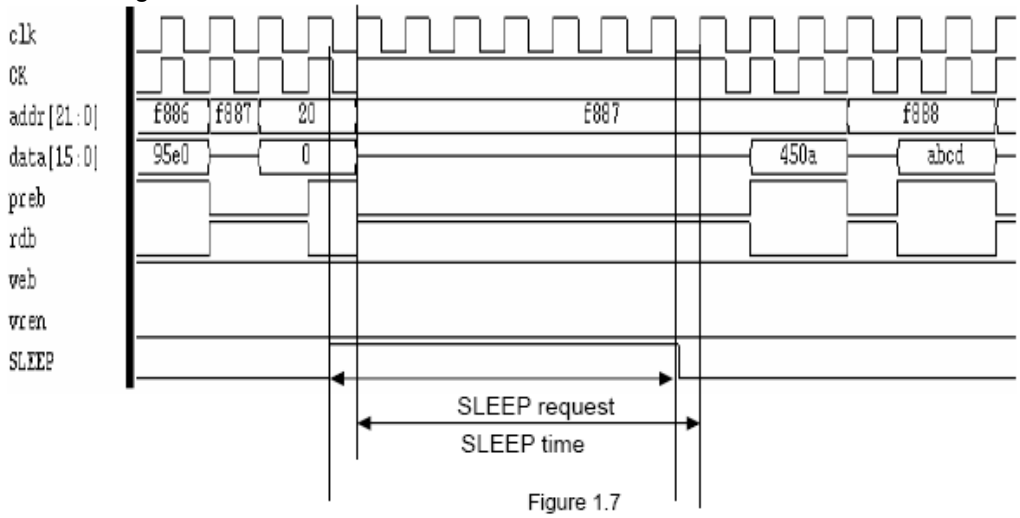


Figure 1.7

- CK: CPU internal clock.
- SLEEP will stall the CPU clock at high and keep address, data, preb, rdb, web, wren at origin value.

■ **PD (Power Down) Timing**

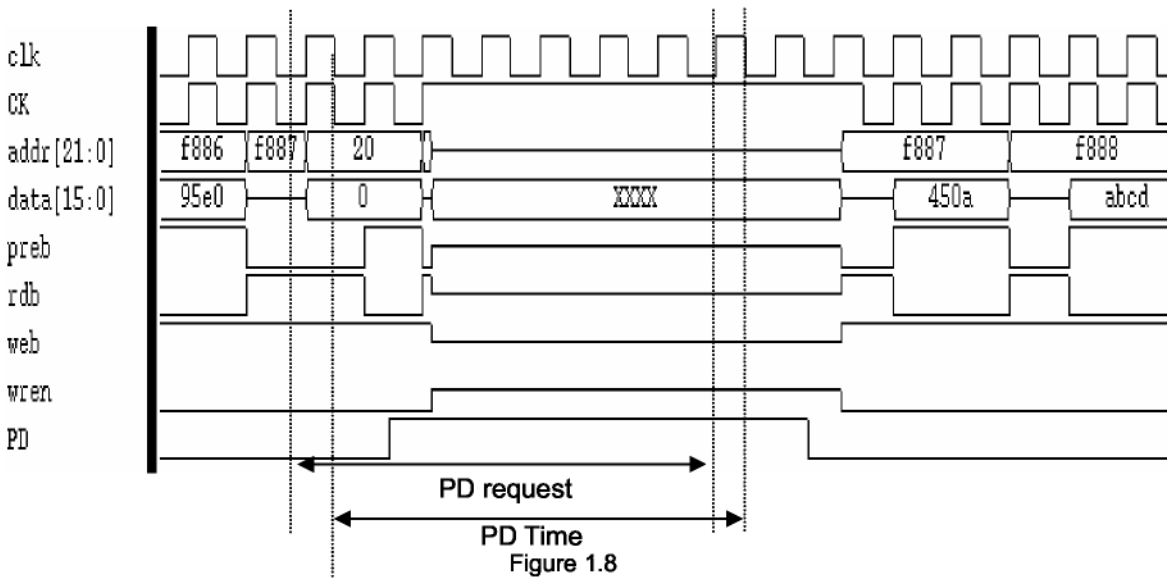


Figure 1.8

PD will stall the CPU clock at high and release the address, data bus, preb, rdb, web, wren signals.

- **HDUMPROM Timing**

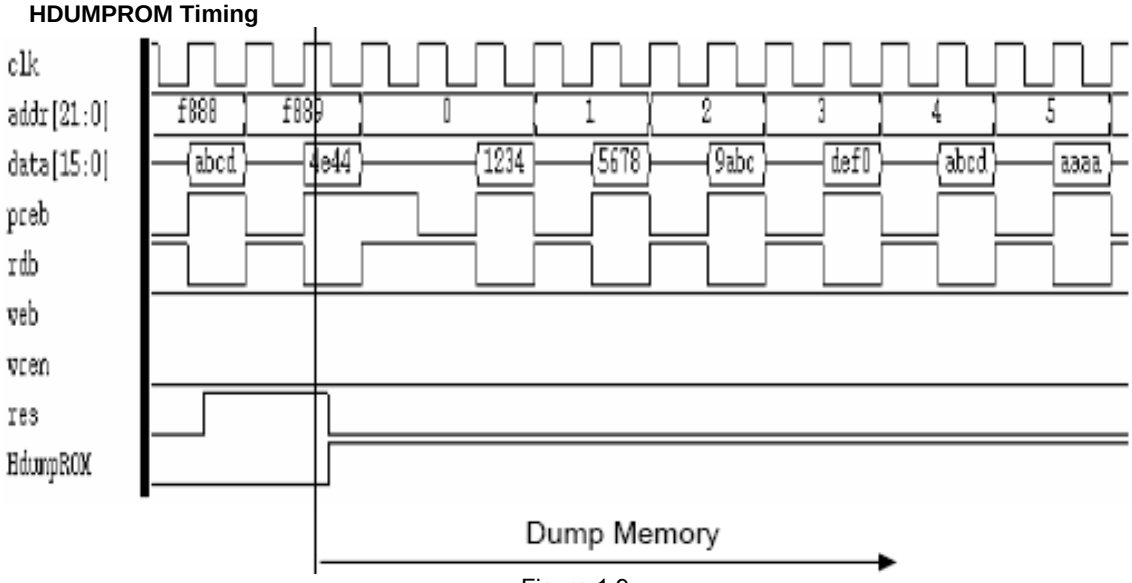


Figure 1.9

HdumpROM is used to dump memory content from current address. If you want to dump all memory content to data bus, you can use RESET and HdumpROM signal. After reset and keep HdumpROM at high, the memory content from 0x000000 ~ 0x3FFFFF will be dumped to memory bus contiguously every 2 cycles.

1.4.4 Architecture of unSP-1.3

The design goal of *unSP* 1.3 is to achieve high performance with low cost. It uses the traditional multi-cycle architecture. The organization of *unSP* 1.3 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), 6 system registers (SS, SP, MDS, 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 selects and holds 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.

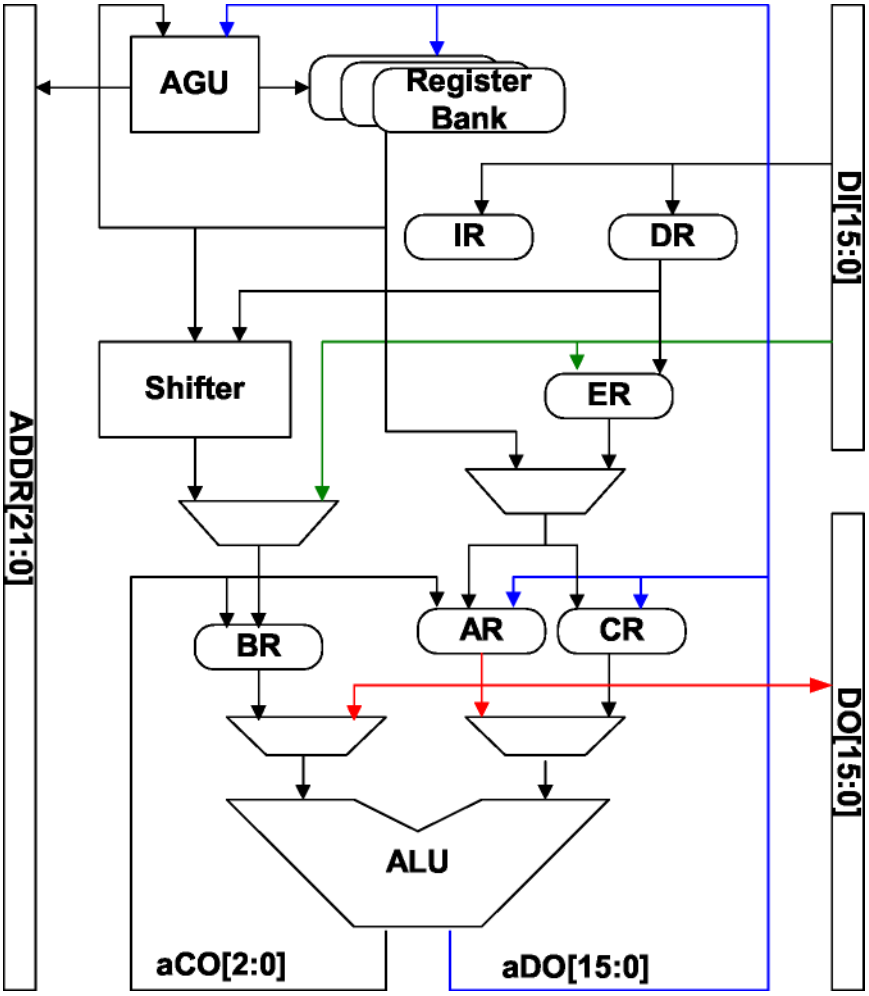

AGU
Register Bank
IR
DR
DI[15:0]
Shifter
ER
ADDR[21:0]
BR
AR
CR
DO[15:0]
ALU
aCO[2:0]
aDO[15:0]

Figure 1.10

1.4.5 Architecture of *unSP*-2.0

unSP–2.0 is a 4-stage pipeline architecture. Its organization 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), 8 extend registers (R8~R15) and 1 flag register (FR) in the register bank.
- The instruction register (IR), which stores the current instruction fetched from instruction bus. The decoder will generate all control signals for data path according to the instruction register.
  - The parameter register (PR), which stores the second word of current instruction if instruction length is 2 words.
  - The instruction address generator unit (IAG), which selects and holds all instruction address and generate sequential address when required.
  - The data address generator unit (DAG), which generate and hold all data address.
  - The Shifter, which can shift or rotate one operand by any number of 4-bit.
  - The ALU, which performs the arithmetic and logic functions required by current instruction.
  - The MAC, which performs the multiplication and accumulate functions required by current instruction.

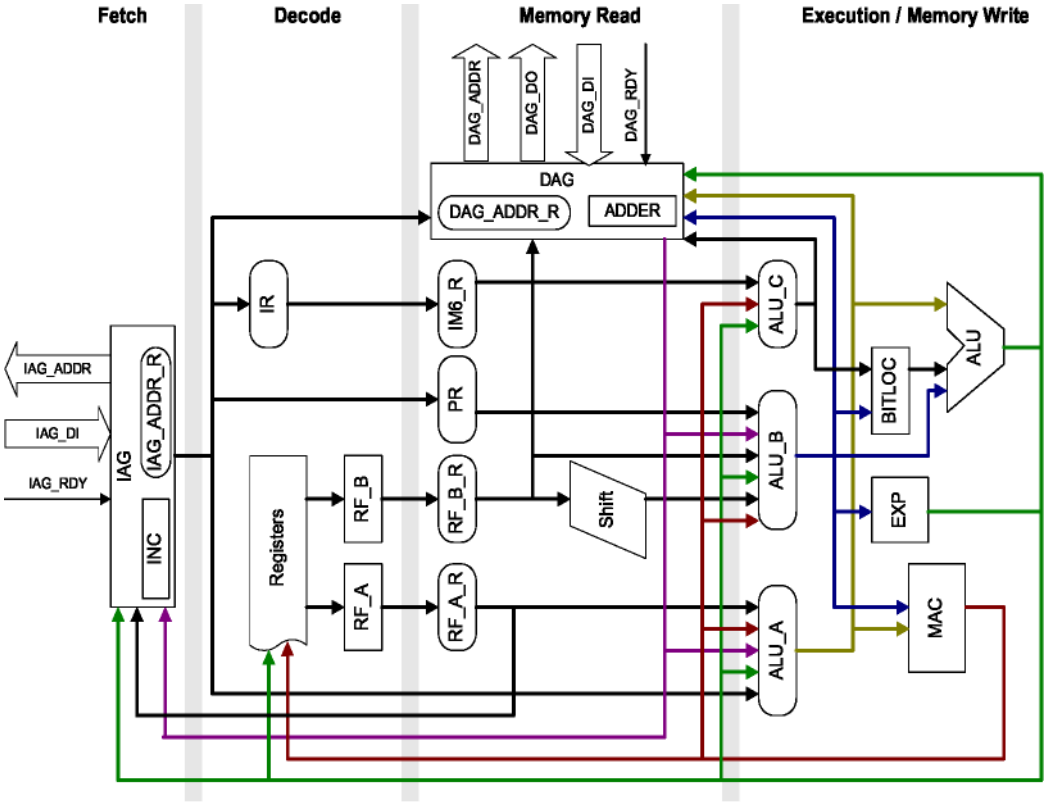


Figure 1.11

1.4.6 Pipeline Feature of unSP-2.0

unSP 2.0 employs a simple 4-stage pipeline with the following pipeline stages:

- Fetch:

  The instruction is fetched from memory and placed in the instruction registers.

- Decode:

  The instruction is decoded and the data-path control signals prepared for the next cycle. If current instruction needs to read data from memory, the access address will be generated and issued in this stage.

- Memory Read:

  Waiting state for memory read access. If current instruction needs performing shift operation, it will be done in this stage.

- Execution / Memory Write:

  All ALU operations, multiplication are executing in this stage and the result will be write back to register or memory at the next cycle, if current instruction need to write data to memory, the access address will be generated and issued in this stage.

At any one time, four different instructions may occupy each of these stages

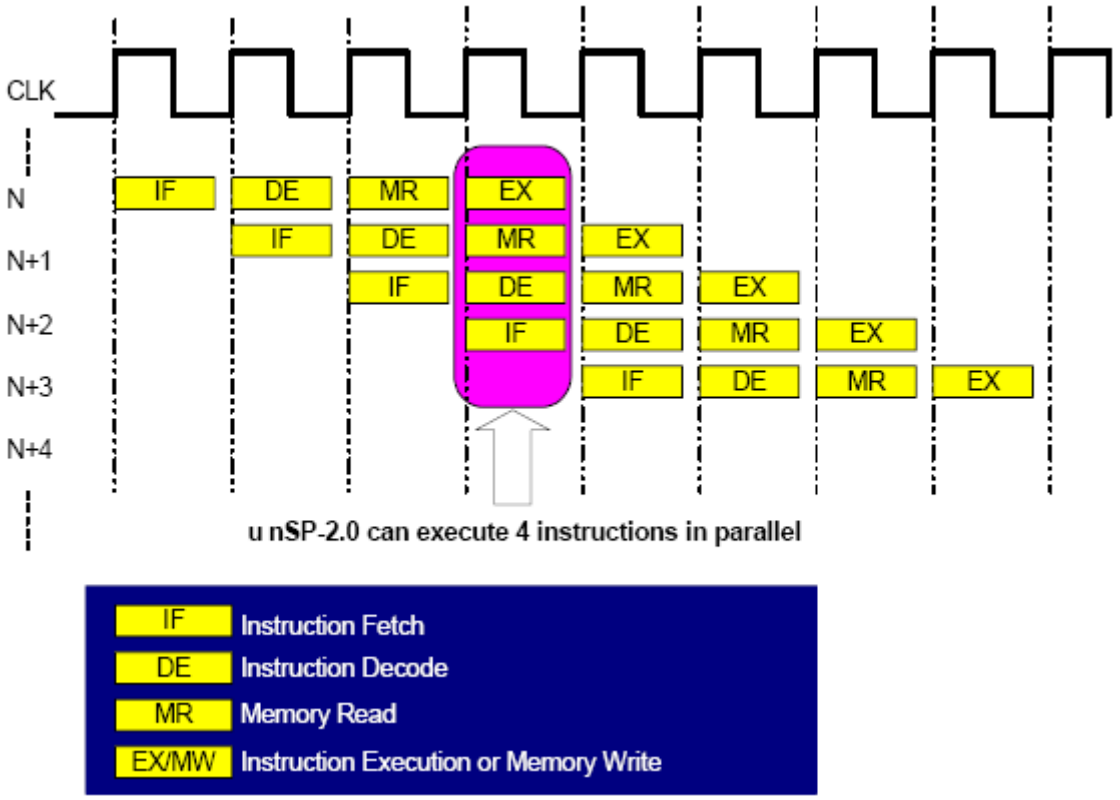


Figure 1.12

1.5 Register

1.5.1 Register of unSP-1.0 and unSP-1.1

unSP 1.0 and *unSP* 1.1 have eight 16-bit registers: Stack Pointer (SP), User Registers (R1, R2, R3, R4), Based Pointer (BP), Status Register (SR) and Program Counter (PC). Please see Table 1.5 for details. The concatenation of R3 and R4 forms a 32-bit register, MR, which is used as the destination register for multiplication and inner-production. The Stack Pointer (SP) automatically increases (POP) or decreases (PUSH) as the *unSP* performing push/pop, subroutine call or interrupt operations. The stack size is limited to 64K, i.e., 0x000000 ~ 0x00FFFF. Since *unSP* 1.0 and *unSP* 1.1 are able to address 4M-word locations, additional 6 bits are needed to construct a 22-bit address from a 16-bit register for fetching instructions (OP codes) and data accessing purposes. These 6 bits reside in the SR register, which contains the Code Segment (CS) and the Data Segment (DS) fields. Therefore, both code addresses and data addresses can be represented in 22 bits.

In *unSP* 1.0, the value in CS will not be changed by sequential execution and conditional branch when crossing page boundaries. This behavior limits each code section cannot be larger than 64K words. Only call instruction and interrupts can change CS. The DS will not be changed by pre-increment addressing, post-increment addressing and post-decrement addressing modes. This behavior limits each data section cannot be larger than 64K words. For example, suppose CS is 0x03 and PC is 0xFFFF, the next instruction fetched by the *unSP* is located at CS:PC = 0x030000, not 0x040000.

In *unSP* 1.1, the content of total 22 bits formed by CS or DS and a register will be changed accordingly when crossing page boundaries. Thus the 64K limitation is removed.

Table 1.5 *unSP* 1.0 and *unSP* 1.1 Registers

Register ID	Name
0 (000)	SP
1 (001)	R1
2 (010)	R2
3 (011)	R3
4 (100)	R4
5 (101)	BP (R5)
6 (110)	SR
7 (111)	PC

1.5.2 Register of unSP–1.2

- **Registers Bank**

  unSP 1.2 adds 4 registers (SR1~SR4) for interrupt service routines 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. The other registers, SP, BP, SR, PC and FR, are not affected by the SECBANK On/Off instruction.

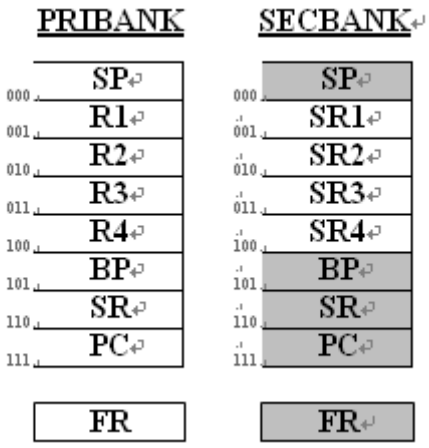


Figure 1.13

- 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. Only the IRQ with higher priority can interrupt it. User can customize the IRQ nest behavior by setting the priority register.

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

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

    For example:

    - If PRI is 4'b1000, IRQ0-7 are enabled
    - If PRI is 4'b0000, IRQ0-7 are disabled
    - If IRQ3 occurred, PRI will be set to 0011. Only IRQ0-2 are enabled.

1.5.3 Register of *unSP*-1.3

- **Registers Bank**

  As *unSP* 1.2, *unSP* 1.3 contains 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. The other registers, SP, BP, SR, PC, FR, SS and MDS, are not affected by the SECBANK On/Off instruction.

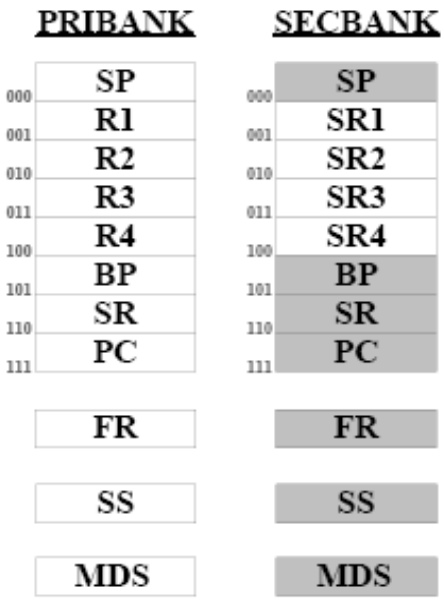


Figure 1.14

- Primary Bank
  - Stack Pointer (SP)
  - General Register (R1~R4)
  - Base Pointer (BP)
  - Status Register (SR)
  - Program Counter (PC)
  - Flag Register (FR)
  - Stack Segment Register (SS)
  - Inner Product Operation Data Segment (MDS)
- 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
AdE	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

- Ade: Byte-mode instruction address alignment error.
- 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. Only the IRQ with higher priority can interrupt it. User can customize the IRQ nest behavior by setting the priority register.

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

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

  For example:

  - If PRI is 4'b1000, IRQ0-7 are enabled
  - If PRI is 4'b0000, IRQ0-7 are disabled
  - If IRQ3 occurred, PRI will be set to 0011. Only IRQ0-2 are enabled.

■ **Stack Segment Register**

The stack segment register (SS) is added to expand the size of stack. The size of SS is 6 bits. After reset, all bits of SS are cleared to be zero. The behavior of following stack related operations are changed.

- PUSH R_H, R_L to [Rs]

  The destination where R_H ~ R_L are pushed in a 22-bit address: $\{SS:Rs\}_{21\sim0}$. That is, higher 6 bits in SS and lower 16 bits in Rs. After the push operation, $\{SS:Rs\}_{21\sim0}$ is decremented by the number of registers pushed.
- POP R_L, R_H from [Rs]

  Increment $\{SS:Rs\}_{21\sim0}$ by 1. Move the content at $\{SS:Rs\}_{21\sim0}$ to Rx. Increment $\{SS:Rs\}_{21\sim0}$ by 1.Move the content at $\{SS:Rs\}_{21\sim0}$ to R_L +1. Repeat these operations (R_H – R_L + 1) times.
- The effective address of [BP+n] addressing mode

The effective address is $\{SS:Rs\}_{21\sim0}$ +n. Move the content at $\{SS:Rs\}_{21\sim0}$ to Rx. Increment $\{SS:Rs\}_{21\sim0}$ by 1.Move the content at $\{SS:Rs\}_{21\sim0}$ to R_L +1. Repeat these operations (R_H – R_L + 1) ti mes.

If SS is zero, the behaviors of these operations are the same as before.

- **Inner Product Operation Data Segment Register (MDS)**

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

- **Inner product operation data segment (MDS) are added to specify the page numbers of the two sources, Rd and Rs.**
- **The data of the two sources can cross the page boundary. When doing the MULS operation, the Rd and Rs will be added one by one. The carry signal will propagate to these MDS. This causes that the MDS are added by one when the value of Rd and Rs change from 0xFFFF to 0x0000.**

1.5.4 Registers of *unSP*- 2.0

- **Normal mode**

  There are 3 system registers (SP, SR, PC), 4 general registers (R1~R4) and 1 flag registers (FR) can be used in normal mode for user program operation.

- **Registers Bank**

  4 secondary bank registers (SR1~SR4) are added for interrupt service routines 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.

- **Extend Registers**

  unSP 2.0 add 8 extend registers (R8~R15) to reduce register swapping effort while executing complicity operations to improve performance, 8 extend instruction types are also added to do ALU operation between extend registers and memory or original registers sets (R0~PC).

  - Primary Bank
    - Stack Pointer (SP)
    - General Register (R1~R4)
    - Base Pointer (BP)
    - Status Register (SR)
    - Program Counter (PC)
    - Flag Register (FR)
    - Extend Register (R8~R15)
  - Secondary Bank
    - Secondary Register (SR1~SR4)

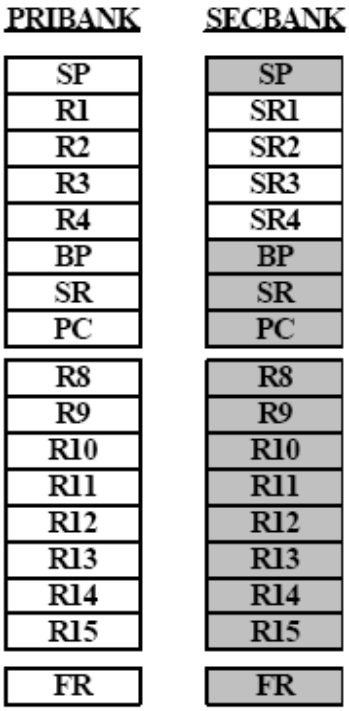


Figure 1.15

- **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. Only the IRQ with higher priority can interrupt it. User can customize the IRQ nest behavior by setting the priority register.

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

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

  For example:

  - If PRI is 4'b1000, IRQ0-7 are enable
  - If PRI is 4'b0000, IRQ0-7 are disable
  - If IRQ3 occurred, PRI will be set to 0011. Only IRQ0-2 are enable.

1.6 Memory

1.6.1 Memory Map of unSP

The address map of *unSP* is divided by every 64K words (64K x 16 bits), called a page. The first page, PAGE0, corresponds to A[21:16]=0. The 4M-word (4096K) memory can be divided into total of 64 pages by A [21:16]=0x00 ~ 0x3F. The selection of page is defined by either a 6-bit Code Segment (CS) or 6-bit Data Segment (DS) in Status Register (SR), depends on execution opcode fetching or data accessing respectively. In memory mapping, PAGE0 is designed for storing data that is frequently accessed, e.g. working memory or peripherals. The other pages (non-zero pages) are designed for storing program codes or large chunk of data.

1.6.2 Memory Interface of unSP-1.2

The memory interface of *unSP* 1.2 is an asynchronous interface. Whenever the address transits, the pre-charge (preb) signal will be pulled low 1 cycle to indicated the memory access and the read enable signal (rdb) or write enable signal (web) will be pulled low at next cycle. The CPU may really need data at the second or third cycle after the address transits, so the rdb may be kept 1 cycle or 2 cycles low while reading memory. If the memory bus is not ready for CPU accessing, the memory ready signal (MRDY) must be pulled low to stall CPU accessing before the clock rising edge of rdb or web access cycle. Besides the rdb and web signals, there is an additional signal (wren) indicating the memory read/write accessing within full memory access period. The detail timing diagrams are illustrated as below.

- **Memory read timing without CPU waiting (2 cycles)**

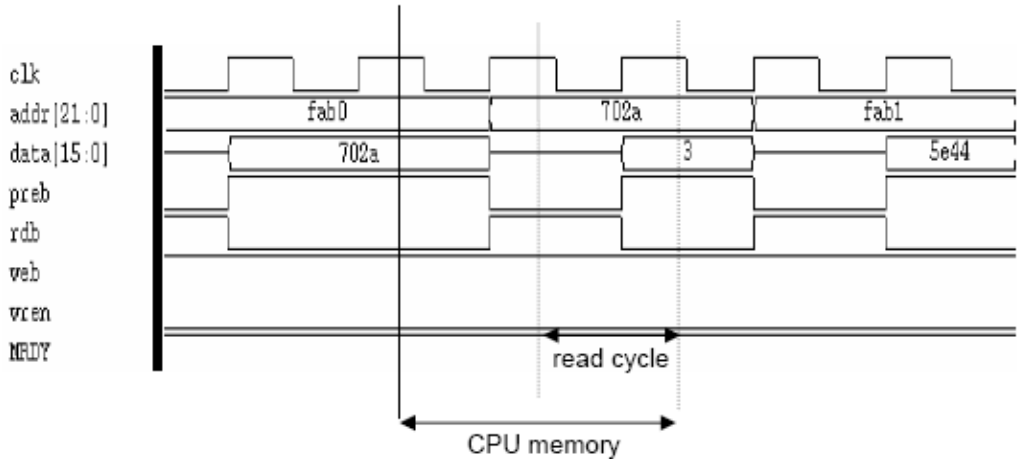


Figure 1.16

- **Memory read timing without CPU waiting (3 cycles)**

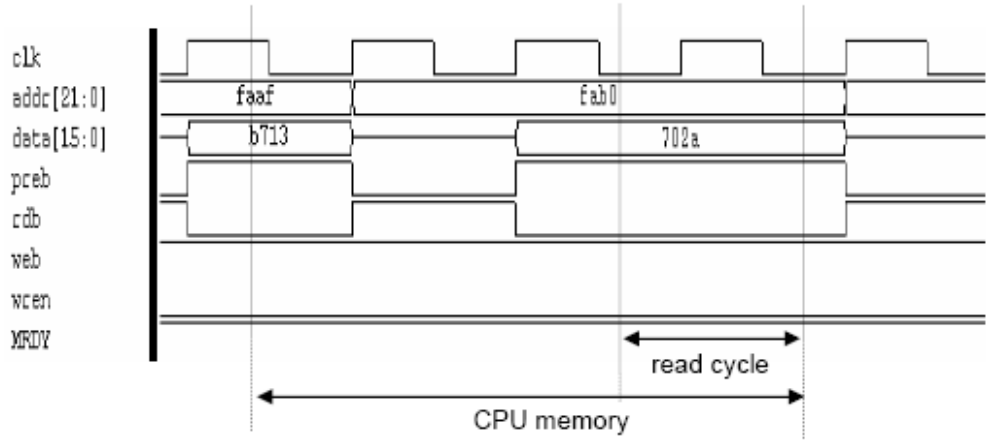


Figure 1.17

- **Memory read timing with CPU waiting (2 cycles)**

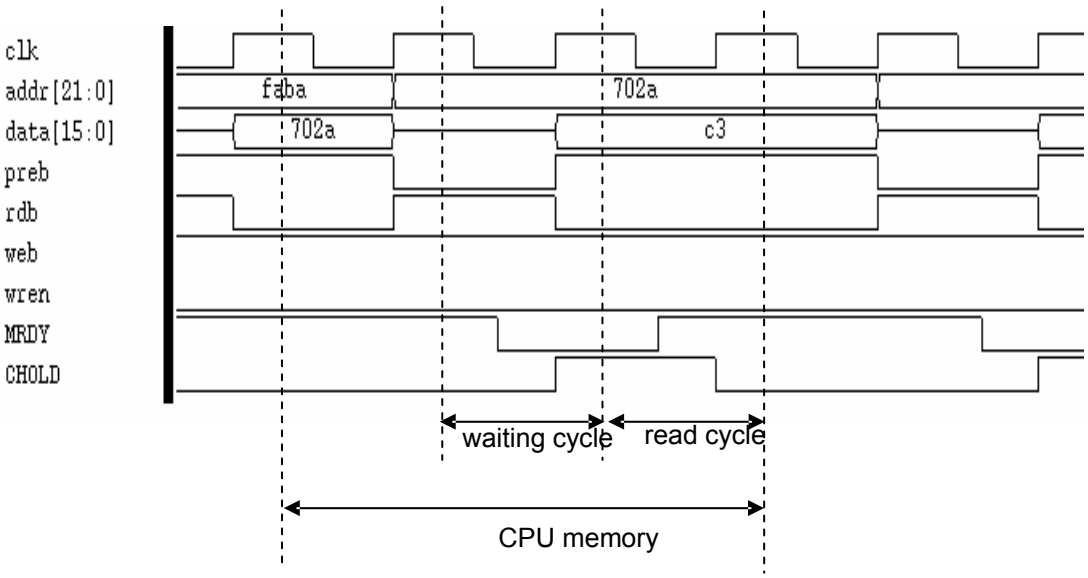


Figure 1.18

- **Memory write timing without CPU waiting**

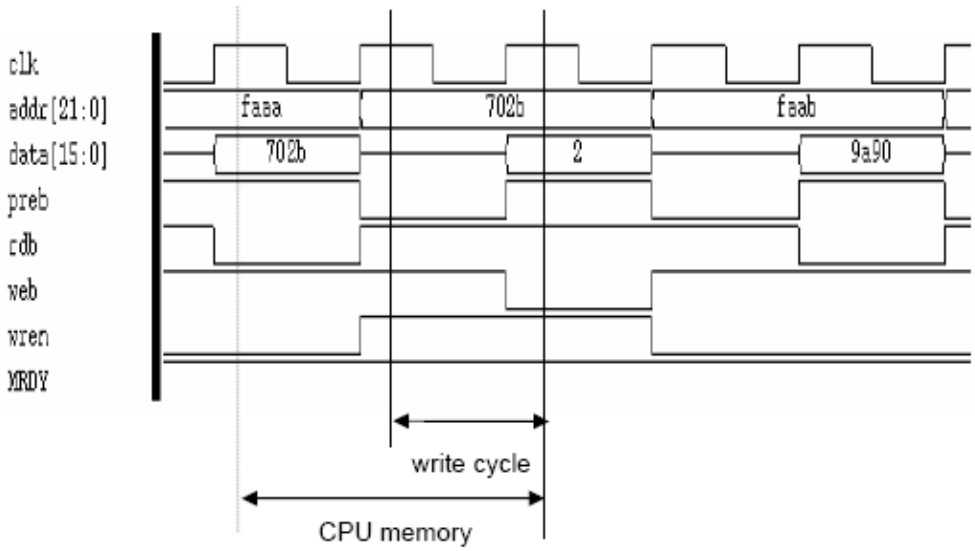


Figure 1.19

- **Memory write cycle with CPU waiting**

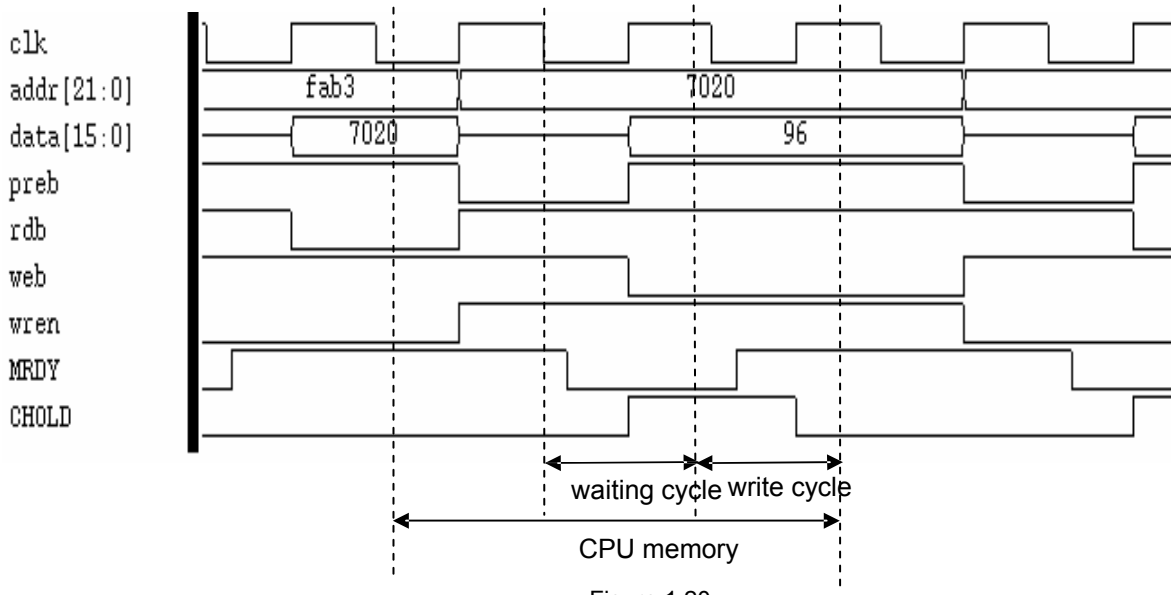


Figure 1.20

WEB: Memory write enable, changes at clock rising edge, active low.

RDB: Memory read enable, changes at clock rising edge, active low.

PREB: Memory pre-charge signal, changes at clock rising edge, active low.

WREN: Memory write signal, changes within full memory access period, active high.

MRDY: Memory bus ready signal, triggered by external device, must be stable before the clock rising edge of read/write cycle.

CHOLD: CPU internal stall signal.

1.6.3 Memory Architecture of *unSP*-1.3

The memory interface of *unSP* 1.3 is an asynchronous interface. Two cycles are needed for CPU to access memory without external memory wait. At the first cycle, the pre-charge (preb) signal will be pulled low 1 cycle to indicate memory accessing and the read enable signal (rdb) or write enable signal (web) will be pulled low at next cycle. If the memory bus need more cycles to prepare data, the memory bus ready signal (MRDY) should be pulled low to stall CPU accessing before end of read or write cycle. Besides the rdb and web signals, there is an additional signal (wren) indicating the memory read/write accessing within full memory access period. ADDR_CHANGE_B indicates the changing of memory address. Byte mode signals RAM_LBE and RAM_UBE enable the lower byte and upper byte access, respectively. The detail timing diagrams are illustrated as below.

- **Memory read timing**

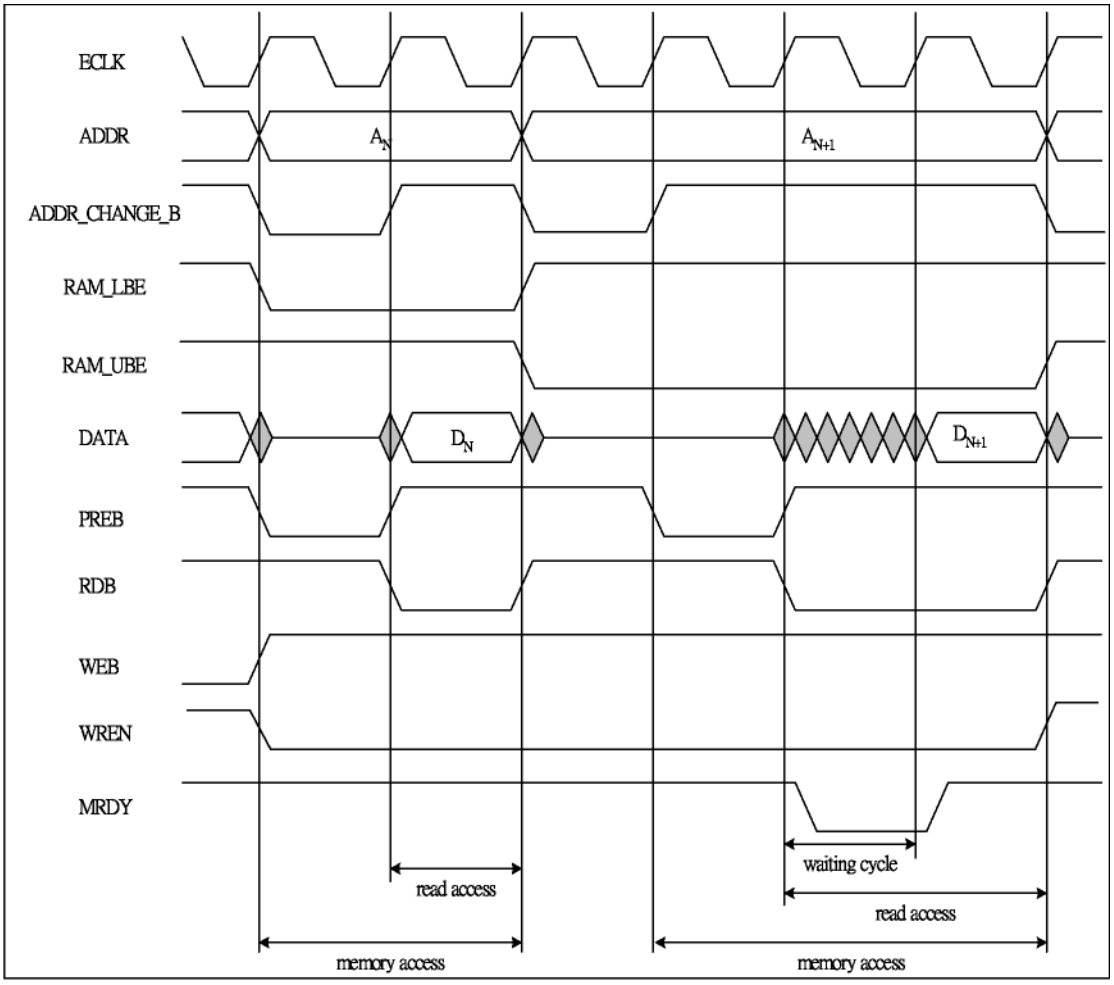


Figure 1.21

- **Memory write timing**

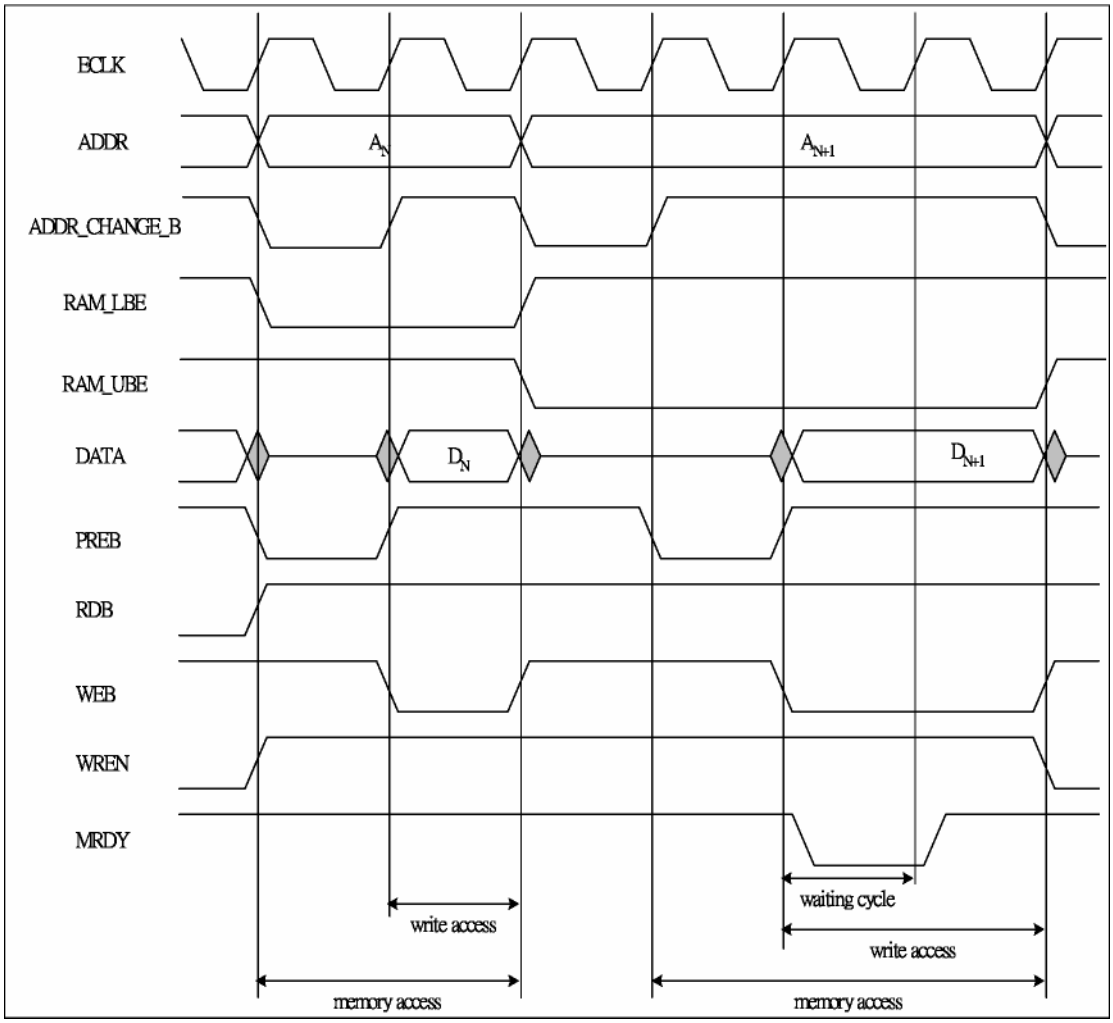


Figure 1.22

1.6.4 Memory Architecture of unSP-2.0

unSP 2.0 uses a modified Harvard architecture to accelerate memory access. The memory bus is separated into 2 parts, instruction and data bus. Program executing address is issued at instruction bus and data access address is issued at data bus.

The memory mapping of a real chip may be divided into several parts including internal memory, external memory, I/O memory, … etc. A memory controller customized by the user is need to manage the memory bus allocation.

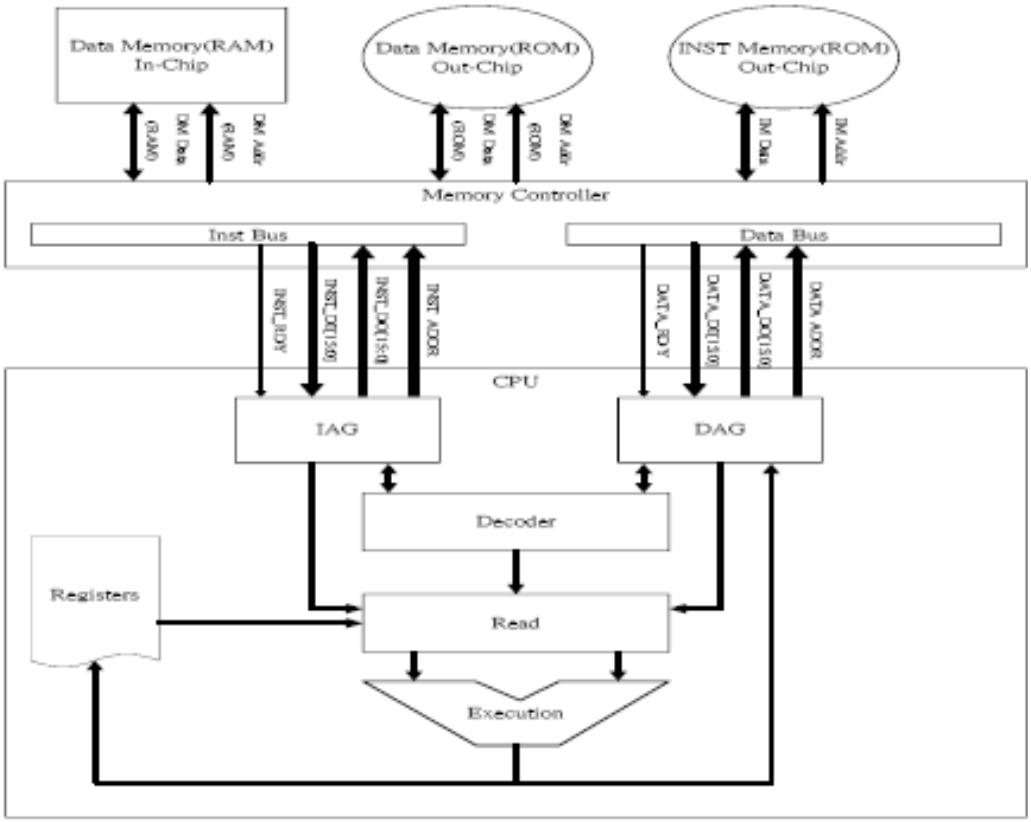


Figure 1.23

1.6.5 Memory Interface of unSP–2.0

- **Instruction Bus Read Timing**

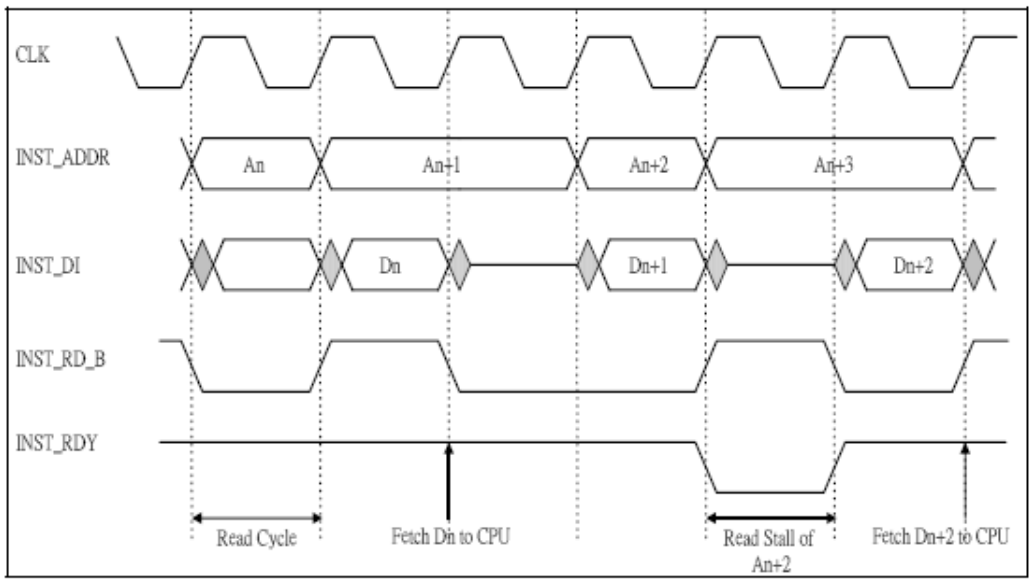


Figure 1.24

■ **Data Bus Read Timing**

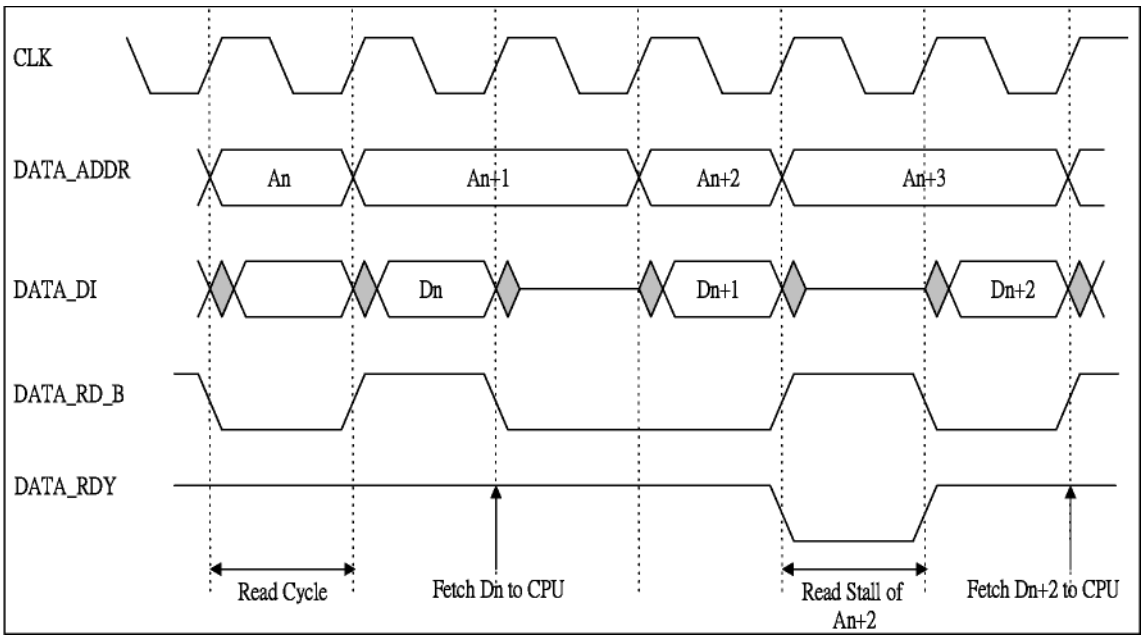


Figure 1.25

■ **Data Bus Write Timing**

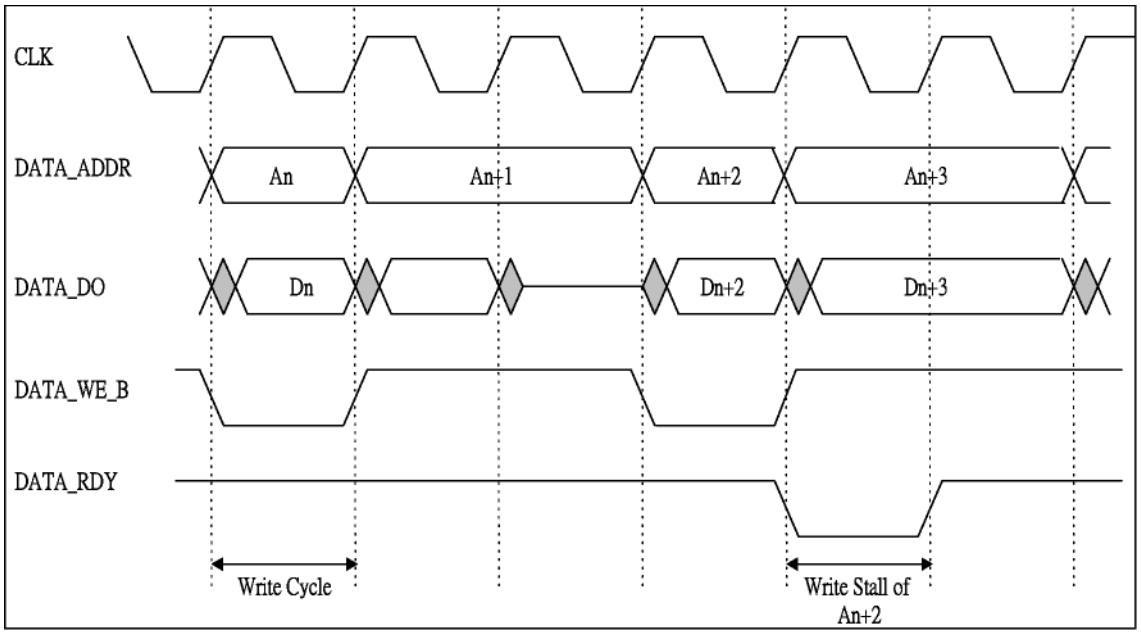


Figure 1.26

■ **INST Bus and Data Bus Access Conflict Timing**

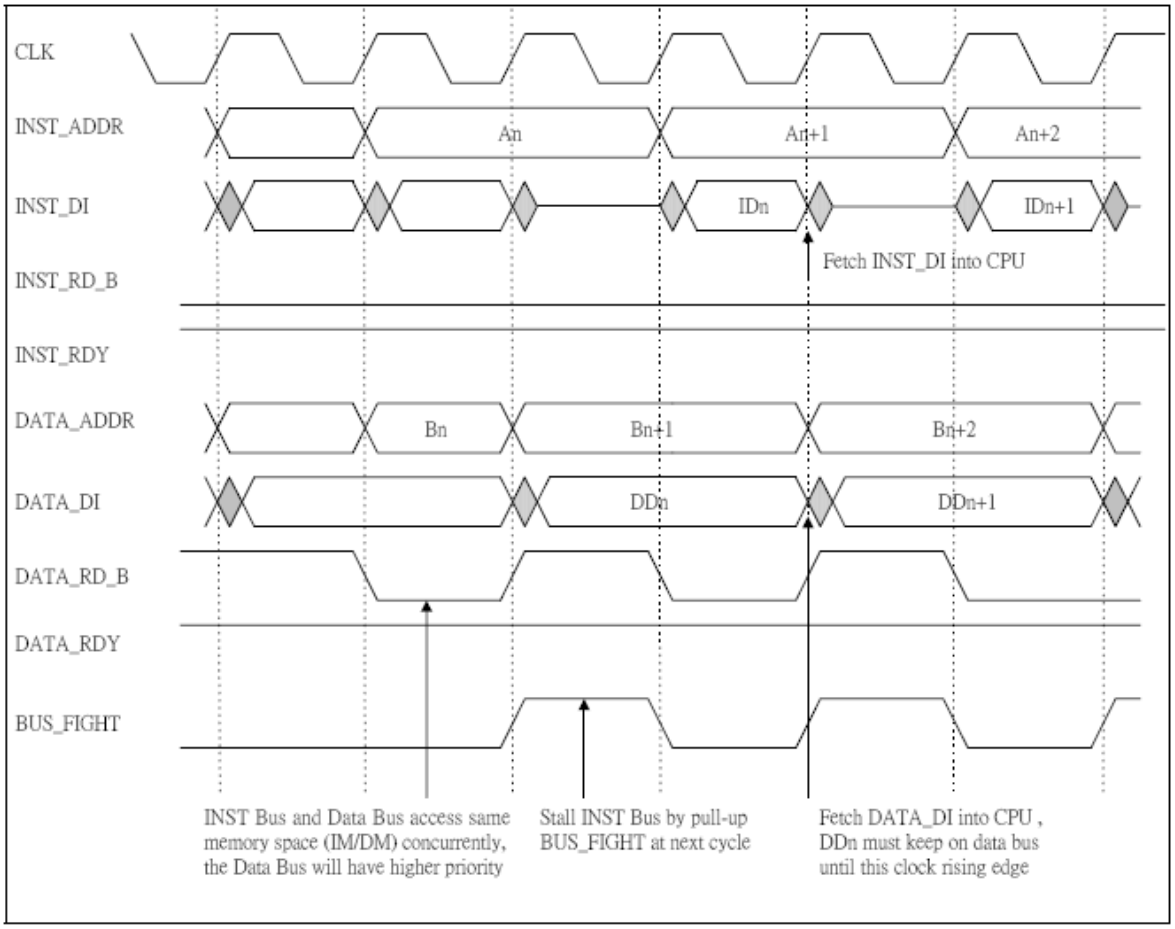


Figure 1.27

If INST Bus and Data Bus access the same memory space (IM/DM) concurrently, the Data Bus will have higher priority than INST Bus. Thus only the Data Bus memory access will be accepted and the memory content of DATA_ADDR is returned to the bus at next cycle.

The BUS_FIGHT signal will be pulled high to prevent the INST Bus from getting wrong data at next cycle. If BUS_FIGHT signal is placed at high then the DATA Bus will release memory bus and let INST Bus to get control of memory bus to prevent starvation condition.

CPU will fetch the data value on INST_DI and DATA_DI into internal registers at next clock rising edge while INST_RDY and DATA_RDY are high, BUS_FIGHT is low. So the data input must keep on bus until them being fetched into CPU.

- **MULS Timing (FIR_MOV OFF, Rd, Rs index to separate memory space)**

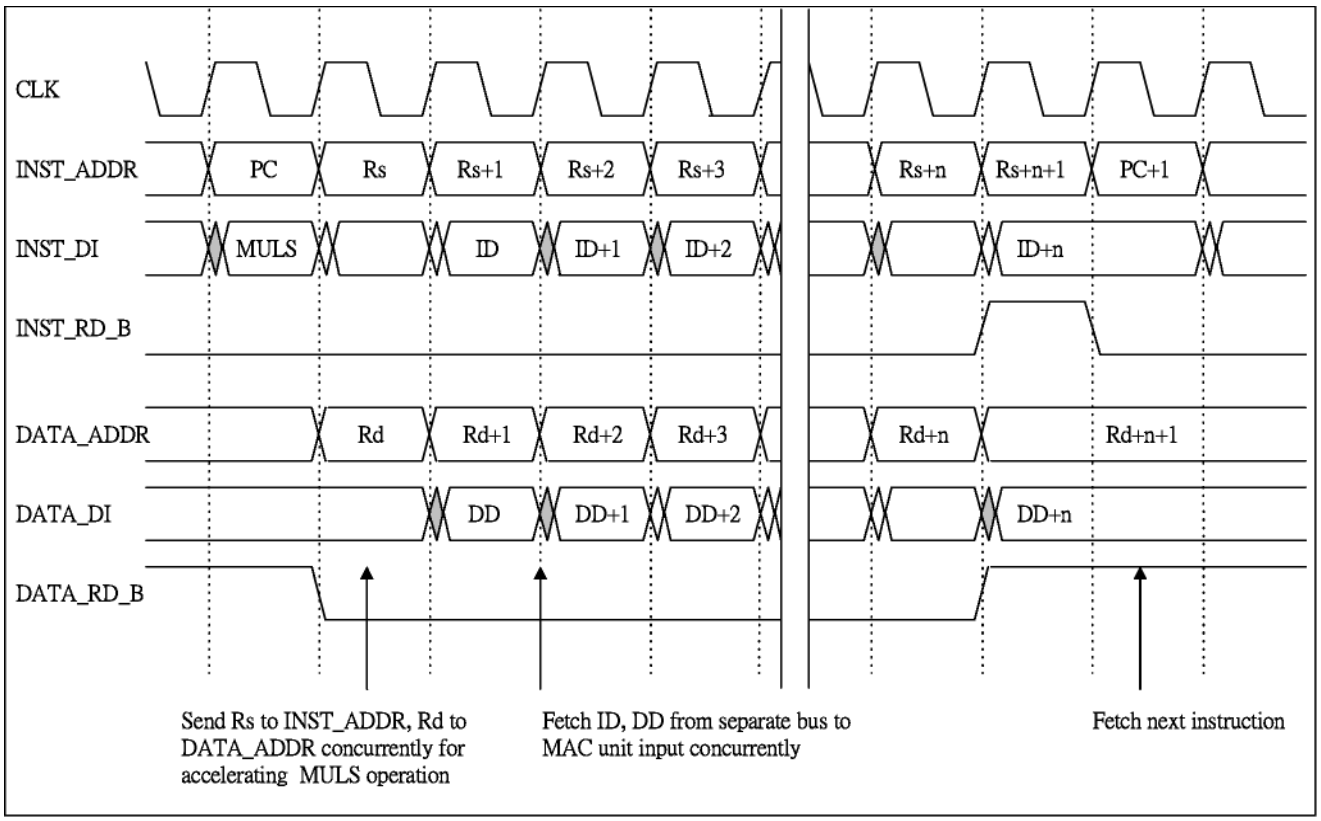


Figure 1.28

MULS operation will fetch data from INST Bus and DATA Bus concurrently to accelerating MAC operation.

If the parameters array location indexed by Rd, Rs are placed at different memory ranges (IM/DM), MULS

will have the best performance. Otherwise, bus conflict stall may be occurred and need 2 times of

executing cycles.

Cycles Count:

(No Bus Conflict, FIR_MOV

OFF): N+2 (No Bus Conflict,

FIR_MOV ON): 2N+1 (Bus

Conflict, FIR_MOV OFF):

2N+2 (Bus Conflict,

FIR_MOV ON): 3N

- **MULS Timing (FIR_MOV ON, Rd, Rs index to separate memory space)**

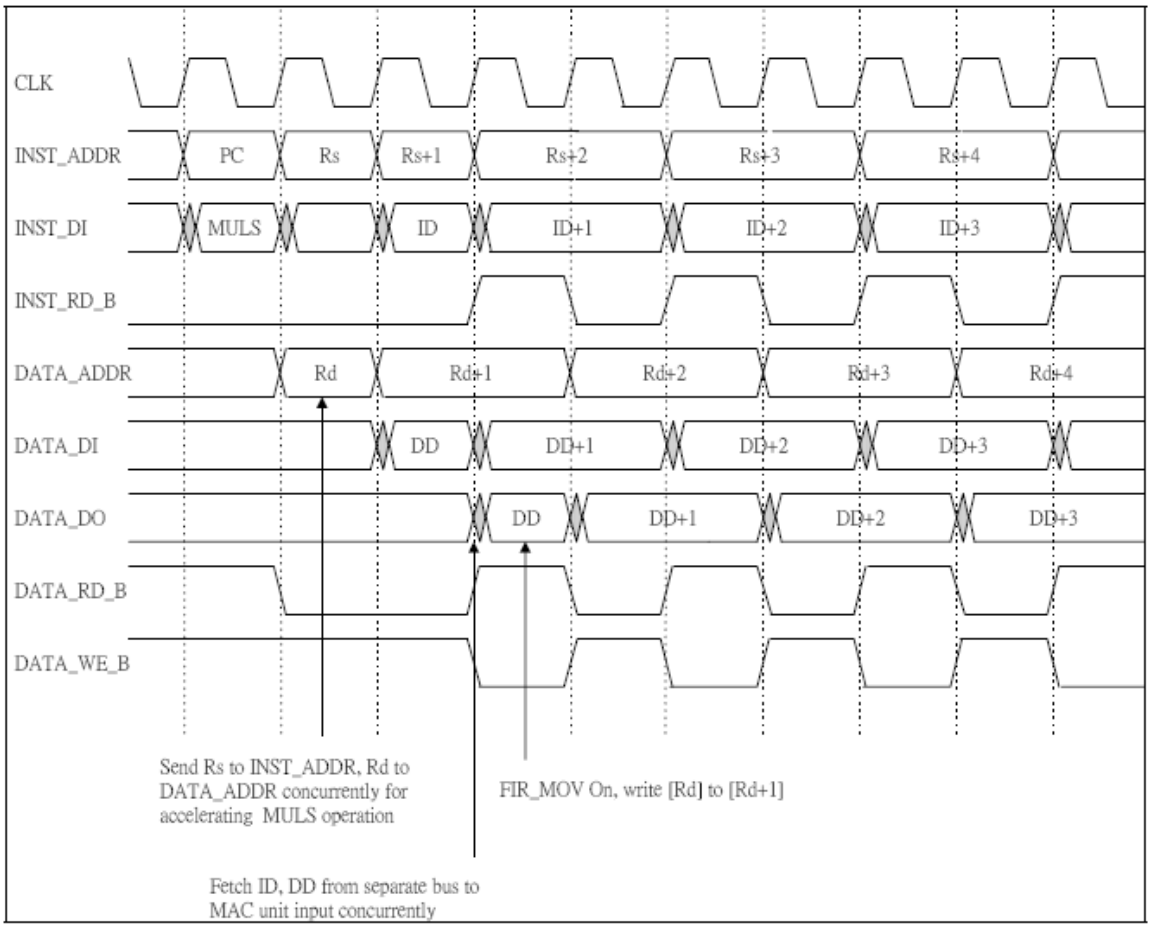


Figure 1.29

- **MULS Timing (FIR_MOV OFF, Rd, Rs index to same memory space)**

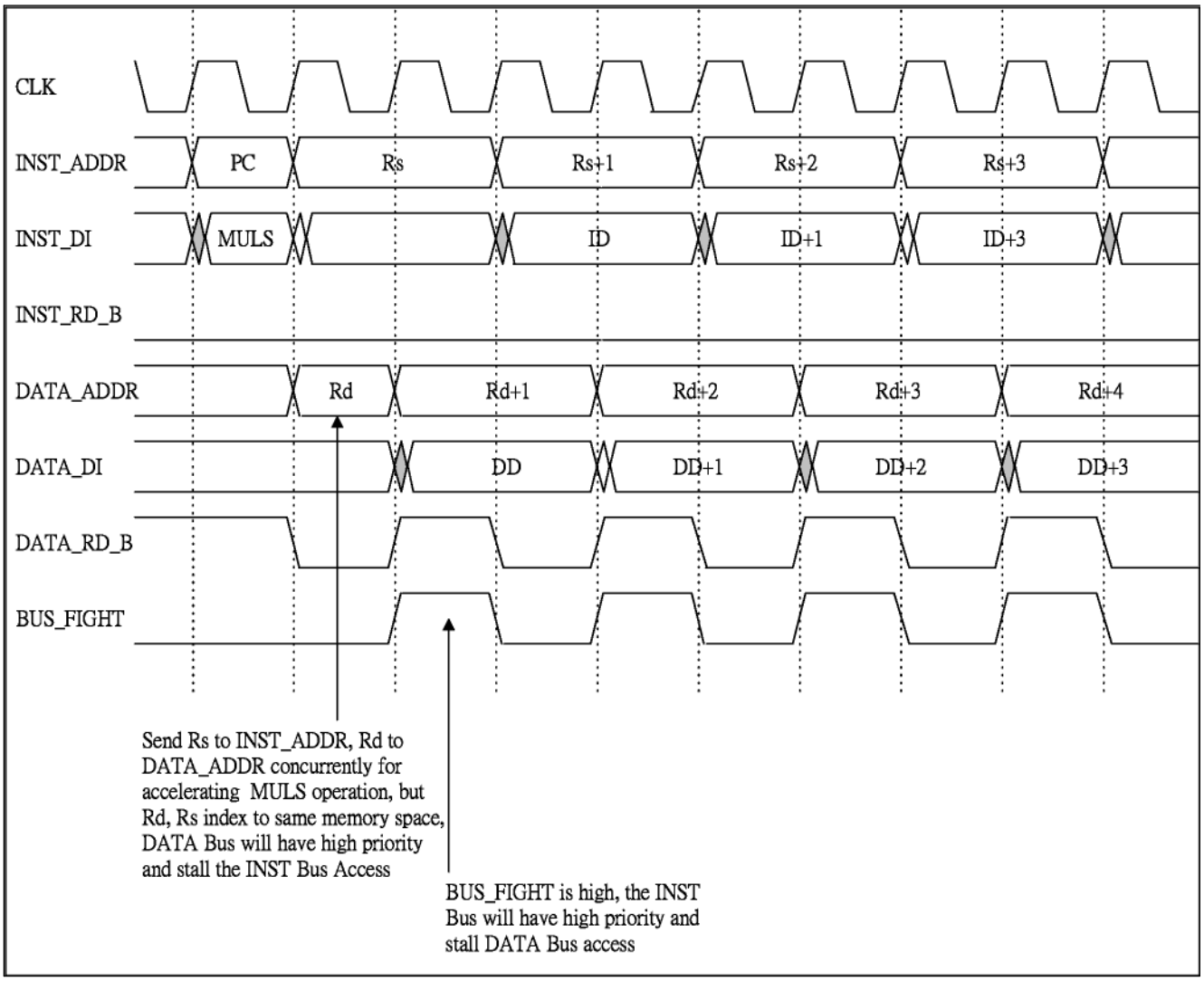

CLK
INST_ADDR
PC
Rs
Rs+1
Rs+2
Rs+3
INST_DI
MULS
ID
ID+1
ID+3
INST_RD_B
DATA_ADDR
Rd
Rd+1
Rd+2
Rd+3
Rd+4
DATA_DI
DD
DD+1
DD+2
DD+3
DATA_RD_B
BUS_FIGHT
Send Rs to INST_ADDR, Rd to DATA_ADDR concurrently for accelerating MULS operation, but Rd, Rs index to same memory space, DATA Bus will have high priority and stall the INST Bus Access
BUS_FIGHT is high, the INST Bus will have high priority and stall DATA Bus access

Figure 1.30

■ **MULS Timing (FIR_MOV ON, Rd, Rs index to same memory space)**

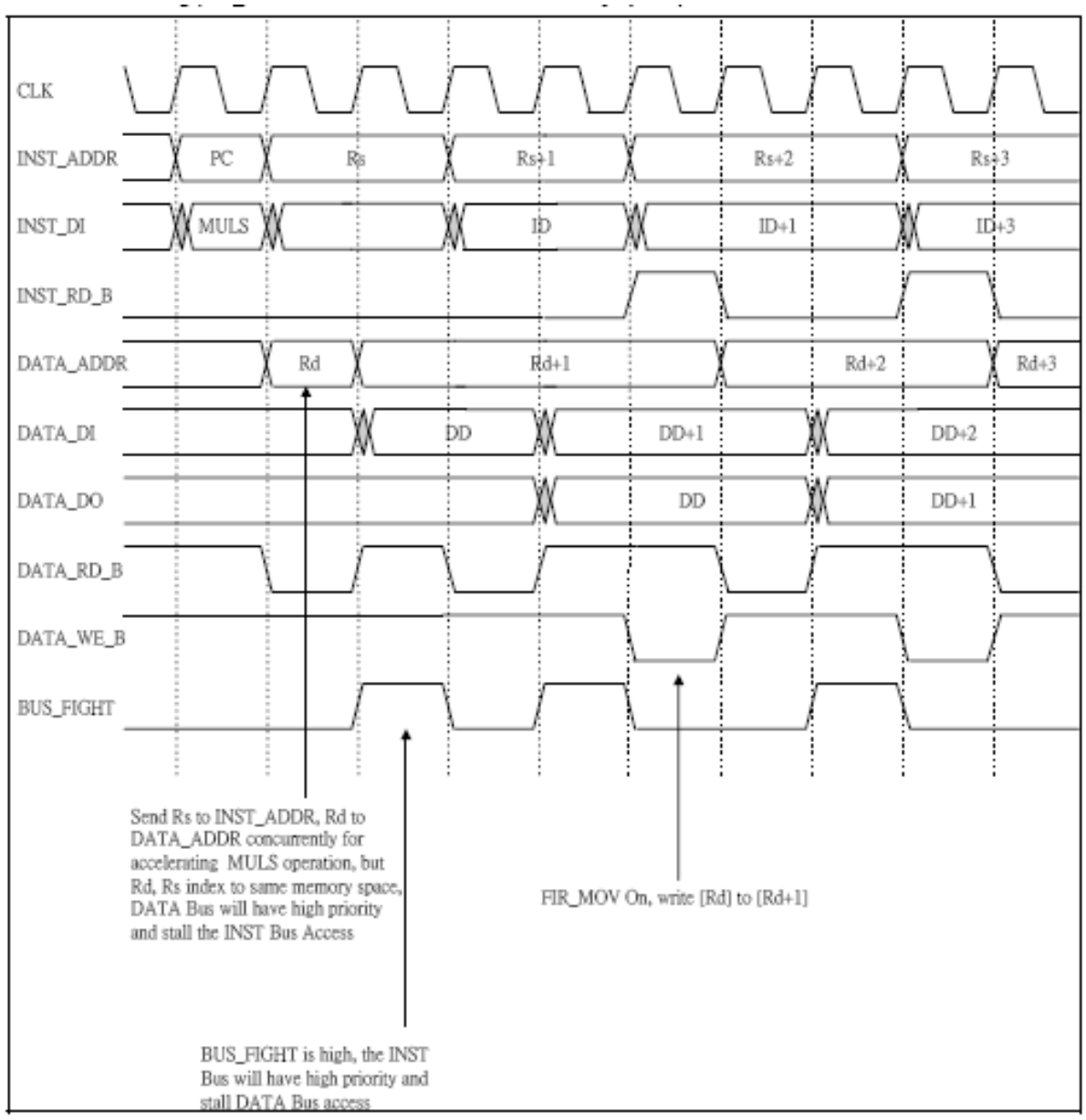


Figure 1.31

1.7 Addressing Modes

1.7.1 6 addressing modes of *unSP*-1.0 and *unSP*-1.1

In *unSP* 1.0 and *unSP* 1.1, performing the same operation on differently addressed operand may require different addressing modes. This indicates that the final destination address of operand can be derived from register, content in register or offset of address. The destination address formed by some calculations is called Effective Address (EA).

The effective address will be divided into three types according to its number of bit. That is 6-bit, 16-bit and 22-bit EA. The first two are offset of address. Only operand in current page can be addressed. Moreover, the 22-bit EA means all operands in the whole range of memory can be addressed. The *unSP* 1.0 and *unSP* 1.1 supports six addressing modes, in which 16-bit data operand or the address operand of transfer instruction can be accessed. In instruction set, most instructions can combine with these six addressing mode to generate an instruction subset.

- **Indexed Address**
  - Addressing space is limited to the memory in PAGE0 (0x000000-0x00FFFF) only
- **PC Relatively**
  - Program jumps to an address related to PC conditionally or unconditionally. The jumping range is limited to PC±63-word. The condition lies on NZSC flags in SR register.
  - In unSP1.1, the jumping range is limited to CS:PC±63-word.
- **Memory Absolute Address**

  Addressing space is limited to:

  (1) First 64 words (0x00 ~ 0x3F) in PAGE0

  (2) PAGE0 (0x000000~0x00FFFF)

  (3) Calling a sub-program in code segment of 64-page absolute address
- **Immediate**
  - The operand is IM6 (6-bit immediate value)
  - The operand is IM16 (16-bit immediate value)
- **Register Direct**

  The operand is in register directly
- **Register Indirect**
  - Addressing space in memory is limited to data segment in PAGE0 or 64-page addresses. Its offset depends on content in register and its page index on DS field of SR register.
  - Addressing space is limited in PAGE0. The offset depends on the content of register.
  - Using register indirect addressing mode in *unSP*1.1, the increment or decrement is the arithmetical operation of 22-bit value, formed by DS register and target register. For instance, suppose R1=0xFFFF. After executing D:[R1++], DS will be incremented by one and R1 becomes 0x0000.

1.7.2 6 addressing modes of *unSP*-1.2 and *unSP*-2.0

- **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-bit or a 16-bit 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-bit or 16-bit operand, then place the result at destination register.
  - Users can store register value to a memory location indexed by 6-bit or 16-bit 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-bit 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-bit displacement.

1.7.3 9 addressing modes of *unSP*-1.3

- **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 s ource register and a 6-bit or a 16-bit 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-bit or 16-bit operand, then place the result at destination register.
  - Users can store register value to a memory location indexed by 6-bit or 16-bit 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-bit of accessing address will use data segment (DS) value or be

zeroed.

- **Multi-indirect**

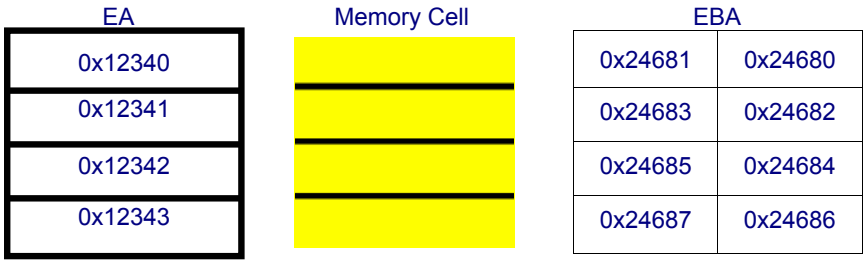
  Users can push or pop multiple registers' value to memory location indexed by stack pointer (SP)

- **Displacement**

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

- **Byte Register Indirect**

  - Users can do ALU operation between destination register and the value at memory location. The first character B or W indicates accessing one byte or word. The effective byte address (abbreviates to EBA) is $\{R_{x+1}:R_x\}_{22\sim0}$. The effective (word) address is $EBA_{22\sim1}$. If Rx_0 is 0, low byte in EBA22~1 is the target. If Rx_0 is 1, high byte in $EBA_{22\sim1}$ is the target.
  - When accessing one word and the least significant bit of EBA is 1, that is, accessing across word boundary, a software interrupt occurs and the AdE bit in FR register will be set. User should take care to prevent such unaligned access takes place. If the software interrupt occurs in the developing phase, user should debug their codes to remove unaligned access.



  - This addressing mode supports post increment, post increment and pre increment operations on $\{R_{x+1}:R_x\}_{22\sim0}$. For post increment, post increment and pre increment operations, the bits in $\{R_{x+1}:R_x\}_{31\sim23}$ (that is, $Rx+1_{15\sim7}$) are not affected.

- **Displacement**

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

- **Displacement**

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

1.8 Interrupts

1.8.1 Interrupts of *unSP*-1.0 and *unSP*-1.1

unSP 1.0 and *unSP* 1.1 accept two types of external interrupts: Fast Interrupt (FIQ) and Interrupt

(IRQ). Both interrupts can be freely turned on or off. In addition, *unSP* 1.0 and *unSP* 1.1 also implements a software interrupt, BREAK. The interrupt vector mappings and priorities are depicted as follow:

Interrupt Vector		Interrupt Priority	
0xFFF5	BREAK	1. Reset (highest)	* If more than two IRQs occurred simultaneously, the priority is from IRQ0 down to UART IRQ. That is, IRQ0 is the highest and UART IRQ is the lowest. However, if a lower priority IRQ occurred first, even a higher priority IRQ can not interrupt the current IRQ. For example, if IRQ4 is occurred first, IRQ3 is unable to interrupt IRQ4. The priority applies only when two IRQs occurred concurrently.
0xFFF6	FIQ	2. FIQ	
0xFFF7	RESET	3. IRQ0 ~ 6, UART IRQ*	
0XFFF8	IRQ0	4. BREAK** (lowest)	
0XFFF9	IRQ1		
0XFFFA	IRQ2		
0XFFFB	IRQ3		
0XFFFC	IRQ4		
0XFFFD	IRQ5		
0XFFFE	IRQ6		
0XFFFF	UART IRQ		**The action of "BREAK" is the same as "CALL" except "BREAK" will jump to the fixed address specified at 0xFFF5.

Figure 1.32

1.8.2 Interrupts of *unSP*-1.2

Interrupts are used to handle exception while program is running. *unSP* 1.2 support 10 interrupt sources and 1 reset request. When an exception arises, *unSP* 1.2 completes the current instruction and then departs from the current instruction sequence to handle the exception. The following sequence of actions will be taken by processor before entering service routine.

- According the interrupt priorities to choose the highest priority event. The interrupt priority is showed as below:
  - RESET > BREAK > FIQ > IRQ0 > IRQ1 > IRQ2 > IRQ3 > IRQ4 > IRQ5 > IRQ6 > IRQ7
  - If IRQ_NEST mode is off and program running in IRQ service routine, only REST, BREAK and FIQ event can interrupt CPU.
  - If IRQ_NEST mode is on and program running in IRQ service routine, besides RESET, BREAK, FIQ event, the IRQ events with priority greater than PRI register also can interrupt CPU.
- Fetch the relevant vector address list below into CPU.

Table 1.6

Interrupts	TEST[1:0]			
	00	01	10	11

Interrupts	TEST[1:0]			
	00	01	10	11
BREAK	0x00FFF5	0x00FFE5	0x007FF5	0x007FE5
FIQ	0x00FFF6	0x00FFE6	0x007FF6	0x007FE6
RESET	0x00FFF7	0x00FFE7	0x007FF7	0x007FE7
IRQ0	0x00FFF8	0x00FFE8	0x007FF8	0x007FE8
IRQ1	0x00FFF9	0x00FFE9	0x007FF9	0x007FE9
IRQ2	0x00FFFA	0x00FFEA	0x007FFA	0x007FEA
IRQ3	0x00FFFB	0x00FFEB	0x007FFB	0x007FEB
IRQ4	0x00FFFC	0x00FFEC	0x007FFC	0x007FEC
IRQ5	0x00FFFD	0x00FFED	0x007FFD	0x007FED
IRQ6	0x00FFFE	0x00FFEE	0x007FFE	0x007FEE
IRQ7	0x00FFFF	0x00FFEF	0x007FFF	0x007FEF

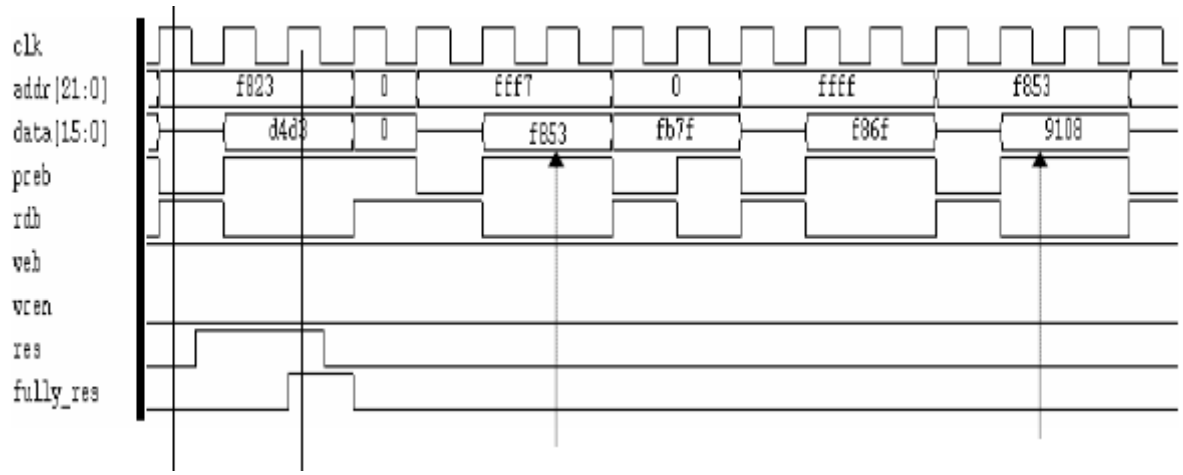
- If the interrupt event is IRQ and IRQ_NEST mode is on, *unSP w*ill save PC, SR and FR into memory stack indexed by SP. If IRQ_NEST mode is off, only PC and SR will be saved.
- *unSP* 1.2 will change PC as the address fetched by vector address and fetching the first instruction.
- If the interrupt event is IRQ and IRQ_NEST mode is on, the PRI register will be changed as the IRQS value.

The leaving sequence of service routine.

- If CPU is servicing IRQ interrupt and IRQ_NEST mode is on, the FR, SR and PC will be restored from memory stack indexed by SP. If IRQ_NEST mode is off, only SR and PC will be restored. Since the values in these registers are changed by the restore operation, CPU will return to the program status before interrupt and keep executing.

The interrupt timing diagrams are illustrated as below:

- **Reset timing**



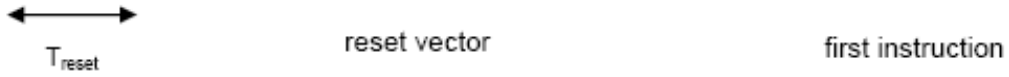


Figure 1.33

- After CPU acknowledge reset signal, the first instruction will be fetched at 10th clock cycle. Program reset vector must place at address 0xFFF7.
- Res: CPU reset signal, active low, Treset Pulse width must keep at least 2 clock cycles for CPU to acknowledge reset pulse.
- fully_res: CPU internal reset signal.

■ **Entering interrupt service routine**

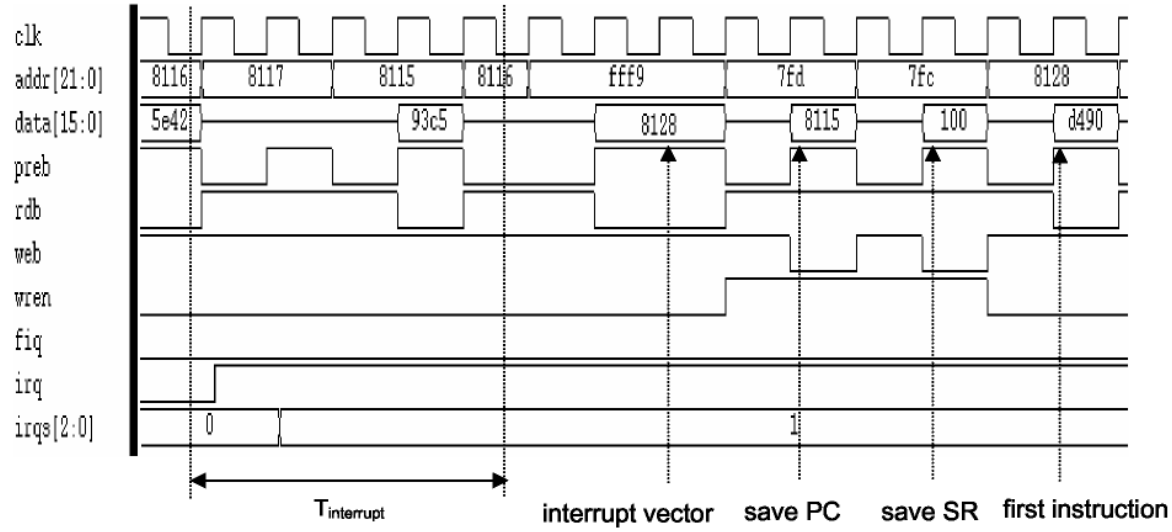


Figure 1.34

■ **Leaving interrupt service routine**

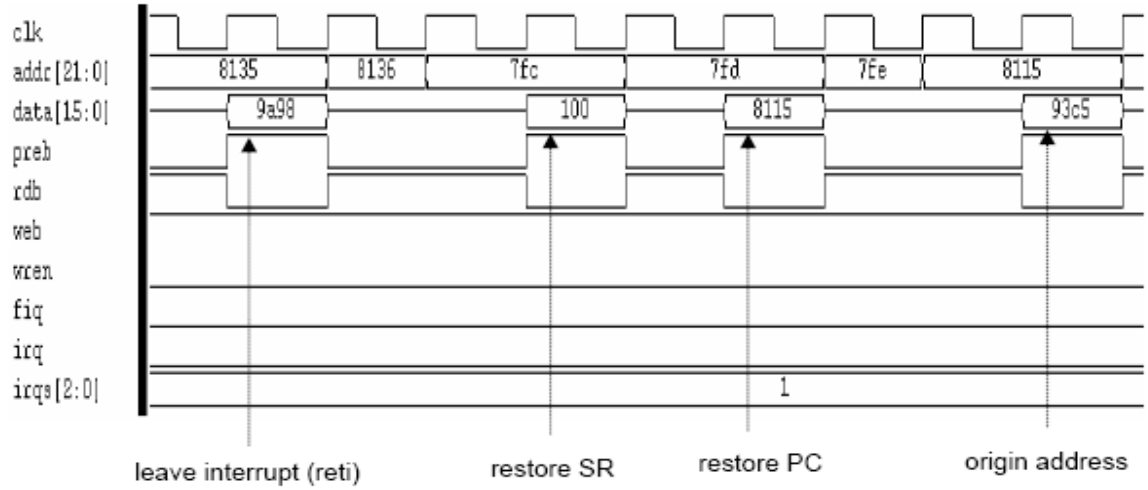


Figure 1.35

- 8 clock cycles needed from CPU accept interrupt signal to get the first instruction of interrupt service routine.
- The longest delay from interrupt signal rising to acknowledge by CPU Tinterrupt <= 182 clock cycles (max instruction executing cycle)

■ **Entering interrupt service routine**

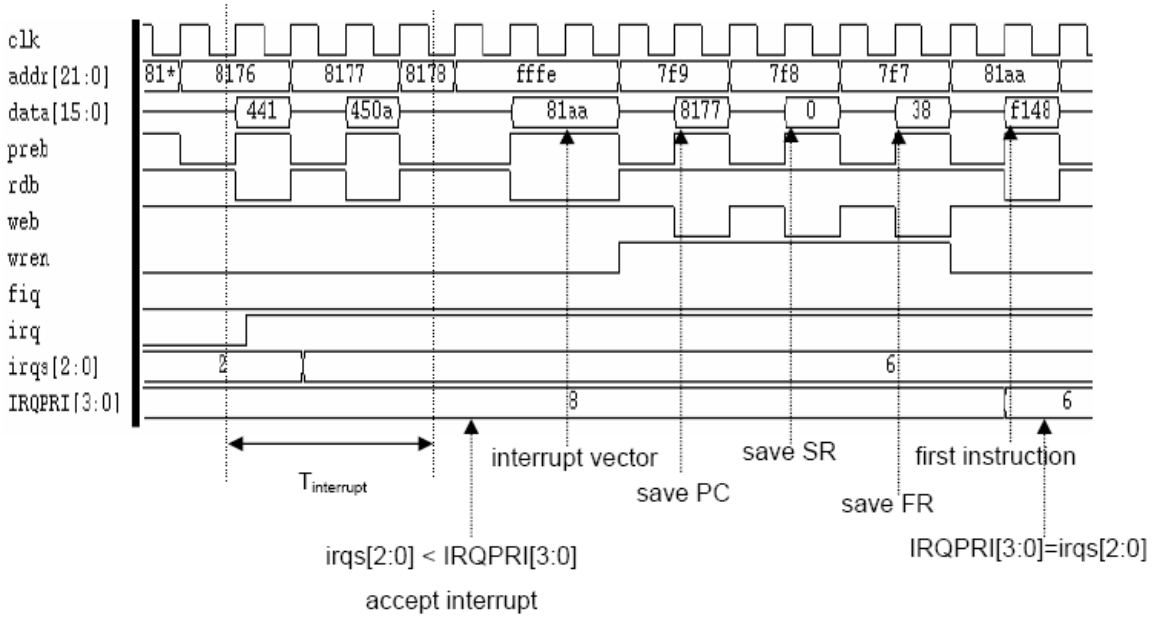
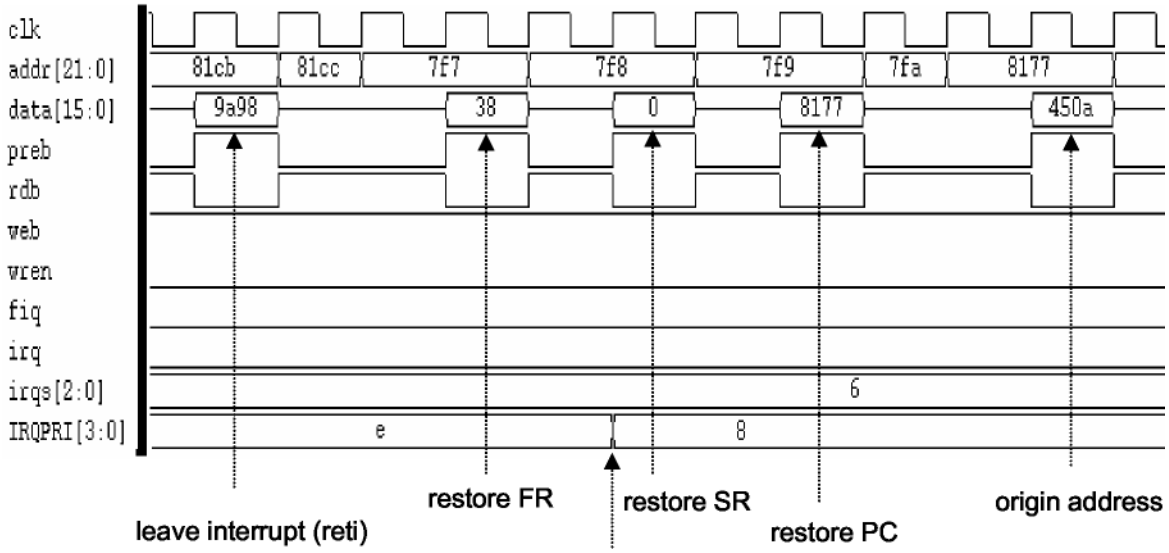


Figure 1.36

■ **Leaving interrupt service routine**



restore IRQPRI[3:0]

Figure 1.37

- Irqs[2:0]: External interrupt source select pins.
- RQPRI[3:0]: Interrupt priority register, user can change its value to allow specify range of interrupts can be accepted by CPU. After entering interrupt service routine, the IRQPRI register will be change to current IRQ number.

1.8.3 Interrupts of *unSP*-1.3

Interrupts are used to handle exception while program is running. *unSP* 1.3 support 10 interrupt sources and 1 reset request. When an exception arises, CPU will complete the current instruction and then departs from the current instruction sequence to handle the exception. The following sequence of actions will be
taken by processor before entering service routine.

- According the interrupt priorities to choose the highest priority event. The interrupt priority is showed as below
  1. RESET > BREAK > FIQ > IRQ0 > IRQ1 > IRQ2 > IRQ3 > IRQ4 > IRQ5 > IRQ6 > IRQ7
  2. IRQ_NEST mode is always on. When program is running in IRQ service routine, besides RESET, BREAK, FIQ event, the IRQ events which priority greater than PRI register also can interrupt CPU.
- Fetch the relevant vector address list below into CPU.

Table 1.7

Interrupts	Vector Address
BREAK	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h0, 3'h5}
FIQ	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h0, 3'h6}
RESET	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h0, 3'h7}
IRQ0	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h0}
IRQ1	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h1}
IRQ2	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h2}
IRQ3	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h3}
IRQ4	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h4}
IRQ5	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h5}
IRQ6	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h6}
IRQ7	**{** ~TEST[1], INT_BASE[9:0], ~TEST[0], 1'h1, 3'h7}

- If the interrupt event is IRQ or FIQ, *unSP*1.3 will save PC, SR and FR into memory stack indexed by {SS, SP}. For RESET and BREAK, only PC and SR will be saved.
- *unSP* 1.3 will change PC as the address fetched by vector address and fetching the first instruction.
- If the interrupt event is IRQ, the PRI register will be changed as the IRQ's value.
- IRQ_ENABLE is turned off automatically when *unSP* 1.3 performing IRQ service routine. User can

turn on IRQ_ENABLE in IRQ service routine to allow higher priority IRQ to interrupt it.

- *unSP* 1.3 can re-execute FIQ service routine when serving FIQ if FIQ_ENABLE is on. Both FIQ_ENABLE and IRQ_ENABLE are turned off automatically when *unSP*1.3 performing FIQ service routine.
- *unSP*1.3 will check the interrupt signals (FIQ/IRQ) at the last cycle of every instruction **except:**
  - RA16 (Direct16 instruction with read) for semaphore implementation of the operating system
  - RETI instruction
  - MDS access instruction

The leaving sequence of service routine.

- If CPU is servicing IRQ or FIQ, the FR, SR, PC will be restored from memory stack indexed by SP else only SR, PC will be restored. CPU will return to the program status before interrupt and keep executing.

The interrupt timing diagrams are illustrated as below.

- **Reset timing**

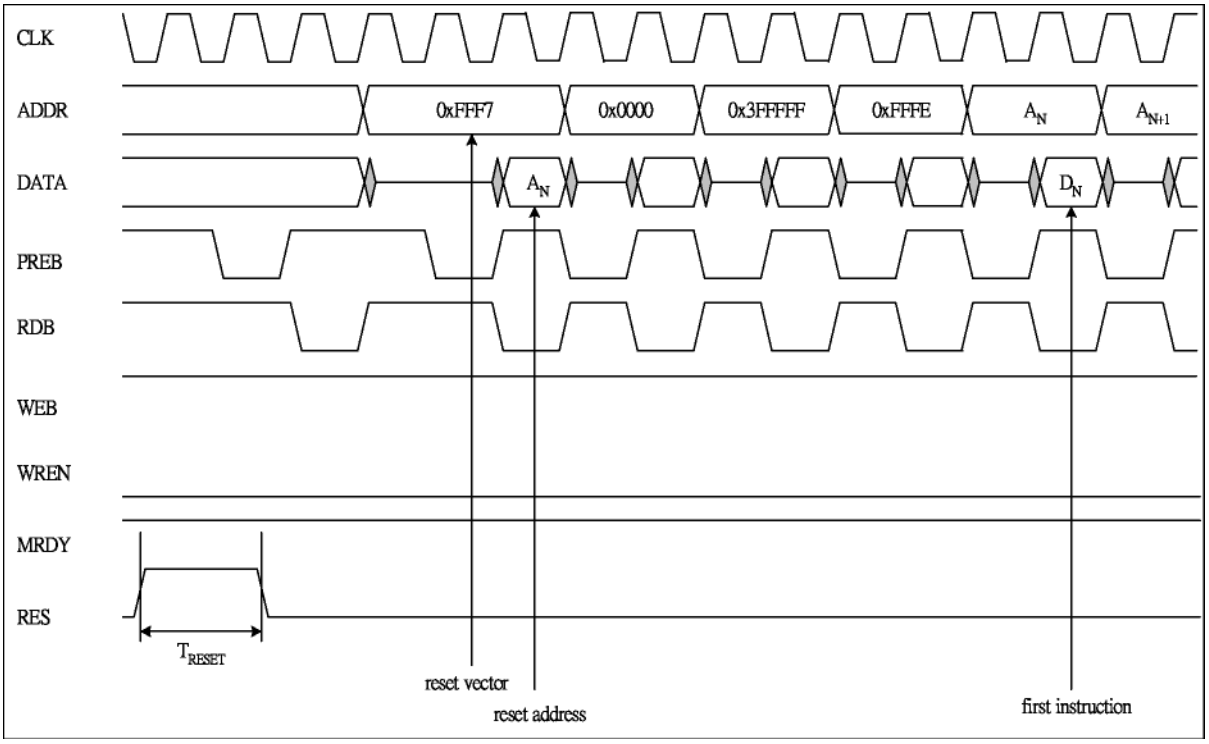


Figure 1.38

- After CPU acknowledge reset signal, the first instruction will be fetched at 12th clock cycle.
- program reset vector must place at address 0xfff7
- RES : CPU reset signal, active high, T_{RESET} pulse width must keep at least 2 clock cycles

- **Entering interrupt service routine**

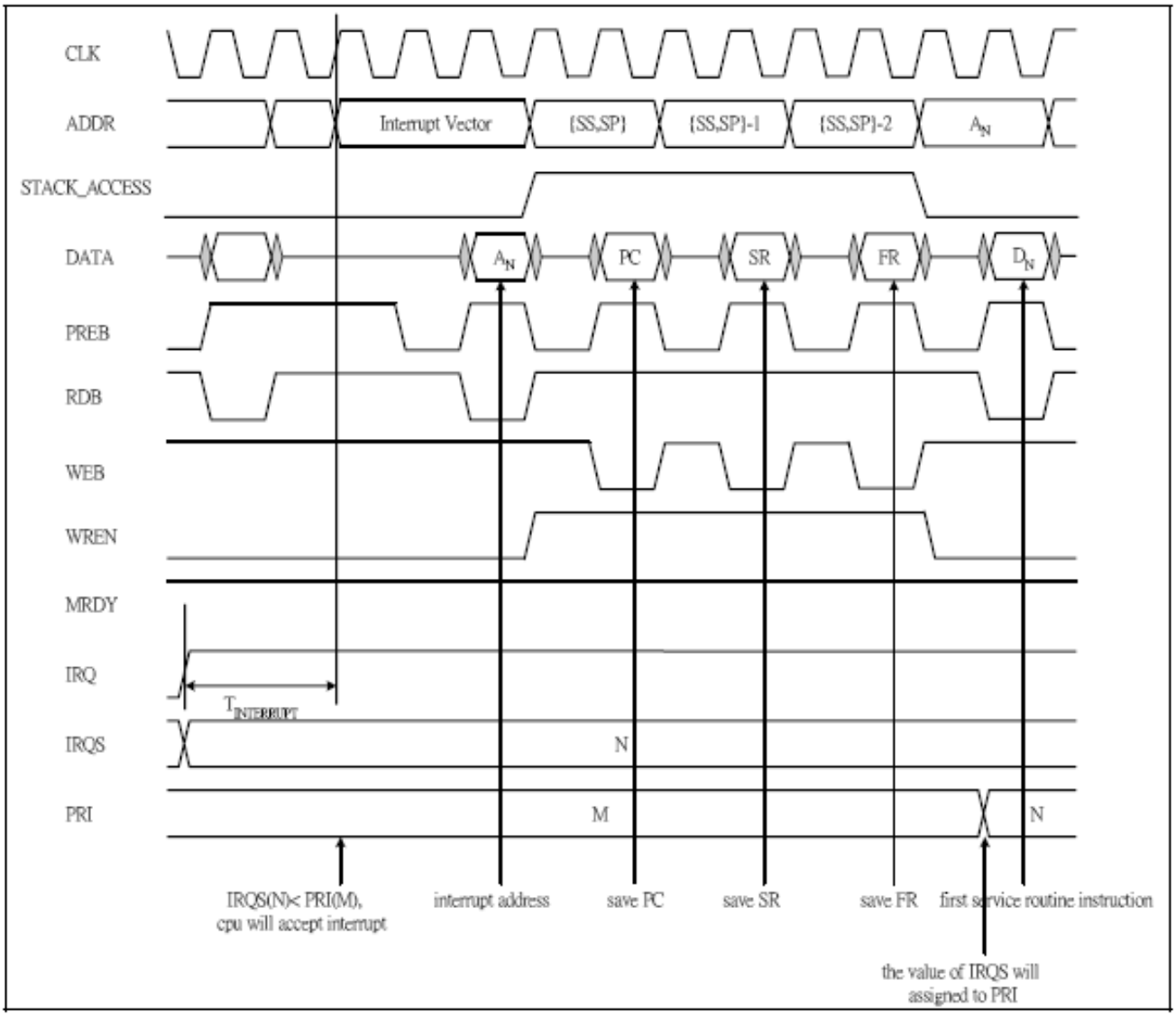


Figure 1.39

- 10 clock cycles needed from CPU accept interrupt signal to get the first instruction of interrupt service routine.
- The longest delay from interrupt signal rising to acknowledge by CPU $T_{INTERRUPT}$ <= 182 clock cycles (max instruction executing cycle)
- IRQS[2:0]: External interrupt source select pins.
- PRI[3:0]: Interrupt priority register, user can change its value to allow specify range of interrupts can be accepted by CPU. After entering interrupt service routine, the IRQPRI register will be change to current IRQ number.

■ **Leaving interrupt service routine**

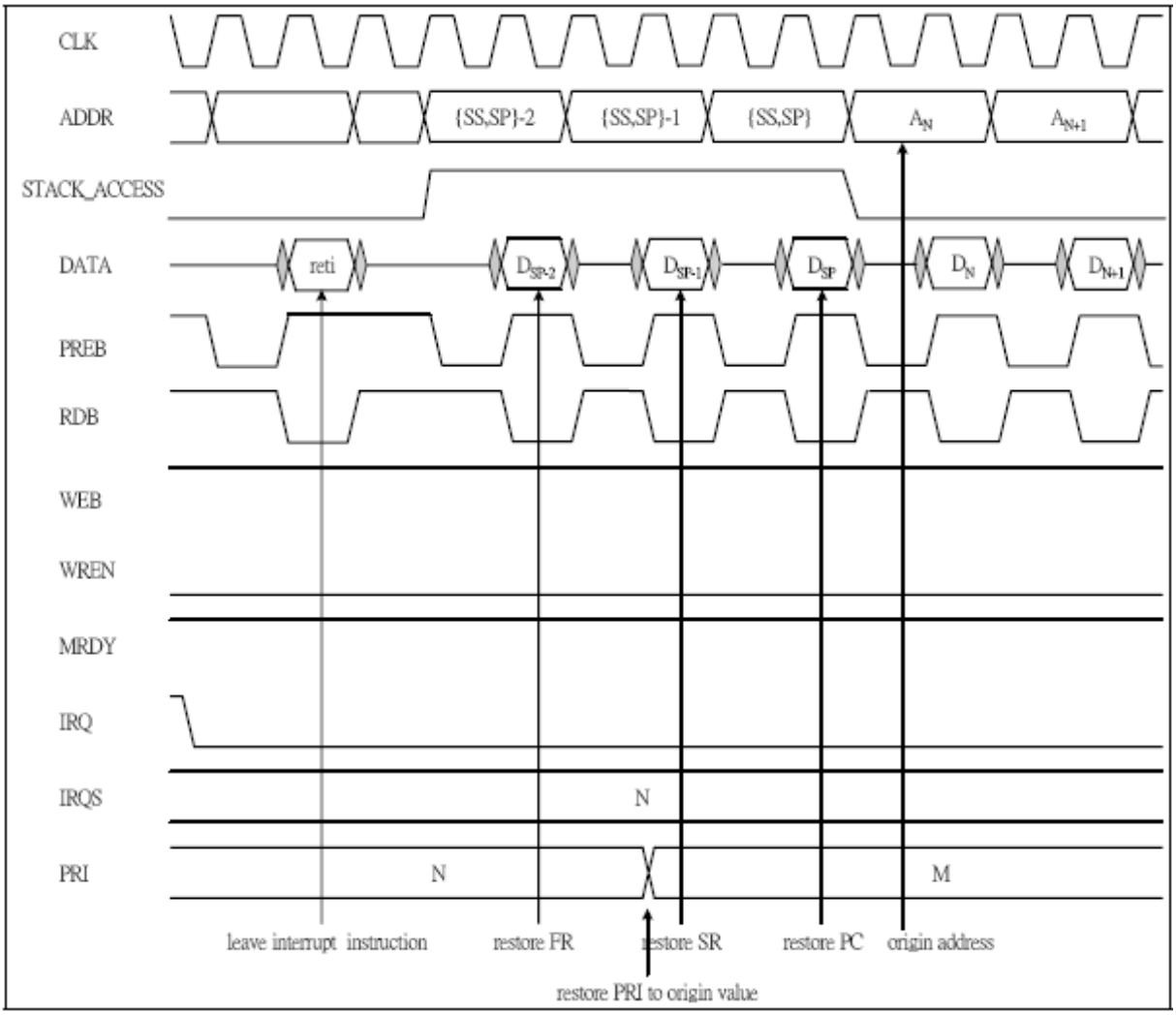


Figure 1.40

1.8.4 Interrupts of *unSP*-2.0

unSP 2.0 supports 10 interrupt sources and 1 reset request. When an exception arises, *unSP* 2.0 completes the current instruction and then departs from the current instruction sequence to handle the exception. The following sequence of actions will be taken by processor before entering service routine.

- According the interrupt priorities to choose the highest priority event. The interrupt priority is shown as below:
  - RESET> BREAK > FIQ > IRQ0 > IRQ1 > IRQ2 > IRQ3 > IRQ4 > IRQ5 > IRQ6 > IRQ7
  - If IRQ_NEST mode is off and program running in IRQ service routine, only RESET BREAK, FIQ event can interrupt CPU.
  - If IRQ_NEST mode is on and program running in IRQ service routine, besides RESET, BREAK, FIQ event, the IRQ events which priority greater than PRI register also can interrupt CPU.

- Fetch the relevant vector address list below into CPU.

Table 1.8

Interru pts	TEST[1:0]			
	00	01	10	11
BREAK	0x00FFF5	0x00FFE5	0x007FF5	0x007FE5
FIQ	0x00FFF6	0x00FFE6	0x007FF6	0x007FE6
RESET	0x00FFF7	0x00FFE7	0x007FF7	0x007FE7
IRQ0	0x00FFF8	0x00FFE8	0x007FF8	0x007FE8
IRQ1	0x00FFF9	0x00FFE9	0x007FF9	0x007FE9
IRQ2	0x00FFFA	0x00FFEA	0x007FFA	0x007FEA
IRQ3	0x00FFFB	0x00FFEB	0x007FFB	0x007FEB
IRQ4	0x00FFFC	0x00FFEC	0x007FFC	0x007FEC
IRQ5	0x00FFFD	0x00FFED	0x007FFD	0x007FED
IRQ6	0x00FFFE	0x00FFEE	0x007FFE	0x007FEE
IRQ7	0x00FFFF	0x00FFEF	0x007FFF	0x007FEF

  - If the interrupt event is IRQ and IRQ_NEST mode is on, *unSP*2.0 will save PC, SR, FR into memory stack indexed by SP else only PC, SR will be saved.
  - *unSP* 2.0 will change PC as the address fetched by vector address and fetching the first instruction.
  - If the interrupt event is IRQ and IRQ_NEST mode is on, the PRI register will be changed to current IRQ number.

  The leaving sequence of service routine.

  - If CPU is servicing IRQ interrupt and IRQ_NEST mode is on, the FR, SR, PC will be restored from memory stack indexed by SP else only SR, PC will be restored, and CPU will return to the program status before interrupt and keep executing.

  The interrupt timing diagrams are illustrated as below:

- **RESET Timing**

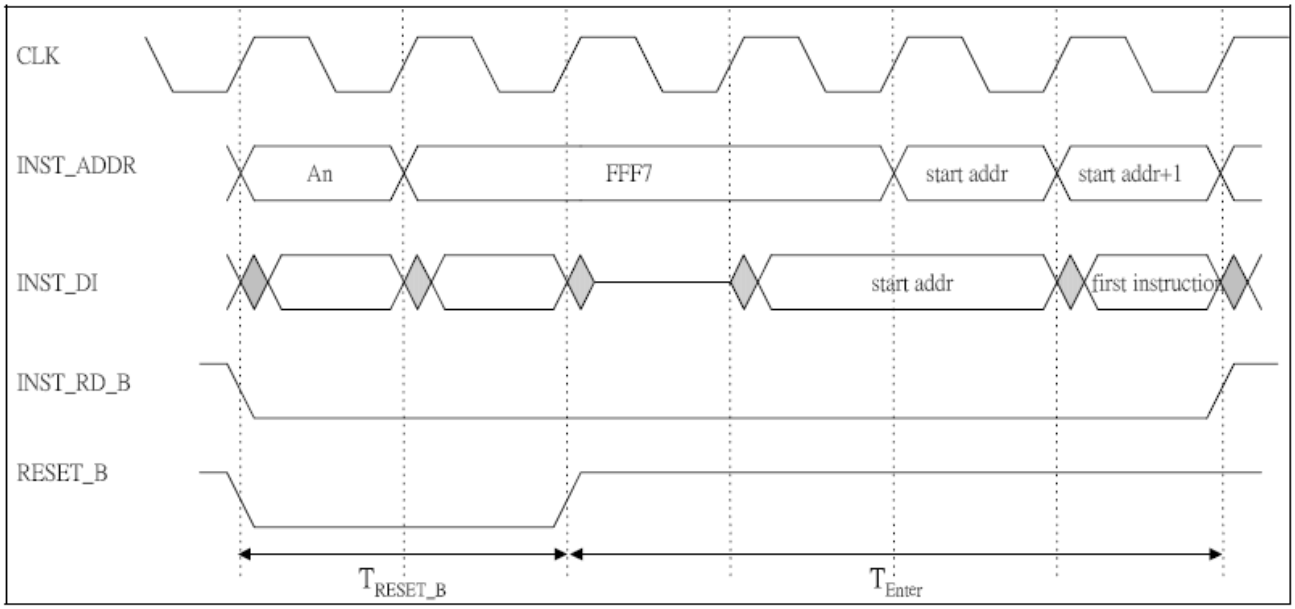


Figure 1.41

T_{RESET_B}: External reset signal, active low, T_{RESET_B} pulse width must keep at least 2 clock cycles for CPU

to acknowledge reset pulse.

T_{Enter}: Reset timing, 4 cycles needed from CPU accept reset signal to fetch the first instruction.

■ **Break Timing**

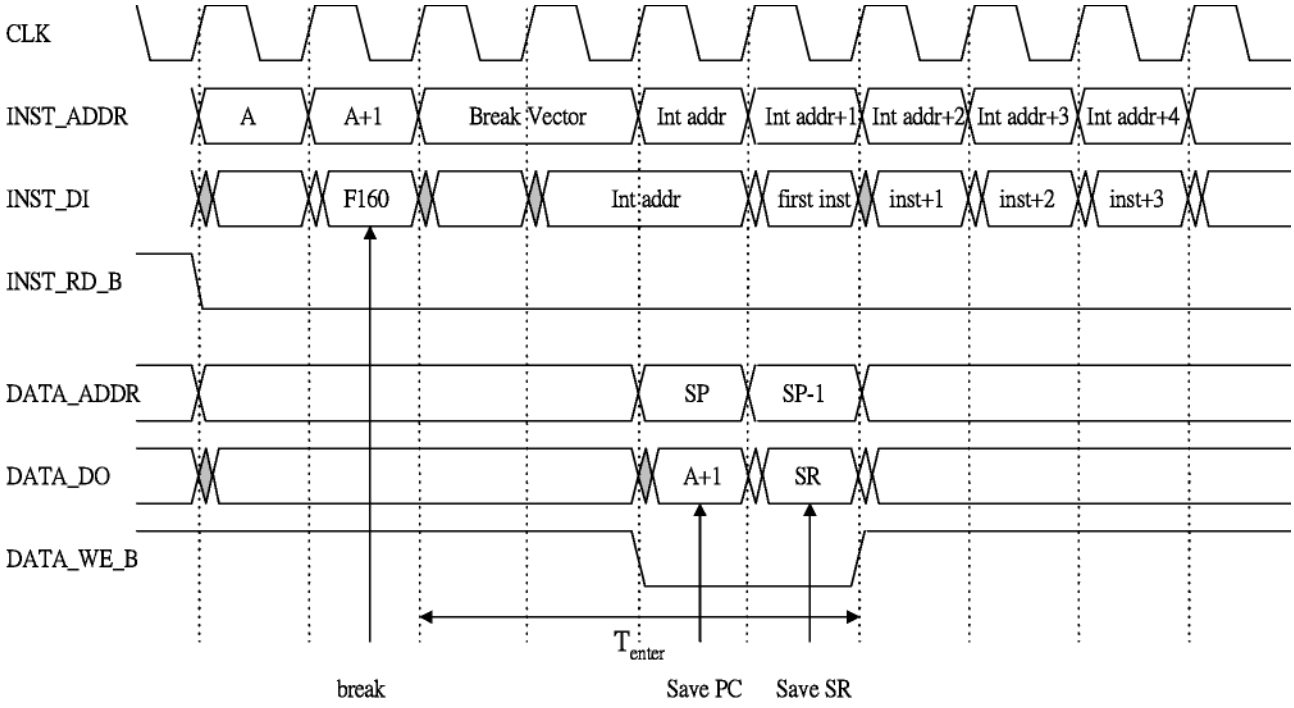


Figure 1.42

T_{enter} : Interrupt entering time, 4 cycles needed from CPU accept interrupt request to fetch the first instruction.

- **Entering interrupt service routine**

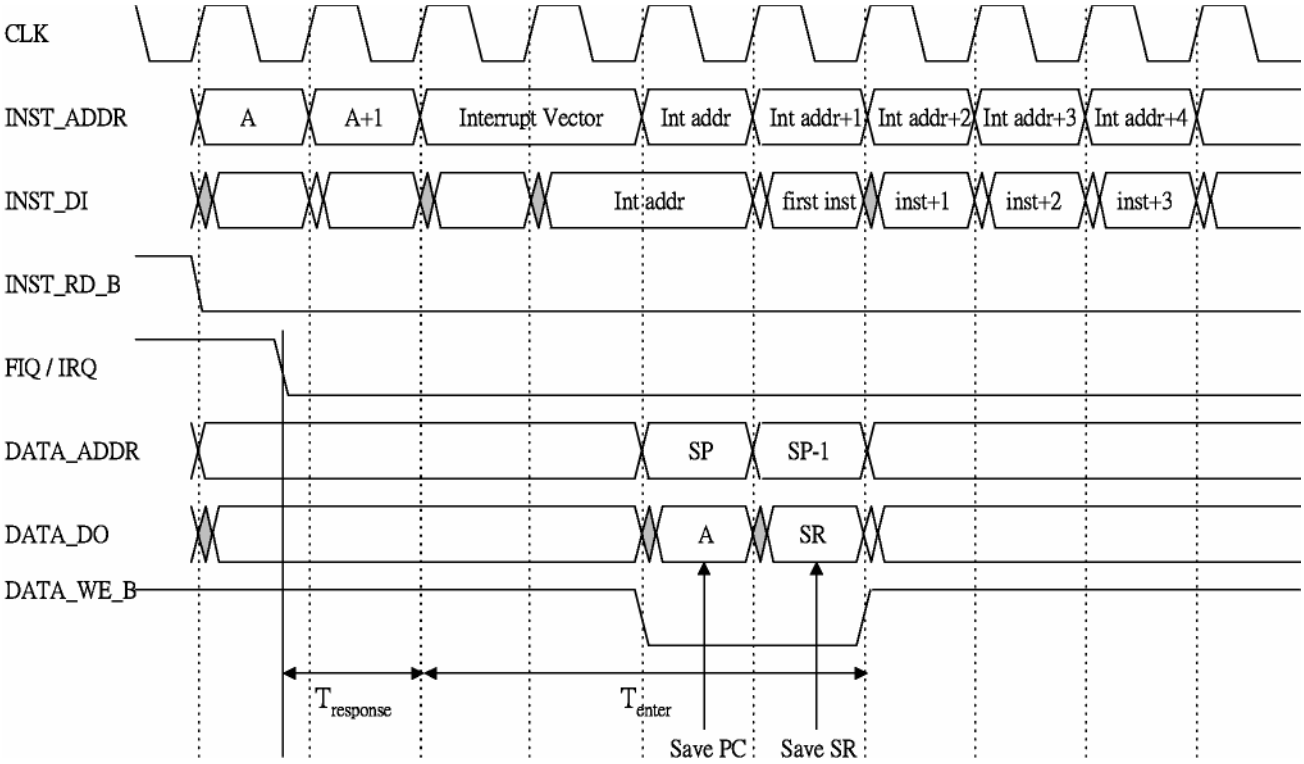


Figure 1.43

$T_{respose}$: Interrupt Response Time, $T_{respose}$ <= 50 clock cycles (max instruction executing cycles, MULS) T_{enter} : Interrupt entering time, 4 cycles needed from CPU accept interrupt request to fetch the first instruction.

- **Leaving interrupt service routine**

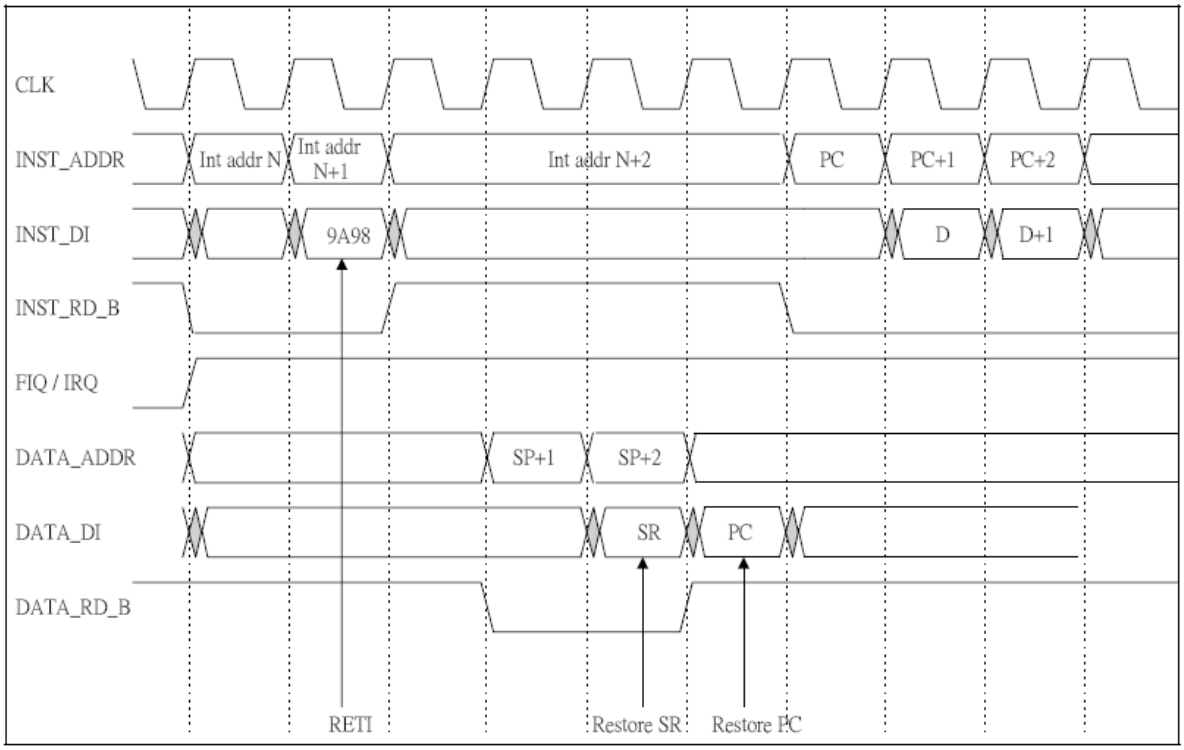


Figure 1.44

- **Entering interrupt service routine**

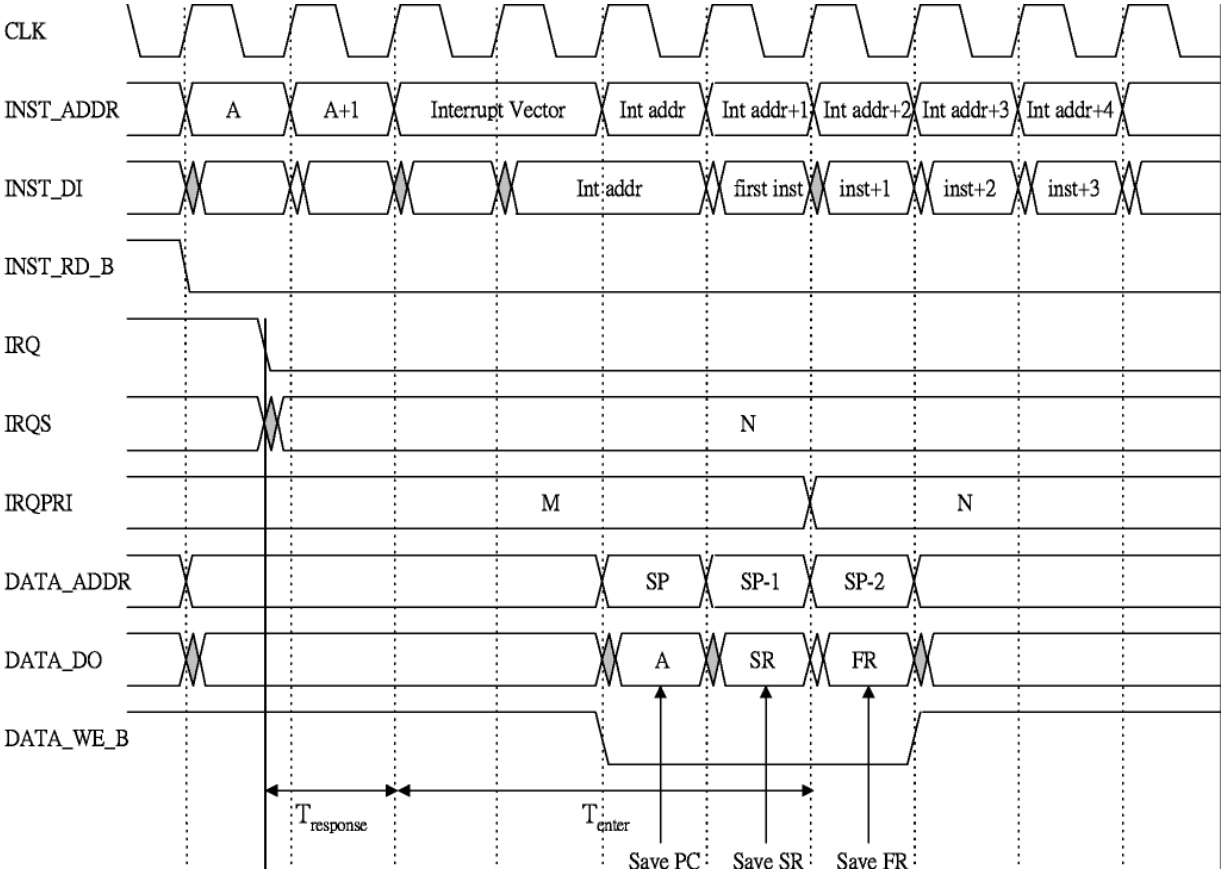


Figure 1.45

$T_{respose}$: Interrupt response time, $T_{respose}$ <= 50 clock cycles (max instruction executing cycles, MULS). T_{enter} : Interrupt entering time, 5 cycles needed from CPU accept interrupt request to fetch the first instruction.

IRQS[2:0]: External triggered IRQ number.

IRQPRI[3:0]: Internal interrupt priority register, user can change its value in FR to disable interrupts with lower interrupt priority. After entering interrupt service routine, the IRQPRI register will be changed to current IRQ number.

- **Leaving interrupt service routine**

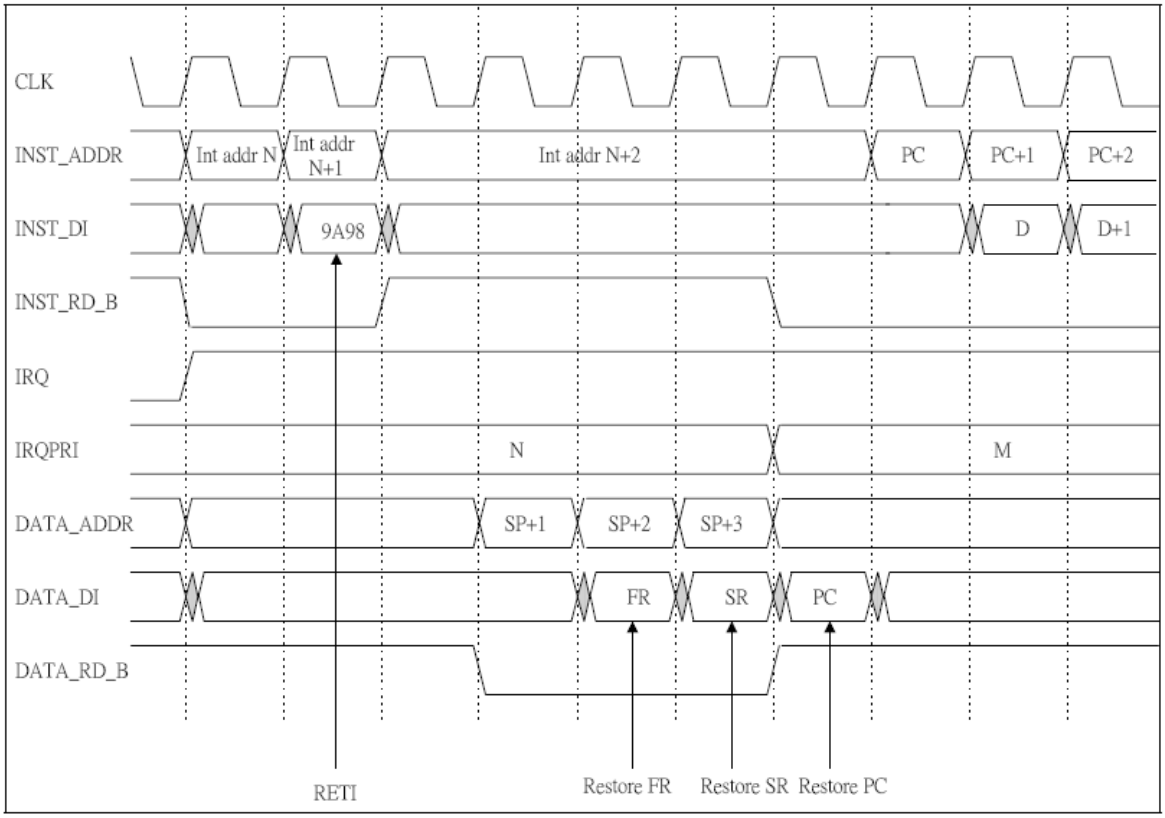


Figure 1.46

1.9 Data Types

The data structure of unSP is a 16-bit data type, called a word. A 6-bit constant data type in machine code is for quick access to the first 64 words (0x000000 ~ 0x00003F) in PAGE0. The 6-bit constant also serves as an offset to branch instructions or pointer type (via Base Pointer, BP) data access. The 22-bit constant data type in machine code is for referencing an address in memory. There is no 8-bit data type in unSP.

1.10 ALU Operation Types

The ALU operation types and its effect on the flags in SR (Status Register) are listed in Table 1.9.

Table 1.9 ALU opcode definition

Operation Type	Operation	N	Z	S	C
Add	a + b	~	~	~	~
Add with carry	a + b + C	~	~	~	~
Subtract	a + ~ b + 1	~	~	~	~
Subtract with carry	a + ~ b + C	~	~	~	~
Compare	a + ~ b + 1	~	~	~	~

Operation Type	Operation	N	Z	S	C
Negative	~ b + 1	~	~	-	-
Exclusive OR	a XOR b	~	~	-	-
OR	a OR b	~	~	-	-
AND	a AND b	~	~	-	-
Test	TEST a, b	~	~	-	-
Load from memory or register to Register	a = b	~	~	-	-
Store from register to memory	a = b	-	-	-	-

N, Z, S, C: Negative, Zero, Sign, and Carry.

The flags are defined as follows.

Flag N is 1: if the MSB (most significant bit) of result is 1.

Flag N is 0: if the MSB (most significant bit) of result is 0.

Flag Z is 1: if the result is 0.

Flag Z is 0: if the result is not 0.

Flag S is 1: if the result is negative (for two's complement).

Flag S is 0: if the result is not negative.

Flag C is 1: if carry occurs.

Flag C is 0: if no carry occurs.

For unsigned operations, the largest number for 16-bit representation is 0xFFFF (65535). If the results are greater than 0xFFFF (65535), flag C is set. For two's complement operations, the largest number is 0x7FFF (32767) and the smallest is 0x8000 (-32768). If the computation result is less than zero, flag S is set. However, the result could be larger than 0x7FFF or smaller than 0x8000. For example, 0x7FFF (32767) + 0x7FFF (32767) = 0xFFFE (65534). The result is positive (S=0) and no carry is set (C=0). In this case, the N flag is set (N=1 since the MSB of the result is 1). Overflow occurs if flag N and S are different, either S=0, N=1 or vice versa. In operation, the flags will not be changed if the destination register is PC.

1.11 Conditional Branches

Conditional branches consult flags. Four bits (Opcode, bits 15:12) in the branch type instructions are defined in Table 1.10.

Table 1.10 OP codes in conditional branch operations

Syntax	Description	Branch
JCC	Carry clear	C==0
JB	Below (unsigned)	C==0
JNAE	Not above and equal (unsigned)	C==0

Syntax	Description	Branch
JCS	Carry set	C==1
JNB	Not below (unsigned)	C==1
JAE	Above and equal (unsigned)	C==1
JSC	Sign clear	S==0
JGE	Great and equal (signed)	S==0
JNL	Not less (signed)	S==0
JSS	Sign set	S==1
JNGE	Not great than (signed)	S==1
JL	Less (signed)	S==1
JNE	Not equal	Z==0
JNZ	Not zero	Z==0
JZ	Zero	Z==1
JE	Equal	Z==1
JPL	Plus	N==0
JMI	Minus	N==1
JBE	Below and equal (unsigned)	Not (Z==0 and C==1)
JNA	Not above (unsigned)	Not (Z==0 and C==1)
JNBE	Not below and equal (unsigned)	Z==0 and C==1
JA	Above (unsigned)	Z==0 and C==1
JLE	Less and equal (signed)	Not (Z==0 and S==0)
JNG	Not great (signed)	Not (Z==0 and S==0)
JNLE	Not less and equal (signed)	Z==0 and S==0
JG	Great (signed)	Z==0 and S==0
JVC	Not overflow (signed)	N == S
JVS	Overflow (signed)	N != S
JMP	Unconditional branch	Always

2 unSP- 1.1 Instruction Set

2.1 unSP Instructions Classification

2.1.1 Notation

The following notations will be effective in the following chapters of instruction set description.

Rd	Destination register or pointer of destination memory
Rs	Source register or pointer of source memory
X, Y	Source operation units. X, Y will be shown as different object according to addressing mode.
Rx ~ Ry	User registers; x and y are the serial number
MR	A 32-bit multiplicative result register composed of R3 and R4 (R4 is high word group, R3 is low word group)
#	Sign of ALU operation
NZSC	Flags for ALU operation
+, -,	Addition, subtraction, multiplication
&, |, ^, ~	Logical AND, logical OR, logical XOR, logical NOT
	Data transfer
SFT	Shift type
Nn	The number of shift bits
IM6, IM16	6-bit immediate value, 16-bit immediate value
A6, A16	bit 0–5 of an address expression, bit 0–15 of an address expression
PC, SP, BP	Program counter register, stack pointer register, base pointer register
SR	Status register
CS, DS	Code segment and data segment in SR
Offset	Bit 0–15 offset of a 22-bit address expression
Segment	Bit 16–21 of a 22-bit address expression, which is the page number
{ }	Optional
[ ]	Sign of register indirect addressing
D	Sign of non-zero pages addressing
++, --	Sign of increasing or decreasing a word for pointer
Ss	Signed to signed number
Us	Unsigned to singed number
If cond = 1	If the result of condition for NZSC is true
Label, sub_prog	Label of program and sub-program
CPUCLK	CPU clock
N	The number of items for inner product signed by MULS

FIR Finite Impulse Response filter

// Commentary line

2.1.2 Instruction Classification

There are 41 instructions in unSP all of which can be divided into four types. See Table 2.1.

Table 2.1 unSP Instruction Classifications

Type	Instruction influx	Operations
Data Transfer	LOAD, STORE	X R d , R d X
	PUSH, POP	Rx~Ry[Rs], [Rs]Rx~Ry
ALU Operation	ADD, SUB	(XY) Rd
	ADC, SBC	(X Y C) Rd
	NEG, CMP	~X+1Rd, X–Y, NZSC will be affected
	MUL	R d R s M R
	MULS	MR + [Rd][Rs]MR
	AND, OR, XOR	X&YRd, X|YRd, X^YRd
	TEST	X&Y, only NZSC will be affected
	SFT	Rd # (Rs SFT nn) Rd
Transfer Control	BREAK	PC[SP], SR[SP+1], [0xFFF5]PC, 0 CS
	CALL label	PC[SP], SR[SP+1], $(A22)_{15\sim0}$ $^{PC.}$ $(A22)_{21\sim16}$CS
	RETF, RETI	[SP]SR, [SP–1]PC
	Jcond, JMP label	If cond=1, PC±IM6; PC±IM6
	GOTO label	*unSP*1.0: A 1 6 P C *unSP*1.1: $(A22)_{15\sim0}$ PC, $(A22)_{21\sim16}$CS
Miscellaneous	FIR_MOV ON/OFF	Enable/disable automatic data movement for FIR filter
	FIQ ON/OFF	Enable/disable FIQ
	IRQ ON/OFF	Enable/disable IRQ
	INT	Set flags to enable/disable FIQ and IRQ
	NOP	Implemented as an unconditional jump to next address

2.2 *unSP* Instruction Format

The assembly instructions of *unSP* will be translated into five types of machine codes by the assembler. Some terms should be defined before we describe these instructions. See Table 2.2.

Table 2.2 Fields in Instruction Format

Field in instruction	Area symbol	Remark
Operation type	OP	Used for appointing the function, addressing mode and operation type of instructions.
Operand	OPD	Operand can be divided into register, immediate and offset of address by different operation type or addressing mode.
Operand expansion	OPDE	Operand can be expanded into 16-bit immediate and 16-bit offset of address by different addressing modes.
Conditional code	COND	Various conditional codes in jump instructions.
Flags	FL	It is used to label the symbols of operation attribute(D, @, S, W, SFT, F, I).
Range	RG	It is used to label fields of operation range (SIZE—Serials, nn—Shift)

Five instruction formats mentioned above will be listed in Table 2.3. Each field in every instruction will show different form according to different operation and addressing mode.

Table 2.3 *unSP* Instruction Format

No.	Instruction Format		Example
	Word Group 1	Word Group 2	
1	15 0 OP 15 0 OP | FL		RETF, RETI, NOP, BREAK FIR_MOV ON/OFF FIQ ON/OFF, IRQ ON/OFF, INT
2	15 0 OP | OPD1 | OPD2 15 0 OP | OPD1 | FL | OPD2 15 0 OP | OPD1 | RG | OPD2 15 0 OP | OPD1 | FL,RG | OPD2		Rd #= IM6, Rd #= [A6] Rd #= [BP+IM6] MR = Rd * Rs {, ss} MR = Rd * Rs, us PUSH Rx, Ry to [Rs] POP Rx, Ry from [Rs] MR = [Rd] * [Rs] {,ss} {, n} MR = [Rd] * [Rs], us {, n}

3	15 0
	OP | OPD1 | OPD2
		15 0
		OPDE
			Rd = Rs # IM16,
			Rd = Rs # [A16]

4	15 0 COND | OP | OPD |	Jcond label, JMP label
5	15 0 15 0 OP | OPDE 15 0 15 0 OP | OPD | OPDE 15 0 15 0 OP | OPD | OPDE	GOTO label (μnSP - 1.0, the same as PC = IM16) Goto label (μnSP - 1.1) CALL label

Therefore, we can find that the number of operand in *unSP* instruction can be 0, 1, 2 or 3. The location of operand depends on the addressing mode. We can take 16-bit word group as a unit and arrange instruction for single word group (short instruction) and double word group (long instruction).

2.3 *unSP*-1.1 Instruction Set

Each instruction subset of instruction set will be listed one by one by the sequence of instruction type.

2.3.1 Data-Transfer Instructions

LOAD	Load Register with Memory/Immediate/Register

Syntax	Instruction Format									Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2				N	Z	S	C
Rd = IM6	xxxx	Rd	xxx	IM6				-		2	IM6	√	√	-	-
Rd = IM16	xxxx	Rd	xxxxxx			Rs		IM16		4 / 5	IM16				
Rd = [BP+IM6]	xxxx	Rd	xxx	IM6				-		6	[BP+IM6]				
Rd = [A6]	xxxx	Rd	xxx	A6				-		5 / 6	[A6]				
Rd = [A16]	xxxx	Rd	xxxxxx			Rs		A16		7 / 8	[A16]				
Rd = Rs	xxxx	Rd	xxxxxx			Rs		-		3 / 5	R				
Rd = {D:}[Rs] Rd = {D:}[++Rs] Rd = {D:}[Rs--] Rd = {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs		-		6 / 7	[R]				

Note: The x in word group denotes the data bit of "0" or "1". They can be fields listed in Table 2.3 except the operand field. The description for operand and addressing mode will be dominated but the other field ignored briefly. The same rule can be applied for the following tables.

Description: The group of instruction will be executed for reading of data transmitting, i.e. Rd=X. X shows different form according to addressing mode.

IM6, IM16: X is a 6-bit or 16-bit immediate. IM6 will be expanded to16 bits filled with zeros first, and then stored to Rd.

[BP+IM6]: X is the memory in PAGE0 addressed as (BP+IM6).

A6,A16: X is the memory in the PAGE0 addressed as (0x00~0x3F) or (0x0000~0xFFFF)

R: X may be register R1~R4, BP, SP or SR.

[R]: X is the memory pointed by offset in Rs. Rs may point data segment in PAGE0 as 'D' is ignored or in non-PAGE0 as 'D' is not ignored and its page index depends on DS in SR register. Rs can be increased or decreased in a word before or after operation. This is only a group of instruction that either addressed as PAGE0 or non-PAGE0 in *unSP* instruction set.

Note: Rd may be register R1~R4, BP, SP or SR if the addressing mode is IM6 or [BP+IM6]. Rd may also be PC besides IM6 and [BP+IM6] addressing mode. Cycles will be longer alternatively in above instruction table and all flags will be unresponsive if Rd is PC.

Examples:

```
R1 = 0x28;                  // IM6
R2 = 0x2400;                // IM16
R3 = [BP+0x08];             // [BP+IM6]
R4 = [0x30];                // A6
BP = [0x2480]; SR =         // A16
R2;                         // R
PC = D:[R1++];              // [R], Write to PC, Cycles:7
```

STORE	Store Register into Memory

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
[BP+IM6] = Rd	xxxx	Rd	xxx	IM6	-	6	[BP+IM6]	-	-	-	-
[A6] = Rd	xxxx	Rd	xxx	A6	-	5 / 6	[A6]				
[A16] = Rd	xxxx	Rd	xxxxxx	Rs	A16	7 / 8	[A16]				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
{D:}[Rs] = Rd {D:}[++Rs] = Rd {D:}[Rs--] = Rd {D:}[Rs++] = Rd	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for writing of data transmitting, i.e. X=Rd. X shows different form according to addressing mode.

[BP+IM6]: X is the memory in PAGE0 addressed as (BP+IM6).

A6,A16: X is the memory in PAGE0 addressed as (0x00-0x3F) or (0x0000-0xFFFF)

[R]: X is the memory pointed by offset in Rs. Rs may point data segment in PAGE0 as 'D' is ignored or in non-PAGE0 as 'D' is not ignored and its page index depends on DS in SR register. Rs can be increased or decreased in a word before or after operation. This is only a group of instruction that either addressed as PAGE0 or non-PAGE0 in *unSP* instruction set.

Note: Rd may be register R1~R4, BP, SP or SR if the addressing mode is IM6 or [BP+IM6]. Rd may also be PC besides IM6 and [BP+IM6] addressing mode. Cycles will be longer alternatively in above instruction table if Rd is PC.

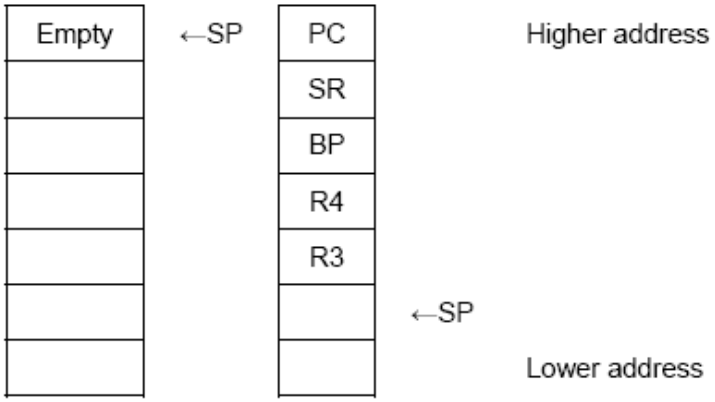
Example:
```
[BP+0x08] = R3;        // Write to [BP+IM6]
[0x30] = R4;           // Write to [A6]
[0x2480] = BP;         // Write to [A16]
D:[R4++] = PC;         // Read from PC, Cycles 7
```

PUSH — Push Registers onto Stack

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
PUSH Rx, Ry to [Rs] Or PUSH Rx to [Rs]	xxxx	Rd	xxx	SIZE	Rs	-	2n+4	[R]	-	-	-	-

Description: Push a number (number n=1~7, SIZE) of registers Rx-Ry (Rx~RySP) to memory pointed by Rs decreasingly.

Example: PUSH R3, PC to [SP]; // Push R3 through PC (R7) to SP



Note: PUSH R3, PC to [SP] is equivalent to PUSH PC, R3 to [SP]

POP	Pop Registers from Stack

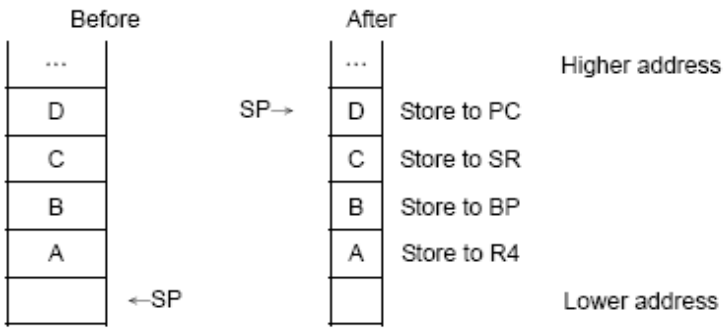
Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
POP Rx, Ry from [Rs] Or POP Rx from [Rs]	xxxx	Rd	xxx	SIZE	Rs	-	2n+4	[R]	*	*	*	*

Description: Copy a set of memory pointed by Rs consecutively to a set of register Rx-Ry (Rx~Ry SP) where n=1~7. It is also equivalent to RETF/RETI when Rx~Ry is SR~PC.

Note:

1. When SR is not in the set of Rx ~ Ry, only N and Z flags will be determined by Ry.
2. When SR is in the set of register Rx~Ry, NZSC flags will be changed. However, N and Z will be eventually determined by Ry.

Example: POP R4, PC from [SP]; // Pop R4 through PC from SP



POP SR, PC from [SP]; // It equals to RETF

Note: POP R4, PC from [SP] is equivalent to POP PC, R4 from [SP]

2.3.2 Arithmetic/Logical-Operation Instructions

This is Arithmetic/Logical-Operation Instructions that carry out the operation as RD = X # Y. X and Y will show different meanings according to the addressing mode. Because the same explanation for X, Y and the description for Rs, Rd will be involved in instruction they will be listed in Table 2.4.

Table 2.4 The meanings for X, Y in operation as Rd = X # Y

Addressing Mode	X, Y
IM6	X is Rd, Y is IM6. IM6 will be expanded to 16-bit filled with zeros first, and then be operated with X.
IM16	X is Rs, Y is IM16
[BP+IM6]	X is Rd, Y is the memory in PAGE0 addressed as (BP+IM6)
[A6]	X is Rd, Y is the memory in PAGE0 addressed as (0x00~0x3F)
[A16]	X is Rs. Y is the memory in PAGE0 addressed as (0x0000~0xFFFF)
R	X is Rd, Y is Rs.
[R]	X is Rd, Y is the memory address pointed by the offset in Rs. Rs may point data segment in PAGE0 as 'D' is ignored or in non-PAGE0 as 'D' is not ignored and its page index depends on DS in SR register. Rs can be increased or decreased in a word before or after operation. This is only a group of instruction that either addressed as PAGE0 or non-PAGE0 in *unSP* instruction set.

Note: Rs may be R1~R4, BP, SP, SR and PC. Rd may be R1~R4, BP, SP and SR if the addressing mode is [BP+IM6]. Rd may also be PC besides [BP+IM6] addressing mode. Cycles will be longer alternatively in above instruction table and all flags will be unresponsive if Rd is PC.

For shift operation instructions:

- FIQ, IRQ and user routine has their own shift buffers. User does not need to save shift buffer for interrupt routines.
- Shift buffer values are unknown after multiplication or filter operations. User should make no assumptions to its value after the operations.
- Carry flag only couple with ALU operation, not shift operation.

ADD — **ADD without Carry**

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
Rd += IM6 Rd = Rd + IM6	xxxx	Rd	xxx	IM6	-	2	IM6	√	√	√	√

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
Rd = Rs + IM16	xxxx	Rd	xxxxxx				Rs	IM16	4 / 5	IM16				
Rd += [BP+IM6] Rd = Rd + [BP+IM6]	xxxx	Rd	xxx	IM6					6	[BP+IM6]				
Rd += [A6] Rd = Rd + [A6]	xxxx	Rd	xxx	A6				-	5 / 6	[A6]				
Rd = Rs + [A16]	xxxx	Rd	xxxxxx				Rs	A16	7 / 8	[A16]				
Rd += Rs	xxxx	Rd	xxxxxx				Rs	-	3 / 5	R				
Rd += {D:}[Rs] Rd += {D:}[++Rs] Rd += {D:}[Rs--] Rd += {D:}[Rs++]	xxxx	Rd	xxx	D	@		Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for addition operation without carry, i.e. Rd = X+Y. X, Y will have different meanings according to the addressing mode. See Table 2.4.

Example:

```
R1 += 0x28;              // IM6
R2 = R1 + 0x2400;        // IM16
R3 += [BP+0x08];         // [BP+IM6]
R4 += [0x30];            // [A6]
BP = R4 + [0x2480];      // [A16]
SR += R2;                // R
PC += D:[BP++];          // Write to PC, Cycles: 7
```

ADC — ADD with Carry

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
Rd += IM6, Carry Rd = Rd + IM6, Carry	xxxx	Rd	xxx	IM6				-	2	IM6	√	√	√	√
Rd = Rs + IM16, Carry	xxxx	Rd	xxxxxx				Rs	IM16	4 / 5	IM16				
Rd += [BP+IM6], Carry Rd = Rd + [BP+IM6], Carry	xxxx	Rd	xxx	IM6				-	6	[BP+IM6]				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += [A6], Carry Rd = Rd + [A6], Carry	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs + [A16], Carry	xxxx	Rd	xxxxxx			Rs	A16	7 / 8	[A16]				
Rd += Rs, Carry	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				
Rd += {D:}[Rs], Carry Rd += {D:}[++Rs], Carry Rd += {D:}[Rs--], Carry Rd += {D:}[Rs++], Carry	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for addition with carry in arithmetical operation, i.e. Rd = X+Y+C. X, Y will have different meanings according to the addressing mode. See Table 2.4.

Example:

```
R1 = 0x28, Carry;                          // R1 = R1 + IM6 + C
R2 = R1 + 0x2400, Carry;                   // R2 = R1 + IM16 + C
R3 += [BP+0x08], Carry;                    // R3 = R3 + [BP+IM6] + C // R4 = R4 + [A6] + C
R4 += [0x30];                              // BP = R4 + [A16] + C // SR = SR + R2 + C
BP = R4 + [0x2480], Carry; SR +=           // Write to PC, Cycles: 7
R2, Carry;
PC += D:[BP++], Carry;
```

SUB — Subtract without Carry

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= IM6 Rd = Rd – IM6	Xxxx	Rd	xxx	IM6			-	2	IM6	√	√	√	√
Rd = Rs – IM16	Xxxx	Rd	xxxxxx			Rs	IM16	4 / 5	IM16				
Rd -= [BP+IM6] Rd = Rd - [BP+IM6]	Xxxx	Rd	xxx	IM6			-	6	[BP+IM6]				
Rd -= [A6] Rd = Rd - [A6]	Xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs - [A16]	Xxxx	Rd	xxxxxx			Rs	A16	7 / 8	[A16]				
Rd -= Rs	Xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= {D:}[Rs] Rd -= {D:}[++Rs] Rd -= {D:}[Rs--] Rd -= {D:}[Rs++]	Xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for subtraction without carry in arithmetical operation, i.e. Rd = X - Y. X, Y will have different meanings according to the addressing mode. See Table 2.4.

Example:

```
R1 -= 0x28;              // R1 = R1 – IM6
R2 = R1 - 0x2400;        // R2 = R1 – IM16
R3 -= [BP+0x08];         // R3 = R3 – [BP+IM6]
R4 -= [0x30];            // R4 = R4 – [A6]
BP = R4 - [0x2480];      // BP = R4 – [A16]
SR -= R2;                // SR = SR – R2
PC -= D:[BP++];          // Write to PC, cycles: 7
```

SBC	Subtract with Carry

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= IM6, Carry Rd = Rd - IM6, Carry	xxxx	Rd	xxx	IM6				2	IM6	√	√	√	√
Rd = Rs - IM16, Carry	xxxx	Rd	xxxxxx			Rs	IM16	4 / 5	IM16				
Rd -= [BP+IM6], Carry Rd = Rd - [BP+IM6], Carry	xxxx	Rd	xxx	IM6			-	6	[BP+IM6]				
Rd -= [A6], Carry Rd = Rd - [A6], Carry	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs - [A16], Carry	xxxx	Rd	xxxxxx			Rs	A16	7 / 8	[A16]				
Rd -= Rs, Carry	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= {D:}[Rs], Carry Rd -= {D:}[++Rs], Carry Rd -= {D:}[Rs--], Carry Rd -= {D:}[Rs++], Carry	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for subtraction with carry in arithmetical operation, i.e. Rd = X – Y – C = X + (~Y) + C. X, Y will have different meanings according to the addressing mode. See Table 2.4.

Example:

```
R1 -= 0x20, Carry;                  // R1 = R1 – IM6 - C
R2 = R1 - 0x2400, Carry;            // R2 = R1 – IM16 - C
R3 -= [BP+0x08], Carry;             // R3 = R3 – [BP+IM6] - C
R4 -= [0x30], Carry;                // R4 = R4 – [A6] - C
BP = R4 - [0x2480], Carry;          // BP = R4 – [A16] - C
SR -= R2, Carry;                    // SR = SR – R2 - C
PC -= D:[BP++], Carry;              // Write to PC, cycles: 7
```

NEG — Negative

Syntax	Instruction Format							Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1										N	Z	S	C
Rd = -IM6	xxxx	Rd	xxx	IM6				-	2	IM6	√	√	-	-
Rd = -IM16	xxxx	Rd	xxxxxx			Rs		IM16	4 / 5	IM16				
Rd = -[BP+IM6]	xxxx	Rd	xxx	IM6				-	6	[BP+IM6]				
Rd = -[A6]	xxxx	Rd	xxx	A6				-	5 / 6	[A6]				
Rd = -[A16]	xxxx	Rd	xxxxxx			Rs		A16	7 / 8	[A16]				
Rd = -Rs	xxxx	Rd	xxxxxx			Rs		-	3 / 5	R				
Rd = -{D:}[Rs] Rd = -{D:}[++Rs] Rd = -{D:}[Rs--] Rd = -{D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs		-	6 / 7	[R]				

Description: The group of instruction will be executed for negation in arithmetical operation, i.e. Rd = -X = ~X+1. The meaning of X will be described as follow according to the different addressing modes.

IM6, IM16: X is IM6 or IM16. IM6 will be expanded to 16 bit filled with zeros first, and then carry out negation.

[BP+IM6]: X is the memory in PAGE0 addressed as (BP+IM6).

[A6], **[A16]:** X is the memory in PAGE0 addressed as (0x00~0x3F) or (0x0000~0xFFFF)

R: X may be R1~BP(R5), SP, SR.

[R]: X is the memory pointed by offset in Rs. Rs may points to data segment in PAGE0 as 'D' is ignored or in non-PAGE0 as 'D' is not ignored and its page index depends on DS in SR register. Rs can be increased or decreased in a word before or after operation. This is only a group of instruction that either addressed as PAGE0 or non-PAGE0 in *unSP* instruction set.

Note: Rd may be register R1~R4, BP, SP or SR if the addressing mode is IM6 or [BP+IM6]. Rd may also be PC besides IM6 and [BP+IM6] addressing mode. Cycles will be longer alternatively in above instruction table and all flags will be unresponsive if Rd is PC.

```
Example:  R1 = -0x27;                 // R1 = - IM6
          R3 = -[BP+0x08];            // R3 = - [BP+IM6]
          R4 = -[0x30];               // R4 = - [A6]
          BP = -[0x2480];             // BP = - [A16]
          SR = -R2;                   // SR = - R2
          PC = -D:[BP++];             // Write to PC, cycles: 7
```

CMP	Compare

Syntax	Instruction Format										Cycles	Addressing Mode	Flags			
	Word Group 1									Word Group 2			N	Z	S	C
CMP Rd, IM6	xxxx	Rd	xxx	IM6						-	2	IM6	√	√	√	√
CMP Rs, IM16	xxxx	Rd	xxxxxx				Rs			IM16	4 / 5	IM16				
CMP Rd, [BP+IM6]	xxxx	Rd	xxx	IM6						-	6	[BP+IM6]				
CMP Rd, [A6]	xxxx	Rd	xxx	A6						-	5 / 6	[A6]				
CMP Rs, [A16]	xxxx	Rd	xxxxxx				Rs			A16	7 / 8	[A16]				
CMP Rd, Rs	xxxx	Rd	xxxxxx				Rs			-	3 / 5	R				
CMP Rd, {D:}[Rs] CMP Rd, {D:}[++Rs] CMP Rd, {D:}[Rs--] CMP Rd, {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs				-	6 / 7	[R]				

Description: The group of instruction will be executed for comparison in arithmetical operation, i.e. X - Y. But its result will not be stored and only affect NZSC flags. X, Y will have different meanings according to the addressing mode. See Table 2.4.

Example:

```
CMP R1, 0x27;          // Compare R1, IM6
CMP R3, [BP+0x08];     // Compare R3, [BP+IM6]
CMP R4, [0x30];        // Compare R4, [A6]
CMP BP, [0x2480];      // Compare BP, [A16]
CMP SR, R2;            // Compare SR, R2
CMP PC, D:[BP++];      // Compare with PC, cycles: 7
```

MUL — **Register Multiplication**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
MR = Rd * Rs {, ss} MR = Rd * Rs, us	xxxx	Rd	S	xxxxx	Rs	-	12	R	-	-	-	-

Description: The group of instruction will be executed for multiplication in arithmetical operation, i.e. MR=Rd*Rs. And "ss" will indicate that two of word data in Rd and Rs are all signed, "us" mean that the word data Rd is unsigned and that in Rs is signed. Rd, Rs may be register R1~R4, BP. The result is put into MR, which is a virtual 32-bit register combined from R4 and R3. R4 contains the higher 16 bits of MR. R3 contains the lower 16 bits of MR.

Example:

```
MR = R2 * R1;          // Two signed values
MR = R1 * R2, us;      // R1 is unsigned and R2 is signed
MR = R3 * R4, ss;      // Two signed values
```

MULS — **Sum of Register Multiplication**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
MR = [Rd] * [Rs] {,ss} {,n} MR = [Rd] * [Rs], us {,n}	xxxx	Rd	S	x	SIZE	Rs	-	10n+6	[R]	-	-	-	-

Description: The group of instruction will be executed in sum of register multiplication. Its result will be stored into MR register. And "ss" will indicate that two word data pointed by the content in Rd and Rs are all signed, "us" means that the word data pointed by Rd is unsigned and that pointed by Rs is signed. The items of operation will be shown by "n" which can be 1~16. 1 is default. See the following chart. Among which Rd and Rs may be R1, R2, and BP. (Note: To avoid misusing, Rd and Rs cannot be SP, SR, PC, R3 and R4; moreover, Rd and Rs cannot be set as the same register).

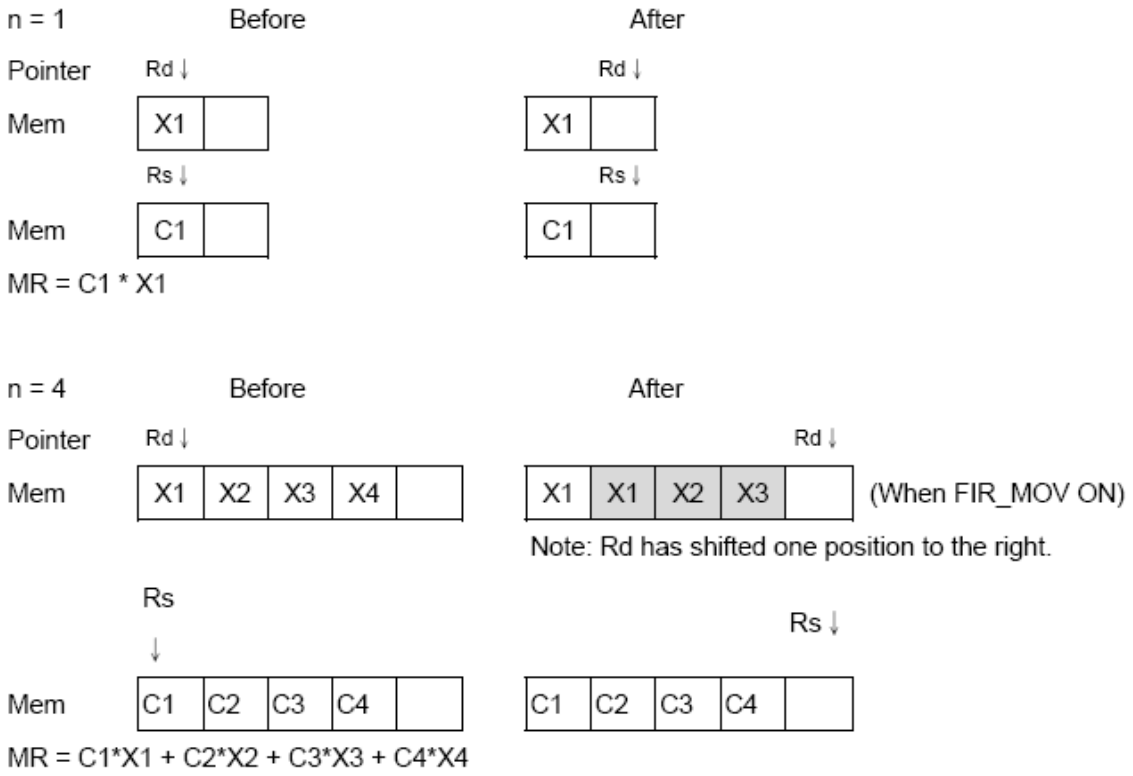


Figure 2.1 Inner Multiplication Operation chart

Example: MR = [R2] * [R1], 8; // The inner multiplication of two signed
MR = [R1] * [R2], us, 2; // R1 is unsigned, R2 is signed. MR = [R2] * [BP], ss, 4; // Two signed value.

AND **Logical AND**

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
Rd &= IM6 Rd = Rd & IM6	xxxx	Rd	xxx	IM6	-	2	IM6	√	√	-	-
Rd = Rs & IM16	xxxx	Rd	xxxxxx	Rs	IM16	4 / 5	IM16				

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd &= [BP+IM6] Rd = Rd & [BP+IM6]	xxxx	Rd	xxx	IM6		-	6	[BP+IM6]				
Rd &= [A6] Rd = Rd & [A6]	xxxx	Rd	xxx	A6		-	5 / 6	[A6]				
Rd = Rs & [A16]	xxxx	Rd	xxxxxx		Rs	A16	7 / 8	[A16]				
Rd &= Rs	xxxx	Rd	xxxxxx		Rs	-	3 / 5	R				
Rd &= {D:}[Rs] Rd &= {D:}[++Rs] Rd &= {D:}[Rs--] Rd &= {D:}[Rs++]	xxxx	Rd	xxx	D	@ Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed in logical AND operation, i.e. Rd = X & Y. The X and Y will have different meanings according to the addressing mode. See Table 2.4.

Example:

```
R1 &= 0x2F;                 // R1 = R1 & IM6
R3 &= [BP+0x08];            // R3 = R3 & [BP+IM6]
R4 &= [0x30];               // R4 = R4 & [A6]
BP = R2 & [0x2480];         // BP = R2 & [A16]
SR &= R2;                   // SR = SR & R2
PC &= D:[BP++];             // Write to PC, cycles: 7
```

OR	Logical Inclusive OR

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd |= IM6 Rd = Rd | IM6	xxxx	Rd	xxx	IM6		-	2	IM6	√	√	-	-
Rd = Rs | IM16	xxxx	Rd	xxxxxx		Rs	IM16	4 / 5	IM16				
Rd |= [BP+IM6] Rd = Rd | [BP+IM6]	xxxx	Rd	xxx	IM6		-	6	[BP+IM6]				
Rd |= [A6] Rd = Rd | [A6]	xxxx	Rd	xxx	A6		-	5 / 6	[A6]				
Rd = Rs | [A16]	xxxx	Rd	xxxxxx		Rs	A16	7 / 8	[A16]				
Rd |= Rs	xxxx	Rd	xxxxxx		Rs	-	3 / 5	R				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd |= {D:}[Rs] Rd |= {D:}[++Rs] Rd |= {D:}[Rs--] Rd |= {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed in logical OR operation, i.e. Rd = X | Y. The X and Y will have different meanings according to the addressing mode. See Table 2.4.

Example:

```
R1 |= 0x2F;                 // R1 = R1 | IM6
R3 |= [BP+0x08];            // R3 = R3 | [BP+IM6]
R4 |= [0x30];               // R4 = R4 | [A6]
BP = R2 | [0x2480];         // BP = R2 | [A16]
SR |= R2;                   // SR = SR | R2
PC |= D:[BP++];             // Write to PC, cycles: 7
```

XOR	Logical Exclusive OR

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd ^= IM6 Rd = Rd ^ IM6	xxxx	Rd	xxx	IM6			-	2	IM6	√	√	-	-
Rd = Rs ^ IM16	xxxx	Rd	xxxxxx			Rs	IM16	4 / 5	IM16				
Rd ^= [BP+IM6] Rd = Rd ^ [BP+IM6]	xxxx	Rd	xxx	IM6			-	6	[BP+IM6]				
Rd ^= [A6] Rd = Rd ^ [A6]	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs ^ [A16]	xxxx	Rd	xxxxxx			Rs	A16	7 / 8	[A16]				
Rd ^= Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				
Rd ^= {D:}[Rs] Rd ^= {D:}[++Rs] Rd ^= {D:}[Rs--] Rd ^= {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed in logical exclusive OR operation, i.e. Rd = X ^ Y. The X, Y will have different meanings according to the addressing mode. See Table 2.4.

Example:
```
R1 ^= 0x2F;                 // R1 = R1 ^ IM6
R3 ^= [BP+0x08];            // R3 = R3 ^ [BP+IM6]
R4 ^= [0x30];               // R4 = R4 ^ [A6]
BP = R2 ^ [0x2480];         // BP = R2 ^ [A16]
SR ^= R2;                   // SR = SR ^ R2
PC ^= D:[BP++];             // Write to PC, cycles: 7
```

TEST — Logical Test

Syntax	Instruction Format: Word Group 1						Instruction Format: Word Group 2	Cycles	Addressing Mode	Flags: N	Z	S	C
TEST Rd, IM6	xxxx	Rd	xxx	IM6			-	2	IM6	√	√	-	-
TEST Rs, IM16	xxxx	Rd	xxxxxx		Rs		IM16	4 / 5	IM16				
TEST Rd, [BP+IM6]	xxxx	Rd	xxx	IM6			-	6	[BP+IM6]				
TEST Rd, [A6]	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
TEST Rs, [A16]	xxxx	Rd	xxxxxx		Rs		A16	7 / 8	[A16]				
TEST Rd, Rs	xxxx	Rd	xxxxxx		Rs		-	3 / 5	R				
TEST Rd, {D:}[Rs] TEST Rd, {D:}[++Rs] TEST Rd, {D:}[Rs--] TEST Rd, {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for logical AND operation, i.e. X&Y. However, its result will not be stored and it only affects NZ flags. The X and Y will have different meanings according to the addressing mode. See Table 2.4.

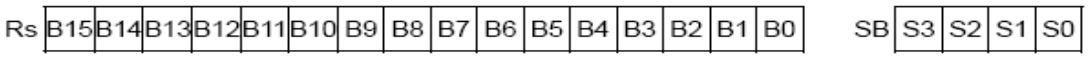
Example:
```
TEST R1, 0x27;              // TEST R1 and IM6
TEST R3, [BP+0x08];         // TEST R3 and [BP+IM6]
TEST R4, [0x30];            // TEST R4 and [A6]
TEST BP, [0x2480];          // TEST BP and [A16]
TEST SR, R2;                // TEST SR and R2
TEST PC, D:[BP++];          // TEST PC and D:[BP++], cycles: 7
```

ASR-ALU — Register Arithmetic-Shift-Right and Arithmetic/Logical Operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += Rs ASR nn {,Carry} Rd -= Rs ASR nn {,Carry} CMP Rd, Rs ASR nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs ASR nn Rd &= Rs ASR nn Rd |= Rs ASR nn Rd ^= Rs ASR nn TEST Rd, Rs ASR nn Rd = Rs ASR nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	-	-

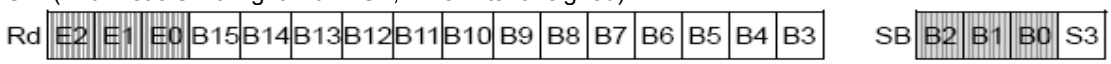
Description: These group of instruction will be executed in arithmetic operation with logical shift right where nn is number of shifting bits and ranged in [1~4]. Or Rs carries out arithmetic and logical operations with Rd (Rd, Rs ~ SP, PC; Rs ~ SR) and the result is stored to Rd.

Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of

ASR: (Arithmetic Shift Right with MSB, which fits for signed)



Note: Carry flag only couples with ALU operation, not shift operation.

Example:

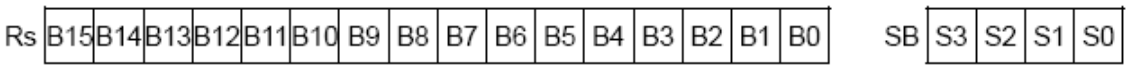
```
SR |= R2 ASR 2;          // SR = SR | (R2 / 2^2)
SP += R1 ASR 4, Carry;   // SP = SP + (R1 / 2^4) + C
R2 = R1 ASR 2;           // R2 = R1 / 2^2
```

LSL-ALU	Register Logical –Shift-Left and Arithmetic/Logical Operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd+=Rs LSL nn {,Carry} Rd-=Rs LSL nn {,Carry} CMP Rd, Rs LSL nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs LSL nn Rd &= Rs LSL nn Rd |= Rs LSL nn Rd ^= Rs LSL nn TEST Rd, Rs LSL nn Rd = Rs LSL nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	-	-

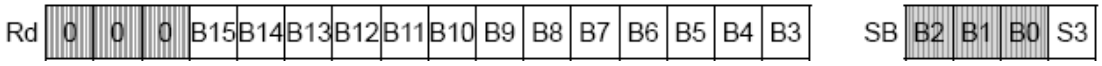
Description: The group of instruction will be executed in arithmetic and logical operations with logical shift left where nn is number of shifting bits and ranged in [1~4]. Or Rs carries out arithmetic and logical operation with Rd (Rd, Rs ~ SP, PC; Rs ~ SR), and then the result is stored to Rd. See the following chart.

Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of

LSL: (Logic Shift Left)



Note: Carry flag only couples with ALU operation, not shift operation.

Example:
```
SR |= R2 LSL 2;          // SR = SR | (R2 << 2)
SP += R1 LSL 4, Carry;   // SP = SP + (R1 << 4) + C
R2 = R1 LSL 2;           // R2 = R1 << 2
```

LSR-ALU	Register Logical-Shift-Right and Arithmetic/Logical Operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd+=Rs LSR nn {,Carry} Rd-=Rs LSR nn {,Carry} CMP Rd, Rs LSR nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs LSR nn Rd &= Rs LSR nn Rd |= Rs LSR nn Rd ^= Rs LSR nn TEST Rd, Rs LSR nn Rd = Rs LSR nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	-	-

Description: The group of instruction will be executed with logical shift right where nn is number of shifting bits and ranged in [1~4]. Or Rs carries out arithmetic and logical operations with Rd (Rd, Rs ~ SP, PC; Rs ~ SR). Then, the result is stored to Rd. See the following chart.

Before shifting op

Rs	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

SB	S3	S2	S1	S0

SB is the shift buffer. Suppose nn=3, then after shift op of

LSR: (Logic Shift Right)

Rd	0	0	0	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3

SB	B2	B1	B0	S3

Note: Carry flag only couples with ALU operation, not shift operation.

Example:
```
SR |= R2 LSR 2;              // SR = SR | (R2 >> 2)
SP += R1 LSR 4, Carry;      // SP = SP + (R1 >> 4) + C
R2 = R1 LSR 2;              // R2 = R1 >> 2
```

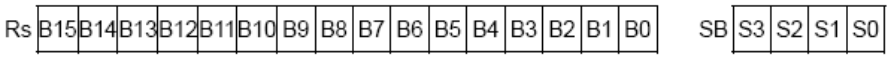
ROL-ALU	Register Rotate-Left and Arithmetic/Logical operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd+=Rs ROL nn {,Carry} Rd-=Rs ROL nn {,Carry} CMP Rd, Rs ROL nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	√	√

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd = - Rs ROL nn Rd &= Rs ROL nn Rd |= Rs ROL nn Rd ^= Rs ROL nn TEST Rd, Rs ROL nn Rd = Rs ROL nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	-	-

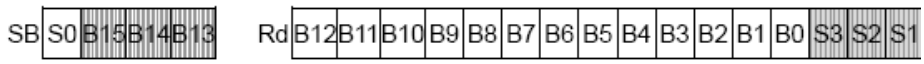
Description: The group of instruction will be executed in arithmetic and logical operations with rotate shift left where nn is number of position shift and ranged in [1~4]. Or Rs carries out arithmetic and logical operations with Rd (Rd, Rs ~ SP, PC; Rs ~ SR) then the result is stored to Rd. See the following chart.

Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of:

ROL: (Rotate Left with SB)



Note: Carry flag only couples with ALU operation, not shift operation.

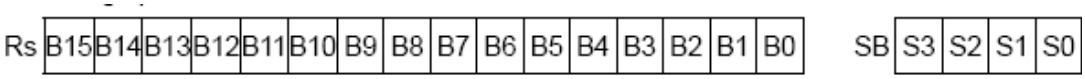
Example:
```
SR |= R2 ROL 2;              // SR = SR | (R2 ROL 2)
SP += R1 ROL 4, Carry;      // SP = SP + (R1 ROL 4) + C
R2 = R1 ROL 2;              // R2 = R1 ROL 2
```

ROR-ALU	Register Rotate-Left and Arithmetic/Logical Operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd+=Rs ROR nn {,Carry} Rd-=Rs ROR nn {,Carry} CMP Rd, Rs ROR nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs ROR nn Rd &= Rs ROR nn Rd |= Rs ROR nn Rd ^= Rs ROR nn TEST Rd, Rs ROR nn Rd = Rs ROR nn	xxxx	Rd	xx	nn	Rs	-	3 / 5	R	√	√	-	-

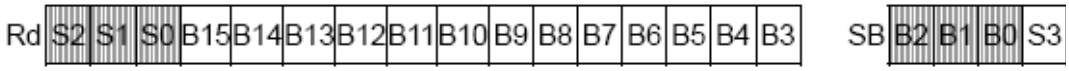
Description: The group of instruction will be executed in arithmetic and logical operations with rotate shift right where nn is number of shifting bits and ranged in [1~4]. Or Rs carries out arithmetic and logical operations with Rd (Rd, Rs ~ SP, PC; Rs ~ SR). After that, the result is stored to Rd. As following chart shows.

Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of

ROR: (Rotate Right with SB)



Note: Carry flag only couples with ALU operation, not shift operation.

Example: SR |= R2 ROR 2; // SR = SR | (R2 ROR 2)

SP += R1 ROR 4, Carry; // SP = SP + (R1 ROR 4) + C

R2 = R1 ROR 2; // R2 = R1 ROR 2

2.3.3 Transfer-Control Instructions

BREAK — **Software Interrupt**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
BREAK	xxxxxxxxxxxxxxxx	-	10	[A16]	-	-	-	-

Description: Generate a software interrupt. CPU will jump to interrupt vector [0x00FFF5] to execute interrupt service routine.

Example: BREAK; // Generate a software interrupt

CALL — **Segmented Far Call**

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
CALL Label	xxxxxxxxxx	CS6	A16	9	[A22]	-	-	-	-

Description: Call a sub-program. Label can be anywhere in the memory space. Both PC and SR are pushed to stack automatically before calling the sub-program. CPU then load CS of SR with CS6 and PC with A16 to jump to Label.

Example:

```
        CALL sub1;          // CALL sub1
        [result] = R1;      // Store the return value of  sub1
        …
Sub1: .PROC
        PUSH BP to [SP];
        BP = SP + 1;
        R2 = [BP+3];        // Parameter 1
        R3 = [BP+4];        // Parameter 2
         …..
        R1 = 0;             // Return
        value RETF;
        .ENDP
```

JUMP	Conditional/Unconditional Jump

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
Jcond label JMP label	COND	xxxxxx	S	IM6	-	2 (not-taken) / 4 (taken)	PC±IM6	-	-	-	-

Description: A group of conditional and unconditional ***short*** jump instruction to local label. Each flag in SR will be checked as routine jump condition. If condition is met, PC will jump to related addresses within63 words. If the condition is not true, PC will go to the position of next instruction. See Conditional Branch Table for details.

User can use "S"+Jcond (ex. SJG Label) format. Such command will become a smart branch that assembler will pick up least code size to encode this branch for backward jump (depending on the distance of this instruction's address with PC and encode it with short or long jump).

Table 2.5 Conditional branch operations

Cond OP Codes	Symbol Operand Type	Description	Flags

0000	JCC	—	Carry clear	C=0
0000	JB	Unsigned	Below	C=0
0000	JNAE	Unsigned	Not above and equal	C=0
0001	JCS	—	Carry Set	C=1
0001	JNB	Unsigned	Not below	C=1
0001	JAE	Unsigned	Above and equal	C=1
0010	JSC	—	Sign clear	S=0
0010	JGE	Signed	Great and equal	S=0
0010	JNL	Signed	Not less	S=0
0011	JSS	—	Sign set	S=1
0011	JNGE	Signed	Not great than	S=1
0011	JL	Signed	Less	S=1
0100	JNE	—	Not equal	Z=0
0100	JNZ	—	Not zero	Z=0
0101	JZ	—	Zero	Z=1
0101	JE	—	Equal	Z=1
0110	JPL	—	Plus	N=0
0111	JMI	—	Minus	N=1
1000	JBE	Unsigned	Below and equal	Not (Z=0 and C=1)
1000	JNA	Unsigned	Not above	Not (Z=0 and C=1)
1001	JNBE	Unsigned	Not below and equal	Z=0 and C=1
1001	JA	Unsigned	Above	Z=0 and C=1
1010	JLE	Signed	Less and equal	Not (Z=0 and S=0)
1010	JNG	Signed	Not great	Not (Z=0 and S=0)
1011	JNLE	Signed	Not less and equal	Z=0 and S=0
1011	JG	Signed	Great	Z=0 and S=0
1100	JVC	Signed	Not overflow	N=S
1101	JVS	Signed	Overflow	N ! =S
1110	JMP		Unconditional jump	

Example:
```
CMP R1, R2;
JNE label1;     // Jump to label1 when not equal
JMP labe2;      // Unconditional jump to label2
```

RETF	Return from Subroutine

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
RETF	xxxxxxxxxxxxxxxx	-	8	[A22]	√	√	√	√

Description: RETF will pop SR and PC from stack and return from subroutine. Note that the SR and PC are popped back after RETF. Therefore, they are the same with those before calling sub-programs.

Example:
```
sub1: .PROC
      … …
      RETF;          // Return from sub1
      ENDP
```

RETI	Return from Interrupt Service Routine

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
RETI	xxxxxxxxxxxxxxxx	-	8	[A22]	√	√	√	√

Description: RETI will pop SR and PC from stack and then return from interrupt service routine. Note that the SR and PC are popped back after RETI. Therefore, they are the same as those of before interrupt responses.

Example:
```
      .TEXT
      .PUBLIC _IRQ1
_IRQ1:
      … …
      RETI;          // Return from IRQ1
```

GOTO	Unconditional Far Jump

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
GOTO label	xxxxxxxxxx	CS6	A16	5	[A22]	-	-	-	-

Description: Going to user's specified address unconditionally. In *unSP* 1.0, Target address is limited to the 64K word of current page. In *unSP*1.1, the whole 4M word addressing space is allowable.

Example: GOTO loop; // Jump to loop unconditionally

2.3.4 Miscellaneous Instructions

FIR_MOV ON	Enable Automatic Data Movement for FIR Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
FIR_MOV ON	xxxxxxxxxxxxxxxx	-	2	-	-	-	-	-

Description: Enable automatic data movement for FIR operations. It affects the behavior of FIR, which is global. Hence, use it in interrupt with care.

Example:

```
_IRQ1:
PUSH R1, R4 to [SP];
CALL F_IRQ1_Service_10kHz; // Sample, FIR, output
POP R1, R4 from [SP];
RETI;

F_IRQ1_Service_10kHz:
 … … ;
R1 = Data_Entry;          // R1 points to sample vector
R2 = Conf_ Entry;         // R2 points coefficient vector
FIR_MOV ON;               // Enable automatic data movement for FIR operations
MR = [R1] * [R2], N;      // Rank n FIR calculation
FIR_MOV OFF;
R3 = R4 LSR 4;            // MR / 2^15 obtains 16-bit output
R3 = R3 LSR 4;
R3 = R3 LSR 4;
R3 = R3 LSR
3;
[P_DAC1] = R3;            // Output to DAC1
RETF;
```

FIR_MOV OFF	Register Rotate-Left and Arithmetic/Logical operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
FIR_MOV OFF	xxxxxxxxxxxxxxxx	-	2	-	-	-	-	-

Description: Disable automatic data movement for FIR operations. It affects the behavior of FIR, which is global. Hence, use it in interrupt with care.

FIQ ON	Enable FIQ

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
FIQ ON	xxxxxxxxxxxxxx	F	I	-	2	-	-	-	-	-

Description: Enable FIQ
Example: FIQ ON; // Enable IRQ

FIQ OFF	Disable FIQ

Syntax	Instruction				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
FIQ OFF	xxxxxxxxxxxxxx	F	I	-	2	-	-	-	-	-

Description: Disable FIQ

Example: FIQ OFF // Disable FIQ

IRQ ON	Enable IRQ

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
IRQ ON	xxxxxxxxxxxxxx	F	I	-	2	-	-	-	-	-

Description: Enable IRQ

Example: IRQ ON; // Enable IRQ

IRQ OFF — Disable IRQ

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
IRQ OFF	xxxxxxxxxxxxxx	F	I	-	2	-	-	-	-	-

Description: Disable IRQ

Example: IRQ OFF // Disable IRQ

INT — Interrupt Set

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
INT FIQ INT IRQ INT FIQ, IRQ INT OFF	xxxxxxxxxxxxxx	F	I	-	2	-	-	-	-	-

Description: Set FIQ/IRQ flags.

Example: INT FIQ; // Enable FIQ, disable IRQ

INT FIQ, IRQ; // Enable IRQ, FIQ

INT OFF; // Disable both IRQ and FIQ

NOP — No Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
NOP	xxxxxxxxxxxxxxxx	-	4	-	-	-	-	-

Description: The instruction will generate waiting time of 4 cycles for delay and other purpose. This is implemented as an unconditional jump to next address.

Example:

```
Delay_Loop:
        NOP;                    // Waiting
        CMP R1, 0xFFFF;         // Search for end waiting flags
        JA Exit_Loop;           // End waiting
        R1 += 1;                // Waiting for delay counting
        JMP Delay_Loop;
Exit_Loop:
```

3 *unSP* -1.0 Instruction Set

3.1 General Description

unSP 1.0 instruction set is the same as unSP 1.1 instruction set except for the instruction format, cycles, and affected flags. So, while introducing unSP 1.0 instruction set, instruction format, cycles, and affected flags are mainly described.

3.2 *unSP*-1.0 Instruction Cycles

LOAD — **Load Register with Memory/Immediate/Register**

Syntax	Instruction Format							Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1										N	Z	S	C
Rd = IM6	xxxx	Rd	xxx	IM6				-	3	IM6	√	√	-	-
Rd = IM16	xxxx	Rd	xxxxxx			Rs		IM16	6 /8	IM16				
Rd = [BP+IM6]	xxxx	Rd	xxx	IM6				-	8	[BP+IM6]				
Rd = [A6]	xxxx	Rd	xxx	A6				-	6 / 8	[A6]				
Rd = [A16]	xxxx	Rd	xxxxxx			Rs		A16	9 / 11	[A16]				
Rd = Rs	xxxx	Rd	xxxxxx			Rs		-	3 / 8	R				
Rd = {D:}[Rs] Rd = {D:}[++Rs] Rd = {D:}[Rs--] Rd = {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs		-	7 / 9	[R]				

STORE — **Store Register into Memory**

Syntax	Instruction Format							Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1										N	Z	S	C
[BP+IM6] = Rd	xxxx	Rd	xxx	IM6				-	8	[BP+IM6]	-	-	-	-
[A6] = Rd	xxxx	Rd	xxx	A6				-	6 / 8	[A6]				
[A16] = Rd	xxxx	Rd	xxxxxx			Rs		A16	9 / 11	[A16]				
{D:}[Rs] = Rd {D:}[++Rs] = Rd {D:}[Rs--] = Rd {D:}[Rs++] = Rd	xxxx	Rd	xxx	D	@	Rs		-	7 / 9	[R]				

PUSH — Push Registers onto Stack

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
PUSH Rx, Ry to [Rs] Or PUSH Rx to [Rs]	xxxx	Rd	xxx	SIZE	Rs	-	3n+4	[R]	-	-	-	-

POP — Pop Registers from Stack

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
POP Rx, Ry from [Rs] Or POP Rx from [Rs]	xxxx	Rd	xxx	SIZE	Rs	-	3n+4 / 3n+6	[R]	*	*	*	*

ADD — Add without Carry

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += IM6 Rd = Rd + IM6	xxxx	Rd	xxx	IM6			-	3	IM6	√	√	√	√
Rd = Rs + IM16	xxxx	Rd	xxxxxx			Rs	IM16	6 / 8	IM16				
Rd += [BP+IM6] Rd = Rd + [BP+IM6]	xxxx	Rd	xxx	IM6				8	[BP+IM6]				
Rd += [A6] Rd = Rd + [A6]	xxxx	Rd	xxx	A6			-	6 / 8	[A6]				
Rd = Rs + [A16]	xxxx	Rd	xxxxxx			Rs	A16	9 / 11	[A16]				
Rd += Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 8	R				
Rd += {D:}[Rs] Rd += {D:}[++Rs] Rd += {D:}[Rs--] Rd += {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

ADC — Add with Carry

Syntax	Instruction Format							Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1										N	Z	S	C
Rd += IM6, Carry Rd = Rd + IM6, Carry	xxxx	Rd	xxx	IM6				-	3	IM6	√	√	√	√
Rd = Rs + IM16, Carry	xxxx	Rd	xxxxxx			Rs		IM16	6 / 8	IM16				
Rd += [BP+IM6], Carry Rd = Rd + [BP+IM6], Carry	xxxx	Rd	xxx	IM6				-	8	[BP+IM6]				
Rd += [A6], Carry Rd = Rd + [A6], Carry	xxxx	Rd	xxx	A6				-	6 / 8	[A6]				
Rd = Rs + [A16], Carry	xxxx	Rd	xxxxxx			Rs		A16	9 / 11	[A16]				
Rd += Rs, Carry	xxxx	Rd	xxxxxx			Rs		-	3 / 8	R				
Rd += {D:}[Rs], Carry Rd += {D:}[++Rs], Carry Rd += {D:}[Rs--], Carry Rd += {D:}[Rs++], Carry	xxxx	Rd	xxx	D	@	Rs		-	7 / 9	[R]	√	√	√	√

SUB — Subtract without Carry

Syntax	Instruction Format							Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1										N	Z	S	C
Rd -= IM6 Rd = Rd - IM6	xxxx	Rd	xxx	IM6				-	3	IM6	√	√	√	√
Rd = Rs - IM16	xxxx	Rd	xxxxxx			Rs		IM16	6 / 8	IM16				
Rd -= [BP+IM6] Rd = Rd - [BP+IM6]	xxxx	Rd	xxx	IM6				-	8	[BP+IM6]				
Rd -= [A6] Rd = Rd - [A6]	xxxx	Rd	xxx	A6				-	6 / 8	[A6]				
Rd = Rs - [A16]	xxxx	Rd	xxxxxx			Rs		A16	9 / 11	[A16]				
Rd -= Rs	xxxx	Rd	xxxxxx			Rs		-	3 / 8	R				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= {D:}[Rs] Rd -= {D:}[++Rs] Rd -= {D:}[Rs--] Rd -= {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

SBC **Subtract with Carry**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= IM6, Carry Rd = Rd - IM6, Carry	xxxx	Rd	xxx	IM6			-	3	IM6	√	√	√	√
Rd = Rs - IM16, Carry	xxxx	Rd	xxxxxx			Rs	IM16	6 / 8	IM16				
Rd -= [BP+IM6], Carry Rd = Rd - [BP+IM6], Carry	xxxx	Rd	xxx	IM6			-	8	[BP+IM6]				
Rd -= [A6], Carry Rd = Rd - [A6], Carry	xxxx	Rd	xxx	A6			-	6 / 8	[A6]				
Rd = Rs + [A16], Carry	xxxx	Rd	xxxxxx			Rs	A16	9 / 11	[A16]				
Rd -= Rs, Carry	xxxx	Rd	xxxxxx			Rs	-	3 / 8	R				
Rd -= {D:}[Rs], Carry Rd -= {D:}[++Rs], Carry Rd -= {D:}[Rs--], Carry Rd -= {D:}[Rs++], Carry	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

NEG **Negative**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd = -IM6	xxxx	Rd	xxx	IM6			-	3	IM6	√	√	-	-
Rd = -IM16	xxxx	Rd	xxxxxx			Rs	IM16	6 / 8	IM16				
Rd = -[BP+IM6]	xxxx	Rd	xxx	IM6			-	8	[BP+IM6]				
Rd = -[A6]	xxxx	Rd	xxx	A6			-	6 / 8	[A6]				
Rd = -[A16]	xxxx	Rd	xxxxxx			Rs	A16	9 / 11	[A16]				
Rd = -Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 8	R				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd = -{D:}[Rs] Rd = -{D:}[++Rs] Rd = -{D:}[Rs--] Rd = -{D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

CMP **Compare**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
CMP Rd, IM6	xxxx	Rd	xxx	IM6			-	3	IM6	√	√	√	√
CMP Rs, IM16	xxxx	Rd	xxxxxx			Rs	IM16	6 / 8	IM16				
CMP Rd, [BP+IM6]	xxxx	Rd	xxx	IM6			-	8	[BP+IM6]				
CMP Rd, [A6]	xxxx	Rd	xxx	A6			-	6 / 8	[A6]				
CMP Rs, [A16]	xxxx	Rd	xxxxxx			Rs	A16	9 / 11	[A16]				
CMP Rd, Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 8	R				
CMP Rd, {D:}[Rs] CMP Rd, {D:}[++Rs] CMP Rd, {D:}[Rs--] CMP Rd, {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

MUL **Register Multiplication**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
MR = Rd * Rs {, ss} MR = Rd * Rs, us	xxxx	Rd	S	xxxxx	Rs	-	12	R	-	-	-	-

MULS **Sum of Register Multiplication**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

MR = [Rd] * [Rs] {,ss} {,n} MR = [Rd] * [Rs], us {,n}	xxxx	Rd	S	x	SIZE	Rs	-	10n+ 8	[R]	-	-	-	-

AND — Logical AND

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
Rd &= IM6 Rd = Rd & IM6	xxxx	Rd	xxx	IM6				-	3	IM6	√	√	-	-
Rd = Rs & IM16	xxxx	Rd	xxxxxx			Rs		IM16	6 /8	IM16				
Rd &= [BP+IM6] Rd = Rd & [BP+IM6]	xxxx	Rd	xxx	IM6				-	8	[BP+IM6]				
Rd &= [A6] Rd = Rd & [A6]	xxxx	Rd	xxx	A6				-	6 /8	[A6]				
Rd = Rs & [A16]	xxxx	Rd	xxxxxx			Rs		A16	9 /11	[A16]				
Rd &= Rs	xxxx	Rd	xxxxxx			Rs		-	3 /8	R				
Rd &= {D:}[Rs] Rd &= {D:}[++Rs] Rd &= {D:}[Rs--] Rd &= {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs		-	7 /9	[R]				

OR — Logical Inclusive OR

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
Rd |= IM6 Rd = Rd | IM6	xxxx	Rd	xxx	IM6				-	3	IM6	√	√	-	-
Rd = Rs | IM16	xxxx	Rd	xxxxxx			Rs		IM16	6 / 8	IM16				
Rd |= [BP+IM6] Rd = Rd | [BP+IM6]	xxxx	Rd	xxx	IM6				-	8	[BP+IM6]				
Rd |= [A6] Rd = Rd | [A6]	xxxx	Rd	xxx	A6				-	6 / 8	[A6]				
Rd = Rs | [A16]	xxxx	Rd	xxxxxx			Rs		A16	9 / 11	[A16]				
Rd |= Rs	xxxx	Rd	xxxxxx			Rs		-	3 / 8	R				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd |= {D:}[Rs] Rd |= {D:}[++Rs] Rd |= {D:}[Rs--] Rd |= {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

XOR **Logical Exclusive OR**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd ^= IM6 Rd = Rd ^ IM6	xxxx	Rd	xxx	IM6			-	3	IM6	√	√	-	--
Rd = Rs ^ IM16	xxxx	Rd	xxxxxx			Rs	IM16	6 / 8	IM16				
Rd ^= [BP+IM6] Rd = Rd ^ [BP+IM6]	xxxx	Rd	xxx	IM6			-	8	[BP+IM6]				
Rd ^= [A6] Rd = Rd ^ [A6]	xxxx	Rd	xxx	A6			-	6 / 8	[A6]				
Rd = Rs ^ [A16]	xxxx	Rd	xxxxxx			Rs	A16	9 / 11	[A16]				
Rd ^= Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 8	R				
Rd ^= {D:}[Rs] Rd ^= {D:}[++Rs] Rd ^= {D:}[Rs--] Rd ^= {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

TEST **Logical Test**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
TEST Rd, IM6	xxxx	Rd	xxx	IM6			-	3	IM6	√	√	-	-
TEST Rs, IM16	xxxx	Rd	xxxxxx			Rs	IM16	6 / 8	IM16				
TEST Rd, [BP+IM6]	xxxx	Rd	xxx	IM6			-	8	[BP+IM6]				
TEST Rd, [A6]	xxxx	Rd	xxx	A6			-	6 / 8	[A6]				
TEST Rs, [A16]	xxxx	Rd	xxxxxx			Rs	A16	9 / 11	[A16]				
TEST Rd, Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 8	R				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
TEST Rd, {D:}[Rs] TEST Rd, {D:}[++Rs] TEST Rd, {D:}[Rs--] TEST Rd, {D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	-	7 / 9	[R]				

ASR-ALU **Register Arithmetic-Shift-Right and Arithmetic/Logical Operation**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += Rs ASR nn {,Carry} Rd -= Rs ASR nn {,Carry} CMP Rd, Rs ASR nn	xxxx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	√	√
Rd = - Rs ASR nn Rd &= Rs ASR nn Rd |= Rs ASR nn Rd ^= Rs ASR nn TEST Rd, Rs ASR nn Rd = Rs ASR nn	xxxx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	-	-

LSL-ALU **Register Logical-Shift-Left and Arithmetic/Logical Operation**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += Rs LSL nn {,Carry} Rd -= Rs LSL nn {,Carry} CMP Rd, Rs LSL nn	xxxx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	√	√

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd = - Rs LSL nn Rd &= Rs LSL nn Rd |= Rs LSL nn Rd ^= Rs LSL nn TEST Rd, Rs LSL nn Rd = Rs LSL nn	xxxx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	-	-

LSR-ALU **Register Logical-Shift-Right and Arithmetic/Logical Operation**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += Rs LSR nn {,Carry} Rd -= Rs LSR nn {,Carry} CMP Rd, Rs LSR nn	xxx x	Rd	xx	nn	Rs	-	3 / 8	R	√	√	√	√
Rd = - Rs LSR nn Rd &= Rs LSR nn Rd |= Rs LSR nn Rd ^= Rs LSR nn TEST Rd, Rs LSR nn Rd = Rs LSR nn	xxx x	Rd	xx	nn	Rs	-	3 / 8	R	√	√	-	-

ROL-ALU **Register Rotate-Left and Arithmetic/Logical Operation**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += Rs ROL nn {,Carry} Rd -= Rs ROL nn {,Carry} CMP Rd, Rs ROL nn	xx xx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	√	√
Rd = - Rs ROL nn Rd &= Rs ROL nn Rd |= Rs ROL nn Rd ^= Rs ROL nn TEST Rd, Rs ROL nn Rd = Rs ROL nn	xx xx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	-	-

ROR-ALU — Register Rotate-Right and Arithmetic/Logical Operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += Rs ROR nn {,Carry} Rd -= Rs ROR nn {,Carry} CMP Rd, Rs ROR nn	xxxx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	√	√
Rd = - Rs ROR nn Rd &= Rs ROR nn Rd |= Rs ROR nn Rd ^= Rs ROR nn TEST Rd, Rs ROR nn Rd = Rs ROR nn	xxxx	Rd	xx	nn	Rs	-	3 / 8	R	√	√	-	-

BREAK — Software Interrupt

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
BREAK	xxxxxxxxxxxxxxxx	-	13	[A16]	-	-	-	-

CALL — Segmented Far Call

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
CALL label	xxxxxxxxxx	CS6	A16	13	[A22]	-	-	-	-

JUMP — Conditional/Unconditional Jump

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
Jcond label JMP label	COND	xxxxxx	S	IM6	-	3 (not-taken) / 5 (taken)	PC+IM6	-	-	-	-

RETF — **Return from Subroutine**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
RETF	xxxxxxxxxxxxxxxx	-	12	[A22]	√	√	√	√

RETI — **Return from Interrupt Service Routine**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
RETI	xxxxxxxxxxxxxxxx	-	12	[A22]	√	√	√	√

GOTO — **Unconditional Far Jump in Page**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
GOTO label	xxxxxxxxxxxxxxxx	A16	12	[A16]	-	-	-	-

FIR_MOV ON — **Enable Automatic Data Movement for FIR Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
FIR_MOV ON	xxxxxxxxxxxxxxxx	-	3	-	-	-	-	-

FIR_MOV OFF — **Disable Automatic Data Movement for FIR Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
FIR_MOV OFF	xxxxxxxxxxxxxxxx	-	3	-	-	-	-	-

FIQ ON — **Enable FIQ**

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
FIQ ON	xxxxxxxxxxxxxx	F	I	-	3	-	-	-	-	-

FIQ OFF — **Disable FIQ**

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
FIQ OFF	xxxxxxxxxxxxxx	F	I	-	3	-	-	-	-	-

IRQ ON — **Enable IRQ**

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
IRQ ON	xxxxxxxxxxxxxx	F	I	-	3	-	-	-	-	-

IRQ OFF — **Disable IRQ**

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
IRQ OFF	xxxxxxxxxxxxxx	F	I	-	3	-	-	-	-	-

INT — **Interrupt Set**

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
INT FIQ INT IRQ INT FIQ, IRQ INT OFF	xxxxxxxxxxxxxx	F	I	-	3	-	-	-	-	-

NOP — **No Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
NOP	xxxxxxxxxxxxxxxx	-	5	-	-	-	-	-

4 *unSP* -1.2 Instruction Set

4.1 unSP-1.2 Instruction Set

4.1.1 Data-Transfer Instructions

LOAD	Load Register with Memory/Immediate/Register

Syntax	Instruction Format										Cycles	Addressing Mode	Flags			
	Word Group 1									Word Group 2			N	Z	S	C
Rd = IM6	xxxx	Rd	xxx	IM6						-	2	IM6				
Rd = IM16	xxxx	Rd	xxxxxx					Rs		IM16	4 / 5	IM16				
Rd = [BP+IM6]	xxxx	Rd	xxx	IM6						-	6	[BP+IM6]				
Rd = [A6]	xxxx	Rd	xxx	A6						-	5 / 6	[A6]	√	√	-	-
Rd = [A16]	xxxx	Rd	xxxxx			W		Rs		A16	7 / 8	[A16]				
Rd = Rs	xxxx	Rd	xxxxxx					Rs		-	3 / 5	R				
Rd = {D:}[Rs@]	xxxx	Rd	xxx	D		@		Rs		-	6 / 7	[R]				

Description: The group of instruction will be executed for reading of data transmitting, i.e. Rd=X. X shows different form according to addressing mode. The prefix of source register

Rs@	Meaning
Rs	No increment/decrement
Rs--	After load, Rs= Rs-1
Rs++	After load, Rs= Rs+1
++Rs	Before load, Rs= Rs+1

Examples:

```
R1 = 0x28;                  // IM6

R2 = 0x2400;                // IM 16

R3 = [BP+0x08];             // [BP+IM6]

R4 = [0x30];                // A6

BP = [0x2480];              // A16

SR = R2;                    // R

PC = D:[R1++];              // [R], Write to PC, cycles:7
```

STORE — Store Register into Memory

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
[BP+IM6] = Rd	xxxx	Rd	xxx	IM6				-	6	[BP+IM6]	-	-	-	-
[A6] = Rd	xxxx	Rd	xxx	A6				-	5 / 6	[A6]				
[A16] = Rd	xxxx	Rd	xxxxx		W	Rs		A16	7 / 8	[A16]				
{D:}[Rs@] = Rd	xxxx	Rd	xxx	D	@	Rs		-	6 / 7	[R]				

Description: The group of instruction will be executed for writing of data transmitting, i.e. X=Rd. X shows different form according to addressing mode. The prefix of source register:

Rs@	Meaning
Rs	No increment/decrement
Rs--	After load, Rs= Rs-1
Rs++	After load, Rs= Rs+1
++Rs	Before load, Rs= Rs+1

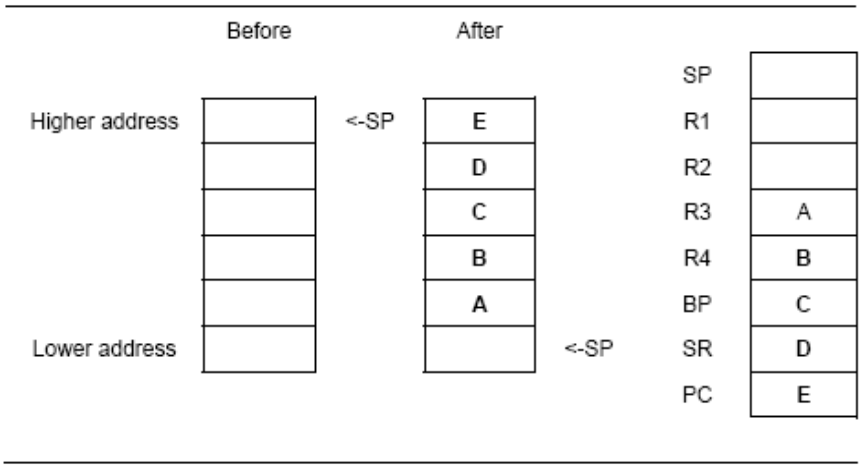
Example:
```
[BP+0x08] = R3;          // Write to [BP+IM6]
[0x30] = R4;             // Write to [A6]
[0x2480] = BP;           // Write to [A16]
D:[R4++] = PC;           // Read from PC, Cycles 7
```

PUSH — Push Registers onto Stack

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
PUSH Rx, Ry to [Rs] Or PUSH Rx to [Rs]	xxxx	Rd	xxx	n	Rs	-	2n+4	[R]	-	-	-	-

Description: Push a number (number n=1~7, SIZE) of register Rx-Ry (Rx~RySP) to memory pointed by Rs decreasingly.

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



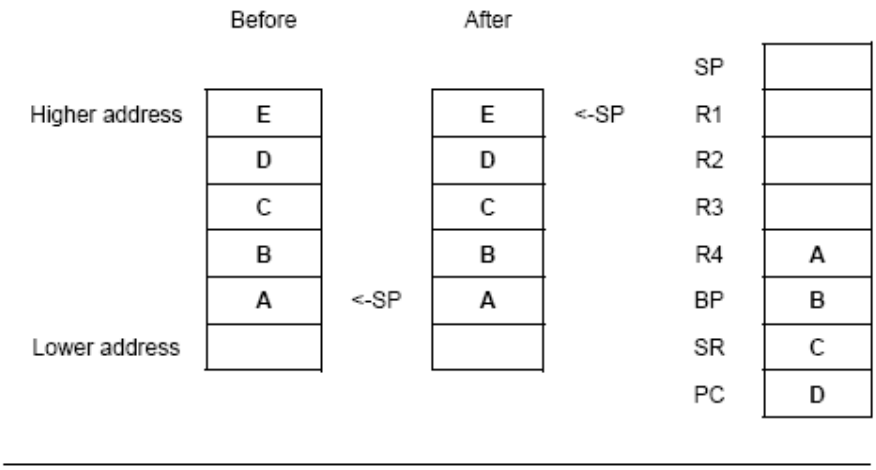
Note: PUSH R1, BP to [SP] is equivalent to PUSH BP, R1 to [SP].

POP	Pop Registers from Stack

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
POP Rx, Ry from [Rs] Or POP Rx from [Rs]	xxxx	Rd	xxx	n	Rs	-	2n+4	[R]	-	-	-	-

Description: Copy a set of memory pointed by Rs consecutively to a set of register Rx-Ry (Rx~Ry SP) where n=1~7.

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



4.1.2 Data Processing Instructions

Data Processing Instructions include ALU Operation, Bit Operation, Shift Operation, Mul Operation, Div Operation, EXP Operation, NOP, etc..

ALU Operation Instructions that carry out the operation as RD = X # Y. X and Y will show different meanings according to the addressing mode. Because the same explanation for X, Y and the description for Rs, Rd will be involved in instruction they will be listed in Table 4.1.

Table 4.1 The meanings for X, Y in operation as Rd = X # Y

Addressing Mode	X, Y
IM6	X is Rd, Y is IM6. IM6 will be expanded to 16-bit filled with zeros first, and then be operated with X.
IM16	X is Rs, Y is IM16
[BP+IM6]	X is Rd, Y is the memory in PAGE0 addressed as (BP+IM6)
[A6]	X is Rd, Y is the memory in PAGE0 addressed as (0x00~0x3F)
[A16]	X is Rs. Y is the memory in PAGE0 addressed as (0x0000~0xFFFF)
R	X is Rd, Y is Rs.
{D:}[R] {D:}[R--] {D:}[R++] {D:}[++R]	X is Rd, Y is the memory address pointed by the offset in Rs. Rs may point data segment in PAGE0 as 'D' is ignored or in non-PAGE0 as 'D' is not ignored and its page index depends on DS in SR register. Rs can be increased by 1 before ALU operation or increased/decreased by 1 after ALU operation.

Note:

- For 16-bit direct memory addressing, there are two kinds of instruction format:
  - Rd = Rs # [A16]; (W=0)
  - [A16] = Rd # Rs; (W=1)
- On the Cycles column, the number after '/' denotes writing to PC.
- The prefix of source register:

Rs@	Meaning
Rs	No increment/decrement
Rs--	After ALU_OP, Rs=Rs-1
Rs++	After ALU_OP, Rs=Rs+1
++Rs	Before ALU_OP, Rs=Rs+1

ADD — ADD without Carry

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += IM6 Rd = Rd + IM6	xxxx	Rd	xxx	IM6			-	2	IM6	√	√	√	√
Rd = Rs + IM16	xxxx	Rd	xxxxxx		Rs		IM16	4 / 5	IM16				
Rd += [BP+IM6] Rd = Rd + [BP+IM6]	xxxx	Rd	xxx	IM6				6	[BP+IM6]				
Rd += [A6] Rd = Rd + [A6]	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs + [A16] [A16] = Rd + Rs	xxxx	Rd	xxxxx		W	Rs	A16	7 / 8	[A16]				
Rd += Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				
Rd += {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for addition operation without carry, i.e. Rd = X + Y. X, Y will have different meanings according to the addressing mode. See Table 4.1.

Example:

```
R1 += 0x28;                 // IM6
R2 = R1 + 0x2400;           // IM16
R3 += [BP+0x08];            // [BP+IM6]
R4 += [0x30];               // [A6]
BP = R4 + [0x2480];         // [A16]
[0x2480] = BP + R2;         // [A16], BP + R2 is assigned to MEM[0x2480]
SR += R2;                   // R
PC += D:[BP++];             // Write to PC, cycles: 7
```

ADC — Add with Carry

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += IM6, Carry Rd = Rd + IM6, Carry	xxxx	Rd	xxx	IM6		-	2	IM6	√	√	√	√
Rd = Rs + IM16, Carry	xxxx	Rd	xxxxxx		Rs	IM16	4 / 5	IM16				

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += [BP+IM6], Carry Rd = Rd + [BP+IM6], Carry	xxxx	Rd	xxx	IM6		-	6	[BP+IM6]				
Rd += [A6], Carry Rd = Rd + [A6], Carry	xxxx	Rd	xxx	A6		-	5 / 6	[A6]				
Rd = Rs + [A16], Carry [A16] = Rd + Rd, Carry	xxxx	Rd	xxxxx	W	Rs	A16	7 / 8	[A16]				
Rd += Rs, Carry	xxxx	Rd	xxxxxx		Rs	-	3 / 5	R				
Rd += {D:}[Rs@], Carry	xxxx	Rd	xxx	D	@ Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for addition with carry in arithmetical operation, i.e. Rd = X+Y+C. X, Y will have different meanings according to the addressing mode. See Table 4.1.

```
Example:  R1 = 0x28, Carry;                 // R1 = R1 + IM6 + C
          R2 = R1 + 0x2400, Carry;          // R2 = R1 + IM16 + C
          R3 += [BP+0x08], Carry;           // R3 = R3 + [BP+IM6] + C
          R4 += [0x30];                     // R4 = R4 + [A6] + C
          BP = R4 + [0x2480], Carry;        // BP = R4 + [A16] + C
          [0x2480] = BP + R2, Carry;        // [A16], BP + R2 + C is assigned to MEM[0x2480]
          SR += R2, Carry;                  // SR = SR + R2 + C
          PC += D:[BP++], Carry;            // Write to PC, cycles: 7
```

SUB **Subtract without Carry**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd -= IM6 Rd = Rd – IM6	xxxx	Rd	xxx	IM6		-	2	IM6	√	√	√	√
Rd = Rs – IM16	xxxx	Rd	xxxxxx		Rs	IM16	4 / 5	IM16				
Rd -= [BP+IM6] Rd = Rd - [BP+IM6]	xxxx	Rd	xxx	IM6		-	6	[BP+IM6]				
Rd -= [A6] Rd = Rd - [A6]	xxxx	Rd	xxx	A6		-	5 / 6	[A6]				

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd = Rs - [A16] [A16] = Rd - Rs	xxxx	Rd	xxxxx	W	Rs		A16	7 / 8	[A16]				
Rd -= Rs	xxxx	Rd	xxxxxx		Rs		-	3 / 5	R				
Rd -= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for subtraction without carry in arithmetical operation, i.e. Rd = X - Y. X, Y will have different meanings according to the addressing mode. See Table 4.1.

Example:
```
R1 -= 0x28;          // R1 = R1 – IM6
R2 = R1 - 0x2400;    // R2 = R1 – IM16
R3 -= [BP+0x08];     // R3 = R3 – [BP+IM6]
R4 -= [0x30];        // R4 = R4 – [A6]
BP = R4 - [0x2480];  // BP = R4 – [A16]
[0x2480] = BP - R4;  // [A16] = BP – R4
SR -= R2;            // SR = SR – R2
PC -= D:[BP++];      // Write to PC, cycles: 7
```

SBC — Subtract with Carry

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= IM6, Carry Rd = Rd - IM6, Carry	xxxx	Rd	xxx	IM6			-	2	IM6	√	√	√	√
Rd = Rs - IM16, Carry	xxxx	Rd	Xxxxxx		Rs		IM16	4 / 5	IM16				
Rd -= [BP+IM6], Carry Rd = Rd - [BP+IM6], Carry	xxxx	Rd	xxx	IM6			-	6	[BP+IM6]				
Rd -= [A6], Carry Rd = Rd - [A6], Carry	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs - [A16], Carry [A16] = Rd – Rs, Carry	xxxx	Rd	xxxxxx		W	Rs	A16	7 / 8	[A16]				
Rd -= Rs, Carry	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				
Rd -= {D:}[Rs@], Carry	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed for subtraction with carry in arithmetical operation, i.e. Rd = X – Y – C = X + (~Y) + C. X, Y will have different meanings according to the addressing mode. See Table 4.1.

Example:

```
R1 -= 0x20, Carry;              // R1 = R1 – IM6 - C
R2 = R1 - 0x2400, Carry;        // R2 = R1 – IM16 - C
R3 -= [BP+0x08], Carry;         // R3 = R3 – [BP+IM6] - C
R4 -= [0x30], Carry;            // R4 = R4 – [A6] - C
BP = R4 - [0x2480], Carry;      // BP = R4 – [A16] - C
[0x2480] = BP - R4, Carry;      // [A16] = BP – R4 - C
SR -= R2, Carry;                // SR = SR – R2 - C
PC -= D:[BP++], Carry;          // Write to PC, cycles: 7
```

NEG — Negative

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
Rd = -IM6	xxxx	Rd	xxx	IM6				-	2	IM6	√	√	-	-
Rd = -IM16	xxxx	Rd	xxxxxx			Rs		IM16	4 / 5	IM16				
Rd = -[BP+IM6]	xxxx	Rd	xxx	IM6				-	6	[BP+IM6]				
Rd = -[A6]	xxxx	Rd	xxx	A6				-	5 / 6	[A6]				
Rd = -[A16] [A16] = -Rd	xxxx	Rd	xxxxx		W	Rs		A16	7 / 8	[A16]				
Rd = -Rs	xxxx	Rd	xxxxxx			Rs		-	3 / 5	R				
Rd = -{D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs		-	6 / 7	[R]				

Description: The group of instruction will be executed for negation in arithmetical operation, i.e. Rd = -X = ~X+1. The meaning of X will be described as follow according to the different addressing modes. See Table 4.1

Example:

```
R1 = -0x27;                     // R1 = - IM6
R3 = -[BP+0x08];                // R3 = - [BP+IM6]
R4 = -[0x30];                   // R4 = - [A6]
BP = -[0x2480];                 // BP = - [A16]
[0x2480] = -BP;                 // [A16] = - BP
SR = -R2;                       // SR = - R2
PC = -D:[BP++];                 // Write to PC, cycles: 7
```

CMP	Compare

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
CMP Rd, IM6	xxxx	Rd	xxx	IM6				-	2	IM6	√	√	√	√
CMP Rs, IM16	xxxx	Rd	xxxxxx			Rs		IM16	4 / 5	IM16				
CMP Rd, [BP+IM6]	xxxx	Rd	xxx	IM6				-	6	[BP+IM6]				
CMP Rd, [A6]	xxxx	Rd	xxx	A6				-	5 / 6	[A6]				
CMP Rs, [A16] CMP Rd, Rs	xxxx	Rd	xxxxx		W	Rs		A16	7 / 8	[A16]				
CMP Rd, Rs	xxxx	Rd	xxxxxx			Rs		-	3 / 5	R				
CMP Rd, {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs		-	6 / 7	[R]				

Description: The group of instruction will be executed for comparison in arithmetical operation, i.e. X - Y. But its result will not be stored and only affect NZSC flags. X, Y will have different meanings according to the addressing mode. See Table 4.1.

Example:
```
CMP R1, 0x27;          // Compare R1, IM6
CMP R3, [BP+0x08];     // Compare R3, [BP+IM6]
CMP R4, [0x30];        // Compare R4, [A6]
CMP BP, [0x2480];      // Compare BP, [A16]
CMP SR, R2;            // Compare SR, R2
CMP PC, D:[BP++];      // Compare with PC, cycles: 7
```

MUL	Register Multiplication

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
MR = Rd * Rs {,ss/,us/,uu}	xx	S	Rd	S	xxxxx	Rs	-	12	R	-	-	-	-
	x	s		d									

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).

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 assembled the same as "MR = R2 * R1, us".

Example:
```
MR = R2 * R1;          // Two signed values
MR = R1 * R2, us;      // R1 is unsigned and R2 is signed
MR = R3 * R4, ss;      // Two signed values
MR = R3 * R4, uu;      // Two unsigned values
```

MULS	Sum of Register Multiplication

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
MR = [Rd] * [Rs] {,ss/,us/,uu}, N	xxx	Ss	Rd	Sd	x	N	Rs	-	10N+6	[R]	-	-	-	-

Description: This operation is using a 36-bit 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-bit 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.

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*-1.0/*unSP*-1.1 doesn't change the sign flag, but the *unSP*-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 assembled the same as "MR = [R2] * [R1], us, 4".

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

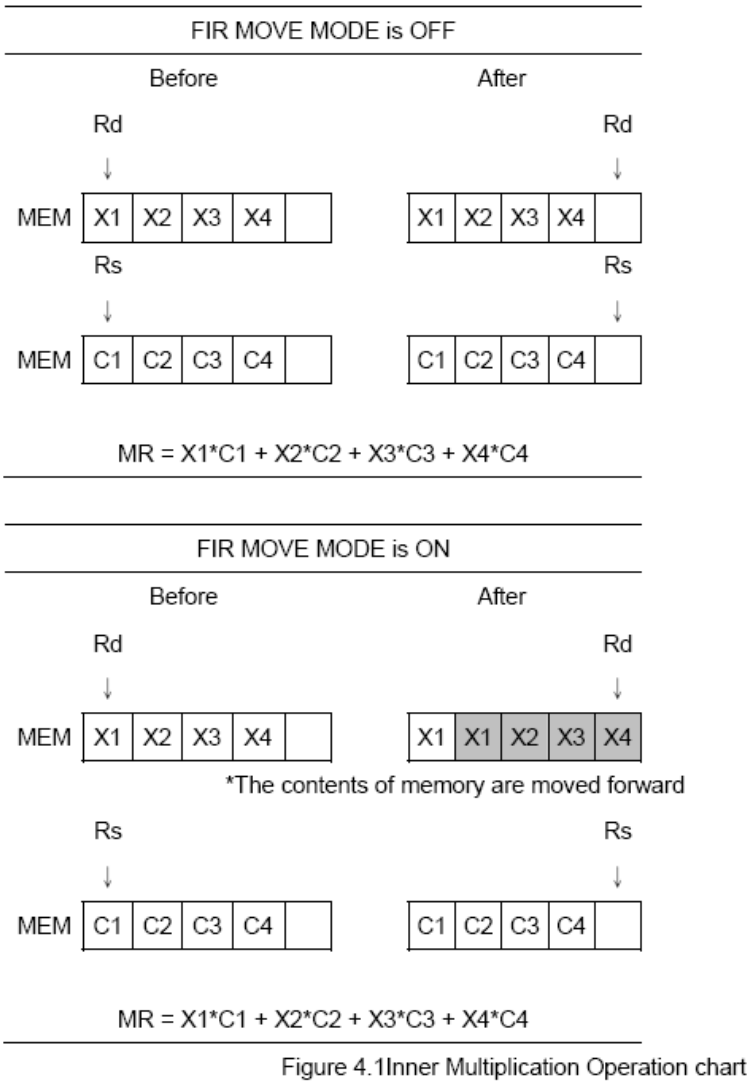


Figure 4.1Inner Multiplication Operation chart

Example:

```
MR = [R2] * [R1], 8;        // The inner multiplication of two signed
MR = [R1] * [R2], us, 2;    // R1 is unsigned, R2 is signed.
MR = [R2] * [BP], ss, 4;    // Two signed value.
MR = [R2] * [BP], uu, 4;    // Two unsigned value.
```

AND — Logical AND

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
Rd &= IM6 Rd = Rd & IM6	xxxx	Rd	xxx	IM6	-	2	IM6	√	√	-	-

Rd = Rs & IM16	xxxx	Rd	xxxxxx			Rs	IM16	4 / 5	IM16		
Rd &= [BP+IM6] Rd = Rd & [BP+IM6]	xxxx	Rd	xxx	IM6			-	6	[BP+IM6]		
Rd &= [A6] Rd = Rd & [A6]	xxxx	Rd	xxx	A6			-	5 / 6	[A6]		
Rd = Rs & [A16] [A16] = Rd & Rs	xxxx	Rd	xxxxx		W	Rs	A16	7 / 8	[A16]		
Rd &= Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R		
Rd &= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]		

Description: The group of instruction will be executed in logical AND operation, i.e. Rd = X & Y. The X and Y will have different meanings according to the addressing mode. See Table 4.1.

Example:

```
R1 &= 0x2F;                 // R1 = R1 & IM6
R3 &= [BP+0x08];            // R3 = R3 & [BP+IM6]
R4 &= [0x30];               // R4 = R4 & [A6]
BP = R2 & [0x2480];         // BP = R2 & [A16]
[0x2480] = R2 & BP;         // [A16] = R2 & BP
SR &= R2;                   // SR = SR & R2
PC &= D:[BP++];             // Write to PC, cycles: 7
```

OR — Logical Inclusive OR

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd |= IM6 Rd = Rd | IM6	xxxx	Rd	xxx	IM6			-	2	IM6	√	√	-	-
Rd = Rs | IM16	xxxx	Rd	xxxxxx			Rs	IM16	4 / 5	IM16				
Rd |= [BP+IM6] Rd = Rd | [BP+IM6]	xxxx	Rd	xxx	IM6			-	6	[BP+IM6]				
Rd |= [A6] Rd = Rd | [A6]	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs | [A16] [A16] = Rd | Rs	xxxx	Rd	xxxxx		W	Rs	A16	7 / 8	[A16]				
Rd |= Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				
Rd |= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed in logical OR operation, i.e. Rd = X | Y. The X and Y will have different meanings according to the addressing mode. See Table 4.1.

Example:
```
R1 |= 0x2F;              // R1 = R1 | IM6
R3 |= [BP+0x08];         // R3 = R3 | [BP+IM6]
R4 |= [0x30];            // R4 = R4 | [A6]
BP = R2 | [0x2480];      // BP = R2 | [A16]
[0x2480] = R2 | BP;      // [0x2480] = R2 | BP
SR |= R2;                // SR = SR | R2
PC |= D:[BP++];          // Write to PC, cycles: 7
```

XOR — Logical Exclusive OR

Syntax	Instruction Format: Word Group 1						Instruction Format: Word Group 2	Cycles	Addressing Mode	Flags: N	Z	S	C
Rd ^= IM6 Rd = Rd ^ IM6	xxxx	Rd	xxx	IM6			-	2	IM6	√	√	-	--
Rd = Rs ^ IM16	xxxx	Rd	xxxxxx		Rs		IM16	4 / 5	IM16				
Rd ^= [BP+IM6] Rd = Rd ^ [BP+IM6]	xxxx	Rd	xxx	IM6			-	6	[BP+IM6]				
Rd ^= [A6] Rd = Rd ^ [A6]	xxxx	Rd	xxx	A6			-	5 / 6	[A6]				
Rd = Rs ^ [A16] [A16] = Rd ^ Rs	xxxx	Rd	xxxxx		W	Rs	A16	7 / 8	[A16]				
Rd ^= Rs	xxxx	Rd	xxxxxx			Rs	-	3 / 5	R				
Rd ^= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	6 / 7	[R]				

Description: The group of instruction will be executed in logical exclusive OR operation, i.e. Rd = X ^ Y. The X, Y will have different meanings according to the addressing mode. See Table 4.1.

Example:
```
R1 ^= 0x2F;              // R1 = R1 ^ IM6
R3 ^= [BP+0x08];         // R3 = R3 ^ [BP+IM6]
R4 ^= [0x30];            // R4 = R4 ^ [A6]
BP = R2 ^ [0x2480];      // BP = R2 ^ [A16]
[0x2480] = R2 ^ BP;      // [0x2480] = R2 ^ BP
SR ^= R2;                // SR = SR ^ R2
PC ^= D:[BP++];          // Write to PC, cycles: 7
```

TEST	Logical Test

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
TEST Rd, IM6	xxxx	Rd	xxx	IM6				-	2	IM6	√	√	-	-
TEST Rs, IM16	xxxx	Rd	xxxxxx			Rs		IM16	4 / 5	IM16				
TEST Rd, [BP+IM6]	xxxx	Rd	xxx	IM6				-	6	[BP+IM6]				
TEST Rd, [A6]	xxxx	Rd	xxx	A6				-	5 / 6	[A6]				
TEST Rs, [A16] TEST Rd, Rs	xxxx	Rd	xxxxx		W	Rs		A16	7 / 8	[A16]				
TEST Rd, Rs	xxxx	Rd	xxxxxx			Rs		-	3 / 5	R				
TEST Rd, {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs		-	6 / 7	[R]				

Description: The group of instruction will be executed for logical AND operation, i.e. X&Y. However, its result will not be stored and it only affects NZ flags. The X and Y will have different meanings according to the addressing mode. See Table 4.1.

Example:
```
TEST R1, 0x27;          // TEST R1 and IM6
TEST R3, [BP+0x08];     // TEST R3 and [BP+IM6]
TEST R4, [0x30];        // TEST R4 and [A6]
TEST BP, [0x2480];      // TEST BP and [A16]
TEST SR, R2;            // TEST SR and R2
TEST PC, D:[BP++];      // TEST PC and D:[BP++], cycles: 7
```

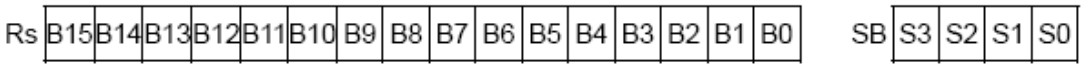
ASR-ALU	Register Arithmetic-Shift-Right and Arithmetic/Logical Operation

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += Rs ASR nn {,Carry} Rd -= Rs ASR nn {,Carry} CMP Rd, Rs ASR nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	√	√

Rd = - Rs ASR nn Rd &= Rs ASR nn Rd |= Rs ASR nn Rd ^= Rs ASR nn TEST Rd, Rs ASR nn Rd = Rs ASR nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	-	-

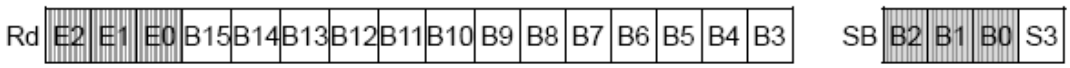
Description: The group of instruction will be executed in arithmetic operation with logical shift right where nn is number of shifting bits and ranged in [1~4].

Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of

ASR: (Arithmetic Shift Right with MSB, which fits for signed)



E2, E1, E0 are sign extension bit of the most significant bit in Rs.

Note: Carry flag only couples with ALU operation, not shift operation.

Example: SR |= R2 ASR 2; // SR = SR | (R2 / 2^2)

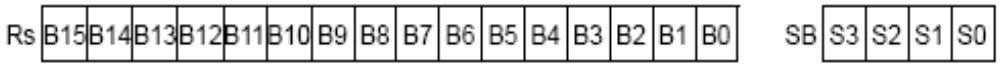
SP += R1 ASR 4, Carry; // SP = SP + (R1 / 2^4) + C

R2 = R1 ASR 2; // R2 = R1 / $_{22}$

LSL-ALU	Register Logical-Shift-Left and Arithmetic/Logical Operation

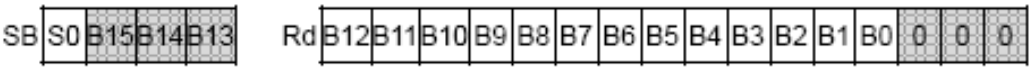
Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += Rs LSL nn {,Carry} Rd -= Rs LSL nn {,Carry} CMP Rd, Rs LSL nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs LSL nn Rd &= Rs LSL nn Rd |= Rs LSL nn Rd ^= Rs LSL nn TEST Rd, Rs LSL nn Rd = Rs LSL nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	-	-

Description: The group of instruction will be executed in arithmetic and logical operations with logical shift left where nn is number of shifting bits and ranged in [1~4].
Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of

LSL: (Logic Shift Left)



Note: Carry flag only couples with ALU operation, not shift operation.

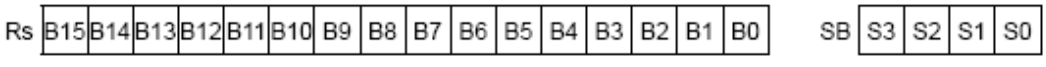
Example:
```
SR |= R2 LSL 2;                  // SR = SR | (R2 << 2)
SP += R1 LSL 4, Carry;          // SP = SP + (R1 << 4) + C
R2 = R1 LSL 2;                  // R2 = R1 << 2
```

LSR-ALU — Register Logical-Shift-Right and Arithmetic/Logical Operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += Rs LSR nn {,Carry} Rd -= Rs LSR nn {,Carry} CMP Rd, Rs LSR nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs LSR nn Rd &= Rs LSR nn Rd |= Rs LSR nn Rd ^= Rs LSR nn TEST Rd, Rs LSR nn Rd = Rs LSR nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	-	-

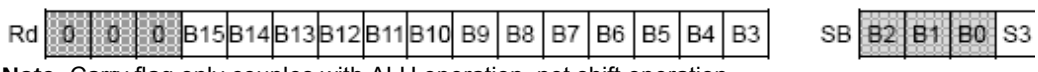
Description: The group of instruction will be executed with logical shift right where nn is number of shifting bits and ranged in [1~4].

Before shifting op:



SB is the shift buffer. Suppose nn=3, then after shift op of

LSR: (Logic Shift Right)



Note: Carry flag only couples with ALU operation, not shift operation.

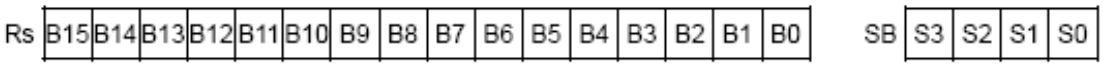
Example:
```
SR |= R2 LSR 2;              // SR = SR | (R2 >> 2)
SP += R1 LSR 4, Carry;       // SP = SP + (R1 >> 4) + C
R2 = R1 LSR 2;               // R2 = R1 >> 2
```

ROL-ALU — Register Rotate-Left and Arithmetic/Logical Operation

Syntax	Instruction Format: Word Group 1						Instruction Format: Word Group 2	Cycles	Addressing Mode	Flags: N	Z	S	C
Rd += Rs ROL nn {,Carry} Rd -= Rs ROL nn {,Carry} CMP Rd, Rs ROL nn	xxx x	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs ROL nn Rd &= Rs ROL nn Rd |= Rs ROL nn Rd ^= Rs ROL nn TEST Rd, Rs ROL nn Rd = Rs ROL nn	xxx x	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	-	-

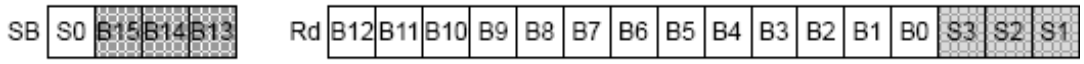
Description: The group of instruction will be executed in arithmetic and logical operations with rotate shift left where nn is number of position shift and ranged in [1~4].

Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of:

ROL: (Rotate Left with SB) SB



Note: Carry flag only couples with ALU operation, not shift operation.

Example:
```
SR |= R2 ROL 2;               // SR = SR | (R2 ROL 2)
SP += R1 ROL 4, Carry;       // SP = SP + (R1 ROL 4) + C
R2 = R1 ROL 2;               // R2 = R1 ROL 2
```

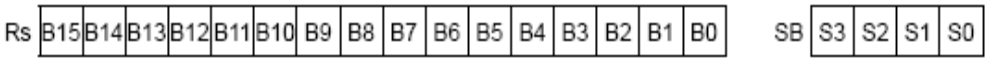
ROR-ALU — Register Rotate-Right and Arithmetic/Logical Operation

Syntax	Instruction Format: Word Group 1	Instruction Format: Word Group 2	Cycles	Addressing Mode	Flags: N	Z	S	C

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += Rs ROR nn {,Carry} Rd -= Rs ROR nn {,Carry} CMP Rd, Rs ROR nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	√	√
Rd = - Rs ROR nn Rd &= Rs ROR nn Rd |= Rs ROR nn Rd ^= Rs ROR nn TEST Rd, Rs ROR nn Rd = Rs ROR nn	xxxx	Rd	x	xxx	nn	Rs	-	3 / 5	R	√	√	-	-

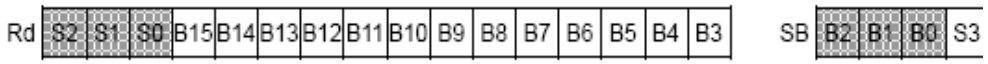
Description: The group of instruction will be executed in arithmetic and logical operations with rotate shift right where nn is number of shifting bits and ranged in [1~4].

Before shifting op:



SB is the shift buffer. Suppose nn=3, after shift op of

ROR: (Rotate Right with SB)



Note: Carry flag only couples with ALU operation, not shift operation.

Example:

```
SR |= R2 ROR 2;          // SR = SR | (R2 ROR 2)
SP += R1 ROR 4, Carry;   // SP = SP + (R1 ROR 4) + C
R2 = R1 ROR 2;           // R2 = R1 ROR 2
```

ASR/ASROR/LSL/LSLOR/LSR/LSROR/ROL/ROR — Shift Operation

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd = Rd SFT_OP Rs	xxxx	Rd	xx	xxx	x	Rs	-	8	R	-	-	-	√

Description: This is a 16-bit multi-cycle shift operation, but it can support 32-bit shift operation by combining 2 shift operations. The result of 32-bit 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).

SFT_OP	Syntax
ASR	Rd = Rd ASR Rs;
ASROR	MR |= Rd ASR Rs;
LSL	Rd = Rd LSL Rs;
LSLOR	MR |= Rd LSL Rs;
LSR	Rd = Rd LSR Rs;
LSROR	MR |= Rd LSR Rs;
ROL	Rd = Rd ROL Rs;
ROR	Rd = Rd ROR Rs;

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

Example:
```
R2 = R2 ASR R1;          // 16-bit arithmetic right shift
R3 = R3 LSR R1;          // 32-bit arithmetic right shift
MR |= R4 ASR R1;
R4 = R4 LSL R1;          // 32-bit logical left
shift MR |= R3 LSL R1;
```

TSTB/SETB/CLRB/INVB — **Bit Operation**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
BIT_OP Rd, Rs	xxxx	Rd	xxx		xx	x Rs	-	4	R R	-	√	-	-
BIT_OP Rd, offset	xxxx	Rd	xxx		xx	offset	-	4		-	√	-	-
BIT_OP {D:}[Rd], Rs	xxxx	Rd	xx	D	xx	x Rs	-	7	[R] R	-	√	-	-
BIT_OP {D:}[Rd], offset	xxxx	Rd	xx	D	xx	offset	-	7		-	√	-	-

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.

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-bit 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: 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	Address Mode	Syntax	Meaning
TSTB	R	TSTB Rd, Rs;	Z= (Rd[Rs[3:0]]== 1)? 1'b0: 1'b1
		TSTB Rd, offset;	Z= (Rd[offset]== 1)? 1'b0: 1'b1
	[R]	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
SETB	R	SETB Rd, Rs;	Rd[Rs[3:0]]= 1
		SETB Rd, offset;	Rd[offset]= 1
	[R]	SETB {D:}[Rd], Rs;	MEM[{DS,Rd}][Rs[3:0]]= 1
		SETB {D:}[Rd], offset;	MEM[{DS,Rd}][offset]= 1
CLRB	R	CLRB Rd, Rs;	Rd[Rs[3:0]]= 0
		CLRB Rd, offset;	Rd[offset]= 0
	[R]	CLRB {D:}[Rd], Rs;	MEM[{DS,Rd}][Rs[3:0]]= 0
		CLRB {D:}[Rd], offset;	MEM[{DS,Rd}][offset]= 0
INVB	R	INVB Rd, Rs;	Rd[Rs[3:0]]= ~Rd[Rs[3:0]]
		INVB Rd, offset;	Rd[offset]= ~Rd[offset]
	[R]	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:

```
INVB R4, R2;        // If R2[3:0]=0x3, R4[3]=~R4[3]
CLRB R3, 10;        // R3[10]=0
SETB [R1], R3;      // If R3[3:0]=0x3, MEM[R1][3]=1
SETB D:[R1], 13;    // MEM[{DS,R1}][13]=1
```

DIVQ	Divide-Quotient Operation

Syntax	Instruction Format	Cycles	Addressing	Flags

	Word Group 1	Word Group 2		Mode	N	Z	S	C
DIVQ MR, R2	xxxxxxxxxxxxxxxx	-	3	R	-	-	-	-

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

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

Example:

```
// 32-bit unsigned dividend / 16-bit unsigned divisor
// 0x0003_1713 / 0x0625
 R4 = 0x0003;

 R3 = 0x1713;          // Load
 data R2 = 0x0625;
 R1 = FR;

 CLRB R1, 14;          // Clear AQ
 flag FR = R1;
 R1 = 1;

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


 R1= 0;

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

DIVS — Divide-Quotient Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

DIVS MR, R2	xxxxxxxxxxxxxxxx	-	2	R	-	-	-	-

Description: DIVS uses the non-restoring division algorithm to compute the sign of the quotient. To implements a division with a 32-bit signed dividend and 16-bit signed divisor, the 32-bit dividend must be placed at MR, 16-bit 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

- **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

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

- **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: // 2-bit signed dividend / 16-bit signed divisor

// 0xFFFF_1713 / 0x0625

```
        R4 = 0xFFFF;
        R3 = 0x1713;        // Load
        data R2 = 0x0625;

        R1 = FR;
        CLRB R1, 14;        // 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, 15;
        JNE div_signed;
```

EXP	Derive Exponent Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
R2 = EXP R4	xxxxxxxxxxxxxxxx	-	2	R	-	-	-	-

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: R2 = EXP R4; // If R4 = 16'b1111_0111_0111_0000, then R2 = 3

// If R4 = 16'b0000_0000_0100_1111, then R2 = 8

4.1.3 Data Segment Access Instruction

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

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
DS = IM6	xxxxxxxxxx	IM6	-	2	IM6	-	-	-	-

Description: DS=0x12;

Example: DS=0x12;

Access Data Segment (DS) value with register	

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
Rs = DS (W=0) DS = Rs (W=1)	xxxxxxxxxxxx	W	Rs	-	2	R	-	-	-	-

Description: Access Data Segment (DS) with register. Only 6-bit 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	

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
Rs = FR (W=0) FR = Rs (W=1)	xxxxxxxxxxxx	W	Rs	-	2	R	-	-	-	-

Description: Access the Flag Register (FR) value.

Example: FR = R1;
R2 = FR;

4.1.4 Transfer-Control Instructions

BREAK	Software Interrupt Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
BREAK	xxxxxxxxxxxxxxxx	-	10	[A16]	-	-	-	-

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; // Generate a software interrupt

CALL — Segmented Far Call

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
CALL A22	xxxxxxxxxx A22[21:16]	A22[15:0]	9	[A22]	-	-	-	-
CALL MR	xxxxxxxxxxxxxxxx	-	8	R	-	-	-	-

Description: For addressing mode [A22], this is a far function call instruction with 22-bit immediate address. Both PC and SR will be pushed to memory indexed by SP and SP, CS will be updated automatically after this operation.

For addressing mode R, this is a far function call instruction with 22-bit indirect address in MR. The 22-bit 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 0x12345;

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

RETF — Return from Subroutine

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
RETF	xxxxxxxxxxxxxxxx	-	8	[A22]	-	-	-	-

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:
```
sub1: .PROC
      … …
      RETF;          // Return from sub1
      .ENDP
```

RETI	Return from Interrupt Service Routine

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
RETI	xxxxxxxxxxxxxxxx	-	8 (IRQ_NEST OFF) / 10 (IRQ_NEST ON)	[A22]	-	-	-	-

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 BREAK, FIQ, IRQ servicing flag inside CPU will be cleared according to priorities.

Example:

```
.TEXT
.PUBLIC _IRQ1
_IRQ1:
    … …
    RETI;                    // Return from IRQ1
```

JUMP	Branch Operation

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
BRANCH_OP IM6	BRANCH_OP	xxxxx	D	IM6	-	2 (not-taken) / 4 (taken)	IM6	-	-	-	-

Description: A conditional short jump instruction to local label (address within$^{\pm}$ **Note:**
D=0 denotes the forward jump, else D=1 denotes the backward jump.

BRANCH_OP	Condition	Description
JCC	C==0	carry clear
JB	C==0	below (unsigned)
JNAE	C==0	not above and equal (unsigned)
JCS	C==1	carry set
JNB	C==1	not below (unsigned)
JAE	C==1	above and equal (unsigned)
JSC	S==0	sign clear

BRANCH_OP	Condition	Description
JGE	S==0	great and equal (signed)
JNL	S==0	not less (signed)
JSS	S==1	sign set
JNGE	S==1	not great than (signed)
JL	S==1	Less (signed)
JNE	Z==0	not equal
JNZ	Z==0	not zero
JZ	Z==1	Zero
JE	Z==1	Equal
JPL	N==0	Plus
JMI	N==1	Minus
JBE	Not (Z==0 and C==1)	below and equal (unsigned)
JNA	Not (Z==0 and C==1)	not above (unsigned)
JNBE	Z==0 and C==1	not below and equal (unsigned)
JA	Z==0 and C==1	above (unsigned)
JLE	Not (Z==0 and S==0)	less and equal (signed)
JNG	Not (Z==0 and S==0)	not great (signed)
JNLE	Z==0 and S==0	not less and equal (signed)
JG	Z==0 and S==0	great (signed)
JVC	N == S	not overflow (signed)
JVS	N != S	overflow (signed)
JMP	Always	unconditional branch

Example: Loop:

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

GOTO **Unconditional Far Jump**

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
GOTO A22	xxxxxxxxxx	A22[21:16]	A22[15:0]	5	[A22]	-	-	-	-
GOTO MR	xxxxxxxxxxxxxxxx		-	4	R	-	-	-	-

Description: For addressing mode [A22], this is a far jump instruction with 22-bit immediate address. The 22-bit target address is range from 0x000000 to 0x3fffff. After this operation, the Code Segment (CS) will be updated automatically. For addressing mode R, this is a far jump instruction with MR register. The 22-bit content of MR {R4[5:0],R3} will be used as destination address.

Example:
```
[example1]
0x008010            GOTO far_func;
   …                ...
0x035678    far_func:
[example2]
GOTO MR;            // Jump to {R4[5:0],R3}
```

4.1.5 Miscellaneous Instructions

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			

FIR_MOV ON/OFF — **Enable/Disable Automatic Data Movement for FIR Operation**

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
FIR_MOV ON FIR_MOV OFF	xxxxxxxxxxxxxxx	FIR	-	2	-	-	-	-	-

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 — **Enable/Disable FIQ**

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
FIQ ON FIQ OFF	xxxxxxxxxxxxxx	FIQ	x	-	2	-	-	-	-	-

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; // Enable FIQ

IRQ ON/OFF	Enable/Disable IRQ

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
IRQ ON IRQ OFF	xxxxxxxxxxxxxx	IRQ	-	2	-	-	-	-	-

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; // Enable IRQ

INT	Interrupt Set

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
INT FIQ INT IRQ INT FIQ, IRQ INT OFF	xxxxxxxxxxxxxx	FIQ	IRQ	-	2	-	-	-	-	-

Description: Set FIQ/IRQ flags.

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)

IRQNEST — IRQ Nest Mode

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
IRQNEST ON IRQNEST OFF	xxxxxxxxxxxxxx	INE	x	-	2	-	-	-	-	-

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;

SECBANK — Switch Register Bank

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
SECBANK ON SECBANK OFF	xxxxxxxxxxxxxxx	BNK	-	2	-	-	-	-	-

Description: Switch secondary register bank mode ON/OFF, 4 shadow registers SR1-SR4 are added in *'nSP* -1.2 and above. User can use this instruction to switch secondary register bank mode on/off. When shadow register mode is on, all operation with R1-R4 will map to SR1-SR4. Secondary Bank Registers are suggested to be used in interrupt service routine only to reduce the effort of saving registers in service routine. 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 — Fraction Mode

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

FRACTION ON FRACTION OFF	xxxxxxxxxxxxxxx	FRA	-	2	-	-	-	-	-

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;

NOP	NO Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
NOP	xxxxxxxxxxxxxxxx	-	2	-	-	-	-	-

Description: No operation, only increase PC to the next address.

Example:
```
Delay_Loop:
    NOP;                    // Waiting
    CMP R1, 0xFFFF;         // Search for end waiting flags
    JA Exit_Loop;           // End waiting
    R1 += 1;                // Waiting for delay
    counting JMP Delay_Loop;
Exit_Loop:
```

4.1.6 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	FI	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	FI	IRQ
IRQNEST ON/OFF	1	1	1	1	-	-	-	1	0	1	0	0	1	1	IN	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	ALU_OP				Rd			1	SFT_OP			n		Rs		
Rd {ALU_OP} = [A6]	ALU_OP				Rd			1	1	1	A6					

Syntax	F		E			B		8	7	6	5	4	3	2	1	0
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_O		0	Rs		
BIT_OP Rd, offset	1	1	1	0		Rd		0	0	1	BIT_O		offset			
BIT_OP {D:}[Rd], Rs	1	1	1	0		Rd		1	0	D	BIT_O		0	Rs		
BIT_OP {D:}[Rd], offset	1	1	1	0		Rd		1	1	D	BIT_O		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
R2 = EXP 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

5 *unSP*-1.3 Instruction Set

5.1 *unSP*-1.3 Instruction Set

unSP 1.3 instruction set has a group of instructions the same as *unSP* 1.2 instruction set. Only the new instructions will be introduced below.

5.1.1 Byte Register Indirect

LOAD	Load Register with Byte Memory

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd = B:[R@]	xxxx	Rd	xxxx	@	x	Rs	-	10/11*	B:[R]	√	√	-	-
Rd = W:[R@]	xxxx	Rd	xxxx	@	x	Rs	-	10/11*	W:[R]				

Rs = R1 / R2 / R3 / R4

Rd = SP / R1 / R2 / R3 / R4 / BP / SR / PC

For load operation, the high byte of Rd is 0.

STORE	Store Register/Immediate into Byte Memory

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
B:[R@] = Rd	xxxx	Rd	xxxx	@	x	Rs	-	10/11*	B:[R]	-	-	-	-
W:[R@] = Rd	xxxx	Rd	xxxx	@	x	Rs	-	10/11*	W:[R]				
B:[R@] = IMM8	xxxxxxxxxxx		@	X	Rs		IMM8	10	B:[R]				
W:[R@] = IMM16	xxxxxxxxxxx		@	X	Rs		IMM16	10	W:[R]				

Rs = R1 / R2 / R3 / R4

Rd = SP / R1 / R2 / R3 / R4 / BP / SR / PC

For store operation, only the low byte of Rd is stored to B:[Rs@]

*: Write to PC

5.1.2 Byte Indexed Address

LOAD	Load Register with Byte Memory

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
Rd = B:[BP+IM6]	xxxxx	Rd	xxx	IM6	-	6/7*	B:[BP+IM6]	√	√	-	-
Rd = W:[BP+IM6]	xxxxx	Rd	xxx	IM6	-	6/7*	W:[BP+IM6]				

Rd = R1 / R2 / R3 / R4

For load operation, the high byte of Rd is 0.

STORE	Store Register/Immediate into Byte Memory

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
B:[BP+IM6] = Rd	xxxxx	Rd	xxx	IM6	-	6/7*	B:[BP+IM6]	-	-	-	-
W:[BP+IM6] = Rd	xxxxx	Rd	xxx	IM6	-	6/7*	W:[BP+IM6]				
B:[BP+IM6] = IMM8	xxxxxxxxxx			IM6	IMM8	10	B:[BP+IM6]				
W:[BP+IM6] = IMM16	xxxxxxxxxx			IM6	IMM16-	10	W:[BP+IM6]				

Rd = R1 / R2 / R3 / R4

For store operation, only the low byte of Rd is stored to B:[BP+IM6]

*: Write to PC

5.1.3 Byte Register Indexed Address

LOAD	Load Register with Byte Memory

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd = B:I[R@]	xxxx	Rd	xxxx	@	x	Rs	-	9/10*	B:I[R]	√	√	-	-
Rd = W:I[R@]	xxxx	Rd	xxxx	@	x	Rs	-	9/10*	W:I[R]				

Rs = R1 / R2 / R3 / R4

Rd = SP / R1 / R2 / R3 / R4 / BP / SR / PC

For load operation, the high byte of Rd is 0.

STORE	Store Register/Immediate into Byte Memory

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
B:I[R@] = Rd	xxxx	Rd	xxxx	@	x	Rs	-	9/10*	B:I[R]	-	-	-	-
W:I[R@] = Rd	xxxx	Rd	xxxx	@	x	Rs	-	9/10*	W:I[R]				
B:I[R@] = IMM8	xxxxxxxxxxx			@	x	Rs	IMM8	9	B:I[R]				
W:I[R@] = IMM16	xxxxxxxxxxx			@	x	Rs	IMM16	9	W:I[R]				

Rs = R1 / R2 / R3 / R4

Rd = SP / R1 / R2 / R3 / R4 / BP / SR / PC

For store operation, only the low byte of Rd is stored to B:I[Rs@] *:

Write to PC

5.1.4 Special Register Access

Stack Segment Register Access	

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
Rs = SS (W=0) SS = Rs (W=1)	xxxxxxxxxxxx	W	Rs	-	2	R	-	-	-	-

Description: Access the Stack Segment Register (SS) value.

Inner Product Operation Data Segment Access	

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
MDS = R3 R3 = MDS	xxxxxxxxxxxxxxxx	-	2	R	-	-	-	-

Description: Access the Inner Product Data Segment Register (MDS) value with R3.

TSTB/SETB/CLRB/INVB — **Bit operations with direct memory addressing**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
BIT_OP {D:}[A16], offset	xxxxx	DS	xxxx	BIT_OP	offset	A16	8	A16		√	-	-

Description: Executing bit operation with the value at memory location indexed by 16 bits operand. Users can use the "D:" indicator to access memory space large than 64K words, If "D:" indicator is used, the MSB 6-bit 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.

BIT_OP	Syntax	Meaning
TSTB	TSTB {D:}[A16], offset;	Z= (MEM[{DS,A16}][offset]== 1)? 1'b0: 1'b1
SETB	SETB {D:}[A16], offset;	MEM[{DS,A16}][offset]= 1
CLRB	CLRB {D:}[A16], offset;	MEM[{DS,A16}][offset]]= 0
INVB	INVB {D:}[A16], offset;	MEM[{DS,A16}][offset]= ~ MEM[{DS,A16}]offset]

Example:
```
SETB [0x5678], 5;        // MEM[0x5678][5]= 1
SETB D:[0x1234], 13;     // If DS=3, MEM[{0x31234}][13]= 1
```

DIVISION — **Single Instruction Division Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
DIVUU MR, R2	xxxxxxxxxxxxxxxx	-	48	R	-	-	-	-
DIVSS MR, R2	xxxxxxxxxxxxxxxx		47	R				

Description: ISA 1.2 used the non-resorting division algorithm to yield a 1-bit quotient at each instruction. 16 instructions at least are needed to implement a division with a 32-bits dividend and 16-bits division. Therefore, two new division instructions, DIVUU and DIVSS, are introduced to help reducing code size in ISA-1.3. DIVUU performs dividing unsigned 32-bit dividend (MR) by unsigned

16-bit divisor (R2). DIVSS performs dividing signed 32-bit dividend (MR) by signed 16-bit divisor (R2)

Example:

```
// 32-bits unsigned dividend / 16-bits unsigned divisor
// 0x0003_1713 / 0x0625
R4 = 0x0003;
R3 = 0x1713;          // Load data
R2 = 0x0625;

R1 = FR;
CLRB R1, 14;          // Clear AQ
flag FR = R1;

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

DIVUU MR, R2;         // Implement an unsigned division
```

Example:

```
// 32-bits signed dividend / 16-bits signed divisor
// 0xFFFF_1713 / 0x0625
R4= 0xFFFF;
R3= 0x1713;           // Load data
R2= 0x0625;

R1= FR;
CLRB R1, 14;          // Clear AQ
flag FR= R1;

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

DIVSS MR, R2;         // Implement an signed division
```

6 *unSP*-2.0 Instruction Set

6.1 *unSP*-2.0 Instruction Cycles

unSP 2.0 instruction set has a group of instructions the same as *unSP* 1.2 instruction set which are introduced at Chapter 4. So, while introducing the same instructions of *unSP* 2.0 instruction set, instruction format, cycles, and affected flags are mainly described.

6.1.1 Data-Transfer Instructions

LOAD — **Load Register with Memory/Immediate/Register**

Syntax	Instruction Format										Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1													N	Z	S	C
Rd = IM6	xxxx	Rd	xxx	IM6							-	1	IM6	√	√	-	-
Rd = IM16	xxxx	Rd	xxxxxx							Rs	IM16	2	IM16				
Rd = [BP+IM6]	xxxx	Rd	xxx	IM6							-	1 / 2	[BP+IM6]				
Rd = [A6]	xxxx	Rd	xxx	A6							-	1	[A6]				
Rd = [A16]	xxxx	Rd	xxxxx					W	Rs		A16	2	[A16]				
Rd = Rs	xxxx	Rd	xxxxxx						Rs		-	1	R				
Rd = {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs					-	2 / 3	[R]				

STORE — **Store Register into Memory**

Syntax	Instruction Format										Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1													N	Z	S	C
[BP+IM6] = Rd	xxxx	Rd	xxx	IM6							-	1 / 2	[BP+IM6]	-	-	-	-
[A6] = Rd	xxxx	Rd	xxx	A6							-	1	[A6]				
[A16] = Rd	xxxx	Rd	xxxxx					W	Rs		A16	2	[A16]				
{D:}[Rs@] = Rd	xxxx	Rd	xxx	D	@	Rs					-	2 / 3	[R]				

Note: For addressing mode [BP+IM6], STORE operation only need 1 cycle.

PUSH — **Push Registers onto Stack**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

PUSH Rx, Ry to [Rs] Or PUSH Rx to [Rs]	xxxx	Rd	xxx	n	Rs	-	n+1	[R]	-	-	-	-

POP — **Pop Registers from Stack**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
POP Rx, Ry from [Rs] Or POP Rx from [Rs]	xxxx	Rd	xxx	n	Rs	-	n+2	[R]	-	-	-	-

6.1.2 Data Processing Instructions

ADD — **Add without Carry**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += IM6 Rd = Rd + IM6	xxxx	Rd	xxx	IM6		-	1	IM6	√	√	√	√
Rd = Rs + IM16	xxxx	Rd	xxxxxx		Rs	IM16	2	IM16				
Rd += [BP+IM6] Rd = Rd + [BP+IM6]	xxxx	Rd	xxx	IM6		-	1 / 2	[BP+IM6]				
Rd += [A6] Rd = Rd + [A6]	xxxx	Rd	xxx	A6		-	1	[A6]				
Rd = Rs + [A16] [A16] = Rd + Rs	xxxx	Rd	xxxxx	W	Rs	A16	2	[A16]				
Rd += Rs	xxxx	Rd	xxxxxx		Rs	-	1	R				
Rd += {D:}[Rs@]	xxxx	Rd	xxx	D	@ Rs	-	2 / 3	[R]				

ADC — **Add with Carry**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += IM6, Carry Rd = Rd + IM6, Carry	xxxx	Rd	xxx	IM6			-	1	IM6	√	√	√	√
Rd = Rs + IM16, Carry	xxxx	Rd	xxxxxx		Rs		IM16	2	IM16				
Rd += [BP+IM6], Carry Rd = Rd + [BP+IM6], Carry	xxxx	Rd	xxx	IM6			-	1 / 2	[BP+IM6]				
Rd += [A6], Carry Rd = Rd + [A6], Carry	xxxx	Rd	xxx	A6			-	1	[A6]				
Rd = Rs + [A16], Carry [A16] = Rd + Rd, Carry	xxxx	Rd	xxxxx		W	Rs	A16	2	[A16]				
Rd += Rs, Carry	xxxx	Rd	xxxxxx			Rs	-	1	R				
Rd += {D:}[Rs@], Carry	xxxx	Rd	xxx	D	@	Rs	-	2 / 3	[R]				

SUB — Subtract without Carry

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd -= IM6 Rd = Rd – IM6	xxxx	Rd	xxx	IM6			-	1	IM6	√	√	√	√
Rd = Rs – IM16	xxxx	Rd	xxxxxx		Rs		IM16	2	IM16				
Rd -= [BP+IM6] Rd = Rd - [BP+IM6]	xxxx	Rd	xxx	IM6			-	1 / 2	[BP+IM6]				
Rd -= [A6] Rd = Rd - [A6]	xxxx	Rd	xxx	A6			-	1	[A6]				
Rd = Rs - [A16] [A16] = Rd - Rs	xxxx	Rd	xxxxx		W	Rs	A16	2	[A16]				
Rd -= Rs	xxxx	Rd	xxxxxx			Rs	-	1	R				
Rd -= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	2 / 3	[R]				

SBC — Subtract with Carry

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

Rd -= IM6, Carry Rd = Rd - IM6, Carry	xxxx	Rd	xxx	IM6		-	1	IM6			
Rd = Rs - IM16, Carry	xxxx	Rd	xxxxxx		Rs	IM16	2	IM16			
Rd -= [BP+IM6], Carry Rd = Rd - [BP+IM6], Carry	xxxx	Rd	xxx	IM6		-	1 / 2	[BP+IM6]	√	√	√	√
Rd -= [A6], Carry Rd = Rd - [A6], Carry	xxxx	Rd	xxx	A6		-	1	[A6]			
Rd = Rs - [A16], Carry [A16] = Rd – Rs, Carry	xxxx	Rd	xxxxxx		W Rs	A16	2	[A16]			
Rd -= Rs, Carry	xxxx	Rd	xxxxxx		Rs	-	1	R			
Rd -= {D:}[Rs@], Carry	xxxx	Rd	xxx	D	@ Rs	-	2 / 3	[R]			

NEG — Negative

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd = -IM6	xxxx	Rd	xxx	IM6		-	1	IM6				
Rd = -IM16	xxxx	Rd	xxxxxx		Rs	IM16	2	IM16				
Rd = -[BP+IM6]	xxxx	Rd	xxx	IM6		-	1 / 2	[BP+IM6]				
Rd = -[A6]	xxxx	Rd	xxx	A6		-	1	[A6]	√	√	-	-
Rd = -[A16] [A16] = -Rd	xxxx	Rd	xxxxx		W Rs	A16	2	[A16]				
Rd = -Rs	xxxx	Rd	xxxxxx		Rs	-	1	R				
Rd = -{D:}[Rs@]	xxxx	Rd	xxx	D	@ Rs	-	2 / 3	[R]				

CMP — Compare

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
CMP Rd, IM6	xxxx	Rd	xxx	IM6		-	1	IM6	√	√	√	√
CMP Rs, IM16	xxxx	Rd	xxxxxx		Rs	IM16	2	IM16				
CMP Rd, [BP+IM6]	xxxx	Rd	xxx	IM6		-	1 / 2	[BP+IM6]				
CMP Rd, [A6]	xxxx	Rd	xxx	A6		-	1	[A6]				
CMP Rs, [A16] CMP Rd, Rs	xxxx	Rd	xxxxx		W Rs	A16	2	[A16]				

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1							Word Group 2			N	Z	S	C
CMP Rd, Rs	xxxx	Rd	xxxxxx			Rs		-	1	R				
CMP Rd, {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs		-	2 / 3	[R]				

MUL — Register Multiplication

Syntax	Instruction Format						Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1									N	Z	S	C
MR = Rd*Rs {,ss/,us/,uu}	xxx	Ss	Rd	Sd	xxxxx	Rs	-	1	R	-	-	-	-

MULS — Inner Product Operation

Syntax	Instruction Format							Word Group 2	Cycles	Addressing Mode	Flags			
	Word Group 1										N	Z	S	C
MR = [Rd] * [Rs] {,ss/,us/,uu}, N	xxx	Ss	Rd	Sd	x	N	Rs	-	*	[R]	-	-	-	-

*Cycles:

N+2 (DM/IM no conflict, FIR_MOVE Off)

2N+1 (DM/IM no conflict, FIR_MOVE On)

2N+2 (DM/IM conflict, FIR_MOVE Off)

3N (DM/IM conflict, FIR_MOVE On)

Description: This operation is using a 36-bit 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-bit 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.

In *unSP*2.0, multiplication data will be fetched from INST Bus and DATA Bus concurrently to accelerating MAC operation, if the parameter array location indexed by Rd, Rs place at different memory range (IM/DM), MULS will have the best performance or bus conflict stall may be occurred and need 2 times of executing cycles.

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

AND — Logical AND

Syntax	Instruction Format: Word Group 1						Instruction Format: Word Group 2	Cycles	Addressing Mode	Flags: N	Z	S	C
Rd &= IM6 Rd = Rd & IM6	xxxx	Rd	xxx	IM6			-	1	IM6	√	√	-	-
Rd = Rs & IM16	xxxx	Rd	xxxxxx			Rs	IM16	2	IM16				
Rd &= [BP+IM6] Rd = Rd & [BP+IM6]	xxxx	Rd	xxx	IM6			-	1 / 2	[BP+IM6]				
Rd &= [A6] Rd = Rd & [A6]	xxxx	Rd	xxx	A6			-	1	[A6]				
Rd = Rs & [A16] [A16] = Rd & Rs	xxxx	Rd	Xxxxx		W	Rs	A16	2	[A16]				
Rd &= Rs	xxxx	Rd	xxxxxx			Rs	-	1	R				
Rd &= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	2 / 3	[R]				

OR — Logical Inclusive OR

Syntax	Instruction Format: Word Group 1						Instruction Format: Word Group 2	Cycles	Addressing Mode	Flags: N	Z	S	C
Rd |= IM6 Rd = Rd | IM6	xxxx	Rd	xxx	IM6			-	1	IM6	√	√	-	-

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd = Rs | IM16	xxxx	Rd	xxxxxx		Rs		IM16	2	IM16				
Rd |= [BP+IM6] Rd = Rd | [BP+IM6]	xxxx	Rd	xxx	IM6			-	1 / 2	[BP+IM6]				
Rd |= [A6] Rd = Rd | [A6]	xxxx	Rd	xxx	A6			-	1	[A6]				
Rd = Rs | [A16] [A16] = Rd | Rs	xxxx	Rd	Xxxxx	W	Rs		A16	2	[A16]				
Rd |= Rs	xxxx	Rd	xxxxxx		Rs		-	1	R				
Rd |= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	2 / 3	[R]				

XOR — Logical Exclusive OR

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd ^= IM6 Rd = Rd ^ IM6	xxxx	Rd	xxx	IM6			-	1	IM6	√	√	-	--
Rd = Rs ^ IM16	xxxx	Rd	xxxxxx		Rs		IM16	2	IM16				
Rd ^= [BP+IM6] Rd = Rd ^ [BP+IM6]	xxxx	Rd	xxx	IM6			-	1 / 2	[BP+IM6]				
Rd ^= [A6] Rd = Rd ^ [A6]	xxxx	Rd	xxx	A6			-	1	[A6]				
Rd = Rs ^ [A16] [A16] = Rd ^ Rs	xxxx	Rd	xxxxx	W	Rs		A16	2	[A16]				
Rd ^= Rs	xxxx	Rd	xxxxxx		Rs		-	1	R				
Rd ^= {D:}[Rs@]	xxxx	Rd	xxx	D	@	Rs	-	2 / 3	[R]				

TEST — Logical Test

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
TEST Rd, IM6	xxxx	Rd	xxx	IM6			-	1	IM6	√	√	-	-
TEST Rs, IM16	xxxx	Rd	xxxxxx		Rs		IM16	2	IM16				
TEST Rd, [BP+IM6]	xxxx	Rd	xxx	IM6			-	1 / 2	[BP+IM6]				
TEST Rd, [A6]	xxxx	Rd	xxx	A6			-	1	[A6]				

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
TEST Rs, [A16] TEST Rd, Rs	xxxx	Rd	xxxxx	W	Rs	A16	2	[A16]				
TEST Rd, Rs	xxxx	Rd	xxxxxx		Rs	-	1	R				
TEST Rd, {D:}[Rs@]	xxxx	Rd	xxx	D	@ Rs	-	2 / 3	[R]				

ASR-ALU Register Arithmetic-Shift-Right and Arithmetic/Logical Operation

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
Rd += Rs ASR nn {,Carry} Rd -= Rs ASR nn {,Carry} CMP Rd, Rs ASR nn	xxxx	Rd	x	xxx	nn	Rs -	1	R	√	√	√	√
Rd = - Rs ASR nn Rd &= Rs ASR nn Rd |= Rs ASR nn Rd ^= Rs ASR nn TEST Rd, Rs ASR nn Rd = Rs ASR nn	xxxx	Rd	x	xxx	nn	Rs -	1	R	√	√	-	-

Note: FIQ, IRQ and user routine has their own Shift buffers. User does not need to save shift buffer for interrupt routines.

Shift buffer values are unknown after multiplication or filter operations. User should make no assumptions to its value after the operations.

LSL-ALU Register Logical-Shift-Left and Arithmetic/Logical Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

Rd += Rs LSL nn {,Carry} Rd -= Rs LSL nn {,Carry} CMP Rd, Rs LSL nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	√	√
Rd = - Rs LSL nn Rd &= Rs LSL nn Rd |= Rs LSL nn Rd ^= Rs LSL nn TEST Rd, Rs LSL nn Rd = Rs LSL nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	-	-

LSR-ALU **Register Logical-Shift-Right and Arithmetic/Logical Operation**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += Rs LSR nn {,Carry} Rd -= Rs LSR nn {,Carry} CMP Rd, Rs LSR nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	√	√
Rd = - Rs LSR nn Rd &= Rs LSR nn Rd |= Rs LSR nn Rd ^= Rs LSR nn TEST Rd, Rs LSR nn Rd = Rs LSR nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	-	-

ROL-ALU **Register Rotate-Left and Arithmetic/Logical Operation**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += Rs ROL nn {,Carry} Rd -= Rs ROL nn {,Carry} CMP Rd, Rs ROL nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	√	√

Rd = - Rs ROL nn Rd &= Rs ROL nn Rd |= Rs ROL nn Rd ^= Rs ROL nn TEST Rd, Rs ROL nn Rd = Rs ROL nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	-	-

ROR-ALU — **Register Rotate-Right and Arithmetic/Logical Operation**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd += Rs ROR nn {,Carry} Rd -= Rs ROR nn {,Carry} CMP Rd, Rs ROR nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	√	√
Rd = - Rs ROR nn Rd &= Rs ROR nn Rd |= Rs ROR nn Rd ^= Rs ROR nn TEST Rd, Rs ROR nn Rd = Rs ROR nn	xxxx	Rd	x	xxx	nn	Rs	-	1	R	√	√	-	-

ASR/ASROR/LSL/LSLOR/LSR/LSROR/ROL/ROR — **Shift Operation**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2			N	Z	S	C
Rd= Rd SFT_OP Rs	xxxx	Rd	xx	xxx	x	Rs	-	1 / 2	R	-	-	-	√

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

TSTB/SETB/CLRB/INVB — **Bit Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

BIT_OP Rd, Rs	xxxx	Rd	xxx		xx	x	Rs	-	1	R	-	√	-	-
BIT_OP Rd, offset	xxxx	Rd	xxx		xx	offset		-	1	R	-	√	-	-
BIT_OP {D:}[Rd], Rs	xxxx	Rd	xx	D	xx	x	Rs	-	1	[R]	-	√	-	-
BIT_OP {D:}[Rd], offset	xxxx	Rd	xx	D	xx	offset		-	1	R	-	√	-	-

DIVQ — **Divide-Quotient Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
DIVQ MR, R2	xxxxxxxxxxxxxxxx	-	1	R	-	-	-	-

DIVS — **Divide-Quotient Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
DIVS MR, R2	xxxxxxxxxxxxxxxx	-	1	R	-	-	-	-

EXP — **Divide Exponent Operation**

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
R2 = EXP R4	xxxxxxxxxxxxxxxx	-	1	R	-	-	-	-

6.1.3 Data Segment Access Instruction

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

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
DS = IM6	xxxxxxxxxx	IM6	-	1	IM6	-	-	-	-

Access Data Segment(DS)value with register

Syntax	Instruction Format	Cycles	Addressing	Flags

	Word Group 1			Word Group 2		Mode	N	Z	S	C
Rs = DS (W=0) DS = Rs (W=1)	xxxxxxxxxxxx	W	Rs	-	1	R	-	-	-	-

Flag Register Access	

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
Rs = FR (W=0) FR = Rs (W=1)	xxxxxxxxxxxx	W	Rs	-	1	R	-	-	-	-

6.1.4 Transfer-Control Instructions

BREAK	Software Interrupt Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
BREAK	xxxxxxxxxxxxxxxx	-	4	[A16]	-	-	-	-

CALL	Segmented Far Call

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
CALL A22	xxxxxxxxxx	A22[21:16]	A22[15:0]	3	[A22]	-	-	-	-
CALL MR	xxxxxxxxxxxxxxxx		-	4	R	-	-	-	-

RETF	Return from Subroutine

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
RETF	xxxxxxxxxxxxxxxx	-	6	[A22]	-	-	-	-

RETI	Return from Interrupt Service Routine

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

RETI	xxxxxxxxxxxxxxxx	-	6 (IRQ NEST OFF) / 7 (IRQ NEST ON)	[A22]	-	-	-	-

JUMP	Branch Operation

Syntax	Instruction Format					Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2			N	Z	S	C
BRANCH_OP IM6	BRANCH_OP	xxxxx	D	IM6	-	1 (not-taken) / 4 (taken)	IM6	-	-	-	-

Description: A conditional short jump instruction to local label (address within ±

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

BRANCH_OP	Condition	Description
JCC	C==0	carry clear
JB	C==0	below (unsigned)
JNAE	C==0	not above and equal (unsigned)
JCS	C==1	carry set
JNB	C==1	not below (unsigned)
JAE	C==1	above and equal (unsigned)
JSC	S==0	sign clear
JGE	S==0	great and equal (signed)
JNL	S==0	not less (signed)
JSS	S==1	sign set
JNGE	S==1	not great than (signed)
JL	S==1	Less (signed)
JNE	Z==0	not equal
JNZ	Z==0	not zero
JZ	Z==1	Zero
JE	Z==1	Equal
JPL	N==0	Plus
JMI	N==1	Minus
JBE	Not (Z==0 and C==1)	below and equal (unsigned)
JNA	Not (Z==0 and C==1)	not above (unsigned)

BRANCH_OP	Condition	Description
JNBE	Z==0 and C==1	not below and equal (unsigned)
JA	Z==0 and C==1	above (unsigned)
JLE	Not (Z==0 and S==0)	less and equal (signed)
JNG	Not (Z==0 and S==0)	not great (signed)
JNLE	Z==0 and S==0	not less and equal (signed)
JG	Z==0 and S==0	great (signed)
JVC	N == S	not overflow (signed)
JVS	N != S	overflow (signed)
JMP	Always	unconditional branch

GOTO — **Unconditional Far Jump**

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
GOTO A22	xxxxxxxxxx	A22[21:16]	A22[15:0]	3	[A22]	-	-	-	-
GOTO MR	xxxxxxxxxxxxxxxx		-	2	R	-	-	-	-

6.1.5 Miscellaneous Instructions

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			

FIR_MOV ON/OFF — **Enable/Disable Automatic Data Movement for FIR Operation**

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
FIR_MOV ON FIR_MOV OFF	xxxxxxxxxxxxxxx	FIR	-	1	-	-	-	-	-

FIR_MOV ON/OFF — **Enable/Disable FIR**

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
FIQ ON FIQ OFF	xxxxxxxxxxxxxx	FIQ	x	-	1	-	-	-	-	-

IRQ ON/OFF — Enable/Disable IRQ

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
IRQ ON IRQ OFF	xxxxxxxxxxxxxx	IRQ	-	1	-	-	-	-	-

INT — Interrupt Set

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
INT FIQ INT IRQ INT FIQ, IRQ INT OFF	xxxxxxxxxxxxxx	FIQ	IRQ	-	1	-	-	-	-	-

IRQNEST — IRQ Nest Mode

Syntax	Instruction Format				Cycles	Addressing Mode	Flags			
	Word Group 1			Word Group 2			N	Z	S	C
IRQNEST ON IRQNEST OFF	xxxxxxxxxxxxxx	INE	x	-	1	-	-	-	-	-

SECBANK — Switch Register Bank

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1		Word Group 2			N	Z	S	C
SECBANK ON SECBANK OFF	xxxxxxxxxxxxxx	BNK	-	1	-	-	-	-	-

FRACTION — Fraction Mode

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C

FRACTION ON FRACTION OFF	xxxxxxxxxxxxxxx	FRA	-	1	-	-	-	-	-

NOP	No Operation

Syntax	Instruction Format		Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2			N	Z	S	C
NOP	xxxxxxxxxxxxxxxx	-	1	-	-	-	-	-

Description: No operation, only increase PC to the next address.

6.2 New Instructions of *unSP*-2.0 Instruction Set

unSP 2.0 has some new kind of ALU instructions that operation with extend registers.

For this kind of instructions, the operation code contains three parts that are extend code, word group 1, and word group 2, and extend code, the operation code format is extend code + word group 1 + word group 2.

Note:

- Ra, Rb: R0~R15

  Rx, Ry: R8~R15

- For 16-bit direct memory addressing, there are two kinds of instruction format:

  Ra = Rb # [A16]; (W=0)

  [A16] = Ra # Rb; (W=1)

- On the Cycles column, the number after '/' denotes writing to PC.

  For addressing mode [R], the number after '/' denotes using [++Ry] prefix.

~ The prefix of source register

Table 6.1

Rs@	Meaning
Ry	No increment/decrement
Ry--	After ALU_OP, Ry = Ry-1
Ry++	After ALU_OP, Ry = Ry +1
++Ry	Before ALU_OP, Ry = Ry +1

6.2.1 Data-Transfer Instructions

LOAD	Load Extended Register with Memory/Immediate/Register

Syntax	Instruction Format	Cycle	Addressing	Flags

	Word Group 1								Word Group 2	Ext Code			N	Z	S	C
Rx = IM6	xxxx	Rx	xxx	IM6					-	16b	2	IM6				
Ra = IM16	xxxx	Ra2-0	Ra3	xxxx			Rb		IM16	16b	3	IM16				
Rx = [BP+IM6]	xxxx	Rx	xxx	IM6					-	16b	2 / 3	[BP+IM6]				
Rx = [A6]	xxxx	Rx	xxx	A6					-	16b	2	[A6]	√	√	√	√
Ra = [A16]	xxxx	Ra2-0	Ra3	xxx	W		Rb		A16	16b	3	[A16]				
Ra = Rb	xxxx	Ra2-0	Ra3	xxx			Rb		-	16b	2	R				
Rx = {D:}[Ry@]	xxxx	Rx	xxx	D	@		Ry		-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for reading of data transmitting, i.e. Rd=X. X shows different form according to addressing mode.

Example:

```
R10 = 0x28;                 // IM6
R12 = 0x2400;               // IM16
R13 = [BP+0x08];            // [BP+IM6]
R14 = [0x30]; R14 =         // [A6]
[0x2480]; [0x2480] =        // [A16]
R12; SR = R12;              // [A16], R12 is assigned to MEM[0x2480] // R
PC += D:[BP++];             // [R]
```

STORE **Store Extended Register into Memory**

Syntax	Instruction Format								Word Group 2	Ext Code	Cycles	Addressing Mode	Flags			
	Word Group 1												N	Z	S	C
[BP+IM6] = Rx	xxxx	Rx	xxx	IM6					-	16b	2 / 3	[BP+IM6]	-	-	-	-
[A6] = Rx	xxxx	Rx	xxx	A6					-	16b	2	[A6]				
[A16] = Ra	xxxx	Ra_{2-0}	Ra_3	xxx	W		Rb		A16	16b	3	[A16]				
{D:}[Ry@] = Rx	xxxx	Rx	xxx	D	@		Ry		-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for writing of data transmitting, i.e. X=Rd. X shows different form according to addressing mode. **Note:** For addressing mode [BP+IM6], STORE operation only need 2 cycle.

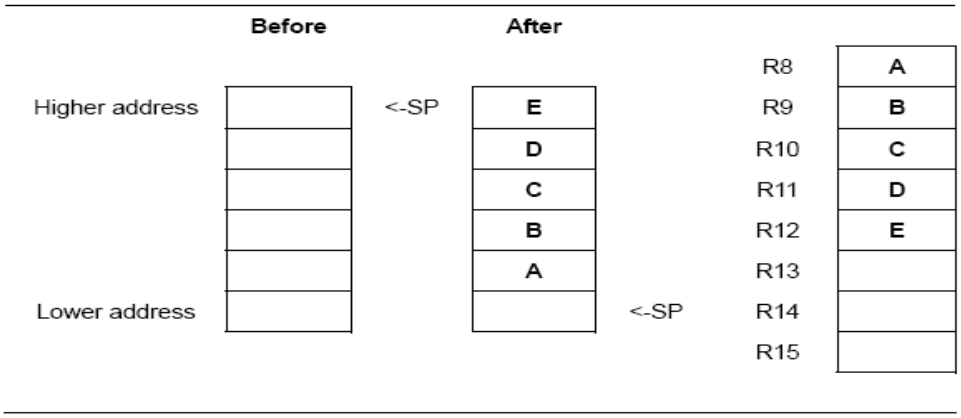
Example:
```
[BP+0x08] = R11;          // [BP+IM6]
[0x30] = R12;             // [A6]
[0x2480] = R13;           // [A16]
D:[R12++] = R14;          // [R]
```

PUSH — **Push Extended Registers onto Stack**

Syntax	Instruction Format							Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2	Ext Code			N	Z	S	C
PUSH R_H, R_L to [Rb]	x	N	Rx	xxxxx	Rb	-	16b	N+2	-	-	-	-	-

Description: Push a set of registers (R_H ~ R_L) to memory location indicated by Rb consecutively. **Note:** PUSH R9, R15 to [SP] is equivalent to PUSH R15, R9 to [SP].

Example: PUSH R8, R12 to [SP]; // Push R8 through R12, and N=5



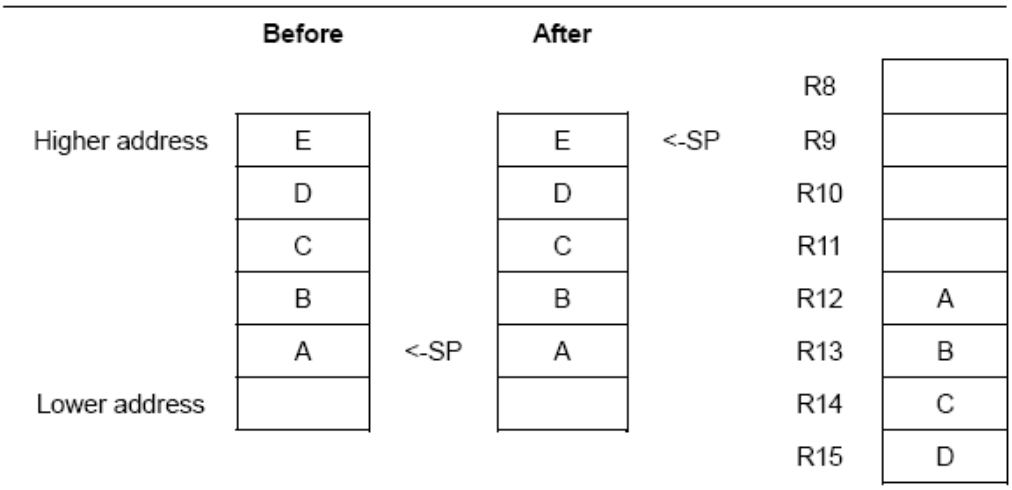
POP — **Pop Extended Registers from Stack**

Syntax	Instruction Format			Cycles	Addressing Mode	Flags			
	Word Group 1	Word Group 2	Ext Code			N	Z	S	C

POP R_L, R_H from [Rb]	x	N	Rx	xxxxx	Rb	-	16b	N+2	-	-	-	-	-

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

Example: POP R12, R15 from [SP]; // Pop R12 through R15, and N=4



6.2.2 Data Processing Instructions

Data Processing Instructions include ALU Operation, Bit Operation, Shift Operation, Mul Operation, Div Operation, EXP Operation, NOP, etc..

ALU Operation Instructions that carry out the operation as RD = X # Y. X and Y will show different meanings according to the addressing mode. Because the same explanation for X, Y and the description for Rs, Rd will be involved in instruction they will be listed in Table 6.2.

Table 6.2 The meanings for X, Y in operation as Rd = X # Y

Addressing Mode	X, Y
IM6	X is Rd, Y is IM6. IM6 will be expanded to 16-bit filled with zeros first, and then be operated with X.
IM16	X is Rs, Y is IM16
[BP+IM6]	X is Rd, Y is the memory in PAGE0 addressed as (BP+IM6)
[A6]	X is Rd, Y is the memory in PAGE0 addressed as (0x00~0x3F)
[A16]	X is Rs. Y is the memory in PAGE0 addressed as (0x0000~0xFFFF)
R	X is Rd, Y is Rs.
{D:}[R] {D:}[R--] {D:}[R++] {D:}[++R]	X is Rd, Y is the memory address pointed by the offset in Rs. Rs may point data segment in PAGE0 as 'D' is ignored or in non-PAGE0 as 'D' is not ignored and its page index depends on DS in SR register. Rs can be increased by 1 before ALU operation or increased/decreased by 1 after ALU operation.

ADD — Add without Carry

Syntax	Instruction Format: Word Group 1							Word Group 2	Ext Code	Cycles	Addressing Mode	Flags: N	Z	S	C
Rx += IM6 Rx = Rx + IM6	xxxx	Rx	xxx	IM6				-	16b	2	IM6	√	√	√	√
Ra = Rb + IM16	xxxx	Ra_{2-0}	Ra_3	xxxx		Rb		IM16	16b	3	IM16				
Rx += [BP+IM6] Rx = Rx + [BP+IM6]	xxxx	Rx	xxx	IM6					16b	2 / 3	[BP+IM6]				
Rx += [A6] Rx = Rx + [A6]	xxxx	Rx	xxx	A6				-	16b	2	[A6]				
Ra = Rb + [A16] [A16] = Ra + Rb	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb		A16	16b	3	[A16]				
Ra += Rb	xxxx	Ra_{2-0}	Ra_3	xxx		Rb		-	16b	2	R				
Rx += {D:}[Ry@]	xxxx	Rx	xxx	D	@	Ry		-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for addition operation without carry, i.e. Rd = X+Y. X, Y will have different meanings according to the addressing mode. See Table 6.2.

Example:

```
R10 += 0x28;                // IM6
R12 = R11 + 0x2400;         // IM 16
R13 += [BP+0x08];           // [BP+IM6]
R14 += [0x30];              // [A6]
BP = R14 + [0x2480];        // [A16]
[0x2480] = BP + R12;        // [A16], BP + R12 is assigned to MEM[0x2480]
SR += R12;                  // R
PC += D:[BP++];             // [R]
```

ADC — Add with Carry

Syntax	Instruction Format: Word Group 1	Word Group 2	Ext Code	Cycles	Addressing Mode	Flags: N	Z	S	C

Syntax	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2	Ext Code			N	Z	S	C
Rx += IM6, Carry Rx = Rx + IM6, Carry	xxxx	Rx	xxx	IM6			-	16b	2	IM6	√	√	√	√
Ra = Rb + IM16, Carry	xxxx	Ra_{2-0}	Ra_3	xxxx		Rb	IM16	16b	3	IM16				
Rx += [BP+IM6] , Carry Rx = Rx + [BP+IM6] , Carry	xxxx	Rx	xxx	IM6			-	16b	2 / 3	[BP+IM6]				
Rx += [A6] , Carry Rx = Rx + [A6] , Carry	xxxx	Rx	xxx	A6			-	16b	2	[A6]				
Ra = Rb + [A16] , Carry [A16] = Ra + Rb, Carry	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb	A16	16b	3	[A16]				
Ra += Rb, Carry	xxxx	Ra_{2-0}	Ra_3	xxx		Rb	-	16b	2	R				
Rx += {D:}[Ry@], Carry	xxxx	Rx	xxx	D	@	Ry	-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for addition with carry in arithmetical operation, i.e. Rd = X+Y+C. X, Y will have different meanings according to the addressing mode. See Table 6.2.

Example:

```
R10 += 0x28, Carry;                          // IM6
R12 = R11 + 0x2400, Carry;                   // IM16
R13 += [BP+0x08] , Carry;                    // [BP+IM6]
R14 += [0x30] , Carry;                       // [A6]
BP = R14 + [0x2480] , Carry;                 // [A16]
[0x2480] = BP + R12, Carry; SR +=            // [A16], BP + R12 + C is assigned to MEM[0x2480]
R12, Carry;                                  // R
PC += D:[BP++], Carry;                       // [R]
```

SUB — Subtract without Carry

	Instruction Format								Cycles	Addressing Mode	Flags			
	Word Group 1						Word Group 2	Ext Code			N	Z	S	C
Rx -= IM6 Rx = Rx - IM6	xxxx	Rx	xxx	IM6			-	16b	2	IM6	√	√	√	√
Ra = Rb - IM16	xxxx	Ra_{2-0}	Ra_3	Xxxx		Rb	IM16	16b	3	IM16				

Rx -= [BP+IM6] Rx = Rx - [BP+IM6]	xxxx	Rx	xxx	IM6		16b	2 / 3	[BP+IM6]	
Rx -= [A6] Rx = Rx - [A6]	xxxx	Rx	xxx	A6	-	16b	2	[A6]	
Ra = Rb - [A16] [A16] = Ra - Rb	xxxx	Ra_{2-0}	Ra_3	xxx W Rb	A16	16b	3	[A16]	
Ra -= Rb	xxxx	Ra_{2-0}	Ra_3	xxx	Rb -	16b	2	R	
Rx -= {D:}[Ry@]	xxxx	Rx	xxx	D @	Ry -	16b	3 / 4	[R]	

Description: The group of instruction will be executed for subtraction without carry in arithmetical operation, i.e. Rd = X - Y. X, Y will have different meanings according to the addressing mode. See Table 6.2.

Example:

```
R10 -= 0x28;                // IM6
R12 = R11 - 0x2400;         // IM16
R13 -= [BP+0x08];           // [BP+IM6]
R14 -= [0x30];              // [A6]
BP = R14 - [0x2480];        // [A16]
[0x2480] = BP - R12;        // [A16], BP - R12 is assigned to MEM[0x2480]
SR -= R12;                  // R
PC -= D:[BP++];             // [R]
```

SBC **Subtract with Carry**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2 / Ext Code			N	Z	S	C
Rx -= IM6, Carry Rx = Rx - IM6, Carry	Xxxx	Rx	xxx	IM6		- / 16b	2	IM6	√	√	√	√
Ra = Rb - IM16, Carry	Xxxx	Ra2-0	Ra3	xxxx	Rb	IM16 / 16b	3	IM16				
Rx -= [BP+IM6] , Carry Rx = Rx - [BP+IM6] , Carry	Xxxx	Rx	xxx	IM6		16b	2 / 3	[BP+IM6]				
Rx -= [A6] , Carry Rx = Rx - [A6] , Carry	Xxxx	Rx	xxx	A6		- / 16b	2	[A6]				

Ra = Rb - [A16] , Carry [A16] = Ra - Rb, Carry	xxxx	Ra2-0	Ra3	xxx	W	Rb	A16	16b	3	[A16]				
Ra -= Rb, Carry	xxxx	Ra2-0	Ra3	xxx		Rb	-	16b	2	R				
Rx -= {D:}[Ry@], Carry	xxxx	Rx	xxx	D	@	Ry	-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for subtraction with carry in arithmetical operation, i.e. Rd = X – Y – C = X + (~Y) + C. X, Y will have different meanings according to the addressing mode. See Table 6.2.

Example:

```
R10 -= 0x28, Carry;                          // IM6
R12 = R11 - 0x2400, Carry;                   // IM16
R13 -= [BP+0x08] , Carry;                    // [BP+IM6]
R14 -= [0x30] , Carry;                       // [A6]
BP = R14 - [0x2480] , Carry;                 // [A16]
[0x2480] = BP - R12, Carry;                  // [A16], BP - R12 is assigned to MEM[0x2480]
SR -= R12, Carry;                            // R
PC -= D:[BP++], Carry;                       // [R]
```

NEG **Negative**

Syntax	Instruction Format						Word Group 2	Ext Code	Cycles	Addressing Mode	Flags			
	Word Group 1										N	Z	S	C
Rx = -IM6	xxxx	Rx	xxx	IM6			-	16b	2	IM6				
Ra = -IM16	xxxx	Ra_{2-0}	Ra_3	xxxx		Rb	IM16	16b	3	IM16				
Rx = -[BP+IM6] Rx = -[BP+IM6]	xxxx	Rx	xxx	IM6				16b	2 / 3	[BP+IM6]				
Rx = -[A6] Rx = -[A6]	xxxx	Rx	xxx	A6			-	16b	2	[A6]	√	√	√	√
Ra = -[A16] [A16] = -Ra	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb	A16	16b	3	[A16]				
Ra = -Rb	xxxx	Ra_{2-0}	Ra_3	xxx		Rb	-	16b	2	R				
Rx = -{D:}[Ry@]	xxxx	Rx	xxx	D	@	Ry	-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for negation in arithmetical operation, i.e. Rd = -X = ~X+1. The meaning of X will be described as follow according to the different addressing modes. See Table 6.2.

Example:
```
R10 = -0x28;                // IM6
R12 = -0x2400;              // IM16
R13 = -[BP+0x08];           // [BP+IM6]
R14 = -[0x30];              // [A6]
BP=-[0x2480];               // [A16]
[0x2480] = -BP;             // [A16], -BP is assigned to MEM[0x2480]
SR = -R12;                  // R
PC = -D:[BP++];             // [R]
```

CMP — Compare

Syntax	Instruction Format: Word Group 1						Word Group 2	Ext Code	Cycles	Addressing Mode	Flags N	Z	S	C
CMP Rx, IM6	xxxx	Rx	Xxx	IM6			-	16b	2	IM6	√	√	√	√
CMP Rb, IM16	xxxx	Ra_{2-0}	Ra_3	xxxx		Rb	IM16	16b	3	IM16				
CMP Rx, [BP+IM6]	xxxx	Rx	Xxx	IM6				16b	2 / 3	[BP+IM6]				
CMP Rx, [A6]	xxxx	Rx	Xxx	A6			-	16b	2	[A6]				
CMP Rb, [A16] CMP Ra, Rb	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb	A16	16b	3	[A16]				
CMP Ra, Rb	xxxx	Ra_{2-0}	Ra_3	xxx		Rb	-	16b	2	R				
CMP Rx, {D:}[Ry@]	xxxx	Rx	Xxx	D	@	Ry	-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for comparison in arithmetical operation, i.e. X - Y. But its result will not be stored and only affect NZSC flags. X, Y will have different meanings according to the addressing mode. See Table 6.2.

Example:
```
CMP R11, 0x27;              // IM6
CMP R11, 0x1227;            // IM16
CMP R13, [BP+0x08];         // [BP+IM6]
CMP R14, [0x30];            // [A6]
CMP R14, [0x2480];          // [A16]
CMP R1, R12;                // R
CMP R14, D:[BP++];          // [R]
```

AND — Logical AND

Syntax	Instruction Format							Word Group 2	Ext Code	Cycles	Addressing Mode	Flags			
	Word Group 1											N	Z	S	C
Rx &= IM6 Rx = Rx & IM6	xxxx	Rx	xxx	IM6				-	16b	2	IM6	√	√	√	√
Ra = Rb & IM16	xxxx	Ra_{2-0}	Ra_3	xxxx		Rb		IM16	16b	3	IM16				
Rx &= [BP+IM6] Rx = Rx & [BP+IM6]	xxxx	Rx	xxx	IM6					16b	2 / 3	[BP+IM6]				
Rx &= [A6] Rx = Rx & [A6]	xxxx	Rx	xxx	A6				-	16b	2	[A6]				
Ra = Rb & [A16] [A16] = Ra & Rb	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb		A16	16b	3	[A16]				
Ra &= Rb	xxxx	Ra_{2-0}	Ra_3	xxx		Rb		-	16b	2	R				
Rx &= {D:}[Ry@]	xxxx	Rx	xxx	D	@	Ry		-	16b	3 / 4	[R]				

Description: The group of instruction will be executed in logical AND operation, i.e. Rd = X & Y. The X and Y will have different meanings according to the addressing mode. See Table 6.2.

Example:

```
R10 &= 0x28;                    // IM6
R12 = R11 & 0x2400;             // IM16
R13 &= [BP+0x08];               // [BP+IM6]
R14 &= [0x30];                  // [A6]
BP = R14 & [0x2480];            // [A16]
[0x2480] = BP & R12;            // [A16], BP & R12 is assigned to MEM[0x2480]
SR &= R12;                      // R
PC &= D:[BP++];                 // [R]
```

OR **Logical OR**

Syntax	Instruction Format							Word Group 2	Ext Code	Cycles	Addressing Mode	Flags			
	Word Group 1											N	Z	S	C
Rx |= IM6 Rx = Rx | IM6	xxxx	Rx	xxx	IM6				-	16b	2	IM6	√	√	√	√
Ra = Rb | IM16	xxxx	Ra_{2-0}	Ra_3	xxxx		Rb		IM16	16b	3	IM16				

Rx |= [BP+IM6] Rx = Rx | [BP+IM6]	xxxx	Rx	xxx	IM6				16b	2 / 3	[BP+IM6]			
Rx |= [A6] Rx = Rx | [A6]	xxxx	Rx	xxx	A6			-	16b	2	[A6]			
Ra = Rb | [A16] [A16] = Ra | Rb	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb	A16	16b	3	[A16]			
Ra |= Rb	xxxx	Ra_{2-0}	Ra_3	xxx		Rb	-	16b	2	R			
Rx |= {D:}[Ry@]	xxxx	Rx	xxx	D	@	Ry	-	16b	3 / 4	[R]			

Description: The group of instruction will be executed in logical OR operation, i.e. Rd = X | Y. The X and Y will have different meanings according to the addressing mode. See Table 10.

Description: The group of instruction will be executed in logical OR operation, i.e. Rd = X | Y. The X and Y will have different meanings according to the addressing mode. See Table 6.2.

Example:
```
R10 |= 0x28;                 // IM6
R12 = R11 | 0x2400;          // IM16
R13 |= [BP+0x08];            // [BP+IM6]
R14 |= [0x30];               // [A6]
BP = R14 | [0x2480];         // [A16]
[0x2480] = BP | R12;         // [A16], BP | R12 is assigned to MEM[0x2480]
SR |= R12;                   // R
PC |= D:[BP++];              // [R]
```

XOR — Logical Exclusive OR

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1				Word Group 2	Ext Code			N	Z	S	C
Rx ^= IM6 Rx = Rx ^ IM6	xxxx	Rx	xxx	IM6	-	16b	2	IM6	√	√	√	√
Ra = Rb ^ IM16	xxxx	Ra_{2-0}	Ra_3	xxxx Rb	IM16	16b	3	IM16				
Rx ^= [BP+IM6] Rx = Rx ^ [BP+IM6]	xxxx	Rx	xxx	IM6		16b	2 / 3	[BP+IM6]				
Rx ^= [A6] Rx = Rx ^ [A6]	xxxx	Rx	xxx	A6	-	16b	2	[A6]				

Ra = Rb ^ [A16] [A16] = Ra ^ Rb	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb	A16	16b	3	[A16]				
Ra ^= Rb	xxxx	Ra_{2-0}	Ra_3	xxx		Rb	-	16b	2	R				
Rx ^= {D:}[Ry@]	xxxx	Rx	xxx	D	@	Ry	-	16b	3 / 4	[R]				

Description: The group of instruction will be executed in logical exclusive OR operation, i.e. Rd = X ^ Y. The X, Y will have different meanings according to the addressing mode. See Table 6.2.

Example:
```
R10 ^= 0x28;                    // IM6
R12 = R11 ^ 0x2400;             // IM16
R13 ^= [BP+0x08];               // [BP+IM6]
R14 ^= [0x30];                  // [A6]
BP = R14 ^ [0x2480];            // [A16]
[0x2480] = BP ^ R12; SR ^=      // [A16], BP ^ R12 is assigned to MEM[0x2480]
R12;                            // R
PC ^=D:[BP++];                  // [R]
```

TEST — Logical TEST

Syntax	Instruction Format: Word Group 1						Word Group 2	Ext Code	Cycles	Addressing Mode	Flags N	Z	S	C
TEST Rx, IM6	xxxx	Rx	xxx	IM6			-	16b	2	IM6	√	√	√	√
TEST Rb, IM16	xxxx	Ra_{2-0}	Ra_3	xxxx		Rb	IM16	16b	3	IM16				
TEST Rx, [BP+IM6]	xxxx	Rx	xxx	IM6				16b	2 / 3	[BP+IM6]				
TEST Rx, [A6]	xxxx	Rx	xxx	A6			-	16b	2	[A6]				
TEST Rb, [A16] TEST Ra, Rb	xxxx	Ra_{2-0}	Ra_3	xxx	W	Rb	A16	16b	3	[A16]				
TEST Ra, Rb	xxxx	Ra_{2-0}	Ra_3	xxx		Rb	-	16b	2	R				
TEST Rx, {D:}[Ry@]	xxxx	Rx	xxx	D	@	Ry	-	16b	3 / 4	[R]				

Description: The group of instruction will be executed for logical AND operation, i.e. X&Y. However, its result will not be stored and it only affects NZ flags. The X and Y will have different meanings according to the addressing mode. See Table 6.2.

Example:
```
TEST R11, 0x27;          // IM6
TEST R11, 0x1227;        // IM16
```

```
TEST R13, [BP+0x08];    // [BP+IM6]
TEST R14, [0x30];       // [A6]
TEST R14, [0x2480];     // [A16]
TEST R1, R12;           // R
TEST R14, D:[BP++];     // [R]
```

TSTB/SETB/CLRB/INVB — **Bit operations with direct memory addressing**

Syntax	Instruction Format						Cycles	Addressing Mode	Flags			
	Word Group 1					Word Group 2			N	Z	S	C
BIT_OP {D:}[A16], offset	xxxxx	DS	xxxx	BIT_OP	offset	A16	2	A16		√	-	-

Description: Executing bit operation with the value at memory location indexed by 16 bits operand. Users can use the "D:" indicator to access memory space large than 64K words, If "D:" indicator is used, the MSB 6-bit 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.

BIT_OP	Syntax	Meaning
TSTB	TSTB {D:}[A16], offset;	Z= (MEM[{DS,A16}][offset]== 1)? 1'b0: 1'b1
SETB	SETB {D:}[A16], offset;	MEM[{DS,A16}][offset]= 1
CLRB	CLRB {D:}[A16], offset;	MEM[{DS,A16}][offset]]= 0
INVB	INVB {D:}[A16], offset;	MEM[{DS,A16}][offset]= ~ MEM[{DS,A16}]offset]

Example:
```
SETB [0x5678], 5;       // MEM[0x5678][5]= 1
SETB D:[0x1234], 13;    // If DS=3, MEM[{0x31234}][13]= 1
```

6.2.3 Instruction Set Summary

Table 6.3

Type	Operation	Cycles	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
DSI6	DS=IM6	1	1	1	1	1	1	1	1	0	0	0	IM6					
CALL	CALL A22	3	1	1	1	1	-	-	0	0	0	1	A22[21:16]					
											A22[6							
JMPF	GOTO A22	3	1	1	1	1	1	1	1	0	1	0	A22[21:16]					
											A22[6							

Type	Operation	Cycles	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JMPR	GOTO MR	2	1	1	1	1	1	1	1	0	1	1	-	-	-	-	-	-
FIR_MOV	FIR_MOV ON/OFF	1	1	1	1	1	-	-	-	1	0	1	0	0	0	1		0~fir
Fraction	Fraction ON/OFF	1	1	1	1	1	-	-	-	1	0	1	0	0	0	1	1	Fr
INT SET	INT FIQ/IRQ	1	1	1	1	1	-	-	-	1	0	1	0	0	0	0	F	I
IRQ	IRQ ON/OFF	1	1	1	1	1	-	-	-	1	0	1	0	0	1	0	0	I
SECBANK	SECBANK ON/OFF	1	1	1	1	1	-	-	-	1	0	1	0	0	1	0	1	S
FIQ	FIQ ON/OFF	1	1	1	1	1	-	-	-	1	0	1	0	0	1	1	F	0
IRQ Nest Mode	IRQNEST ON/OFF	1	1	1	1	1	-	-	-	1	0	1	0	0	1	1	N	1
BREAK	BREAK	4	1	1	1	1	-	-	-	1	0	1	1	-	-	0	0	0
CALLR	CALL MR	4	1	1	1	1	-	-	-	1	0	1	1	-	-	0	0	1
DIVS	DIVS MR,R2	1	1	1	1	1	-	-	-	1	0	1	1	-	-	0	1	0
DIVQ	DIVQ MR,R2	1	1	1	1	1	-	-	-	1	0	1	1	-	-	0	1	1
EXP	R2 = EXP R4	1	1	1	1	1	-	-	-	1	0	1	1	-	-	1	0	0
NOP	NOP	1	1	1	1	1	-	-	-	1	0	1	1	-	-	1	0	1
DS Access	DS=Rs/ Rs=DS	1 (c)	1	1	1	1	-	-	-	0	0	0	1	0	W	Rs		
FR Access	FR=Rs/ Rs=FR	1 (d)	1	1	1	1	-	-	-	0	0	0	1	1	W	Rs		
MUL	MR = Rd* Rs	1 (e)	1	1	1	S_{Rs}	Rd			S_{Rd}	0	0	0	0	1	Rs		
MULS	MR = [Rd]*[Rs], size	(f)	1	1	1	S_{Rs}	Rd			S_{Rd}	1	SIZE				Rs		
Register BITOP	BITOP Rd,Rs	1 (g)	1	1	1	0	Rd			0	0	0	Bit op		0	Rs		
Register BITOP	BITOP Rd,offset	1	1	1	1	0	Rd			0	0	1	Bit op		offset			
Memory BITOP	BITOP DS:[Rd],offset	1	1	1	1	0	Rd			1	1	Ds	Bit op		offset			
Memory BITOP	BITOP DS:[Rd],Rs	1	1	1	1	0	Rd			1	0	Ds	Bit op		0	Rs		
Memory BITOP	BITOP DS:[A16],offset	2	1	1	1	1	-	Ds	1	0	0	1	Bit op		offset			
											A16							
Shift	Rd=Rd LSFT Rs	1/2 (h)	1	1	1	0	Rd			1	0	LSFT			1	Rs		
RETI	RETI	6/7 (i)	1	0	0	1	1	0	1	0	1	0	0	1	1	0	0	0
RETF	RETF	6	1	0	0	1	1	0	1	0	1	0	0	1	0	0	0	0
Base+Disp6	Rd = Rd op [BP+IM6]	1/2 (j)	OP				Rd			0	0	0	IM6					
Imm6	Rd = Rd op IM6	1 (k)	OP				Rd			0	0	1	IM6					
Branch	Jxx label	1/4 (l)	COND?				1	1	1	0	0	D	IM6					
Indirect	PUSH/POP Rx,Ry to/from [Rs]	N+1/N+2 (m)	OP				Rd			0	1	0	SIZE		Rs			

Type	Operation	Cycles	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
DS_Indirect	Rd = Rd op DS:[Rs++]	2/3 (n)	OP				Rd			0	1	1	Ds	@		Rs		
Imm16	Rd = Rs op IMM16	2	OP				Rd			1	0	0	0	0	1	Rs		
											MMI6							
Direct16	Rd = Rs op A16	2	OP				Rd			1	0	0	0	1	W	Rs		
											A16							
Direct6	Rd = Rd op A6	1	OP				Rd			1	1	1	A6					
Register	Rd = Rd op Rs SFT sfc	1 (o)	OP				Rd			1	SFT			SFC		Rs		
Ext Code			1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
Ext Register	Ra=Ra op Rb	2	OP				Ra_{2-0}			Ra_3	0	0	0	0	Rb			
Ext Push/Pop	PUSH/POP Rx, Ry to/from [Rb]	N+2/N+3 (m)	W	SIZE			Rx			0	0	0	1	0	Rb			
Ext IMM16	Ra=Rb op IMM16		3_3 OP				Ra_{2-0}			Ra	0	1	0	0	Rb			
											MMI6							
Ext A16	Ra=Rb op [A16]		3_3 OP				Ra_{2-0}			Ra	0	1	1	W	Rb			
											A16							
Ext DS_Indirect	Rx=Rx op Ds:[Ry++]	3/4 (n)	OP				Rx			0	1	0	Ds	@		Ry		
Ext IM6	Rx=Rx op IM6	2	OP				Rx			1	1	0	IM6					
Ext Base+Disp6	Rx=Rx op [BP+IM6]	2/3 (j)	OP				Rx			0	1	1	IM6					
Ext A6	Rx=Rx op [A6]	2	OP				Rx			1	1	1	A6					

(a) Rd/Rs: R0-R7, Ra/Rb: R0-R15, Rx/Ry: R8-R15.

(b) Extend Operation: use 0xff80 as extension prefix code and followed with extend instruction.

(c) DS Access: W = 0 Rs=Ds, W = 1 Ds=Rs.

(d) FR Access: W = 0 Rs=FR, W = 1 FR=Rs.

(e) MUL: S_{Rd} = 0, Rd is unsigned else Rd is signed, S_{Rs} = 0, Rs is unsigned else Rs is signed. Operation Mode: Rd*Rs: unsigned x unsigned, unsigned x signed, signed x signed.

Rd: support R0-R6 only.

(f) MULS: if FIR MOV flag is on, the parameter array index by Rd will be shift 1 word forward, so need additional N cycles.

S_{Rd} = 0, Rd is unsigned else Rd is signed, S_{Rs} = 0, Rs is unsigned else Rs is signed. Operation Mode: Rd X Rs: unsigned x unsigned, unsigned x signed, signed x signed. Size=0~16 and OP[6:3]=4'b0000 indicate executing 16 levels inner

product.

Rd: support R0-R6 only.

(No Bus Conflict, FIR_MOV OFF): N+2

(No Bus Conflict, FIR_MOV ON): 2N+1

(Bus Conflict, FIR_MOV OFF): 2N+2

(Bus Conflict, FIR_MOV ON): 3N

(g) Bit op:

00	01	10	11
test	set	clear	inverse

(h) Shift: ASR/LSL/LSR/ROL/ROR: 1 cycle, ASROR/LSLOR/LSROR: 2 cycles LSFT:

000	001	010	011	100	101	110	111
ASR	ASROR	LSL	LSLOR	LSR	LSROR	ROL	ROR

(i) RETI: IRQ interrupts with IRQNEST ON must restore FR from stack, so 7 cycles are needed to execute RETI, other interrupts need 6 cycles only.

(j) Base+Disp6: read operation need 1 cycle to calculate BP+IM6 address first, so cycle count will increase by 1, write operation only need 1 executing cycle. Extend operation will increase additional 1 cycle.

Rd: support R0-R6 only.

(k) IMM6: Rd: support R0-R6 only.

(l) Branch taken: 4 cycles, not taken: 1 cycle.

(m) Push: N+1 cycles, Pop: N+2 cycles, POP with update PC: N+4, Extend operation will increase additional 1 cycle.

(n) DS_indirect: read memory with [++Rs] prefix need 1 cycle to calculate [++Rs] address, so cycle count is 3, other operation only need 2 executing cycles. Extend operation will increase additional 1 cycle. @: prefix

00	01	10	11
Rs	Rs--	Rs++	++Rs

(o) SFT: Shift type

000	001	010	011	100	101
NOP	ASR	LSL	LSR	ROL	ROR

SFC: Shift count

00	01	10	11
1	2	3	4

(p) If any instruction updates PC value, CPU will flush the pipeline registers and need 2 additional cycles to fetch the new instruction.

6.3 Stall Condition

unSP 2.0 is a pipelined architecture micro processor, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in the pipeline. If some combinations of instructions cannot be accommodated because of resource conflicts, *unSP* 2.0 will add stall cycles to pipeline data path to resolve such hazards.
All stall condition of *unSP* 2.0 are list as below.

1. **DAG Read after Write:** Data read after write stall, because *unSP* 2.0 separate memory read and memory write at different pipeline stage, we must make sure data read/write sequence must keep in order of instruction execution

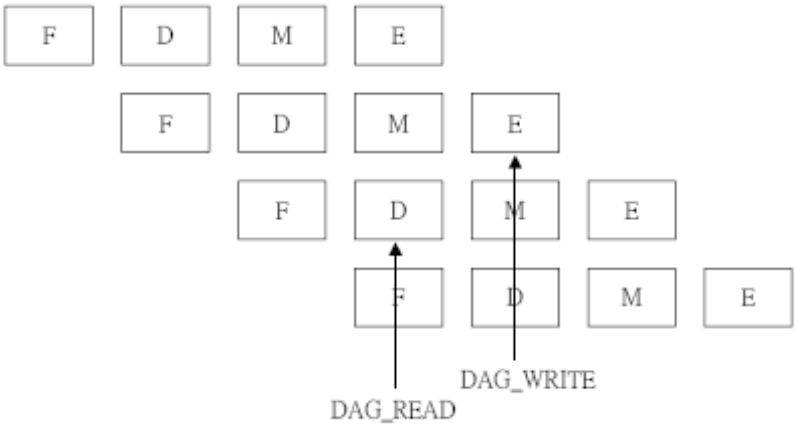


Figure 6.1

Example :

```
[R2] = R3;
R4 = [0x3];        // If this instruction will read data memory and
                   // the previous would write data memory, stall 1 cycle.
```

2. **DAG Read after Branch:** Branch condition is tested at execution stage of branch instruction and the next 2 instructions would be fetched into pipeline sequentially. if branch is taken, these two instructions will be dropped, there may be wrong data read access occurred if the next instruction of branch need to read data from memory. *unSP* 2.0 will stall 1 cycle to avoid such wrong access.

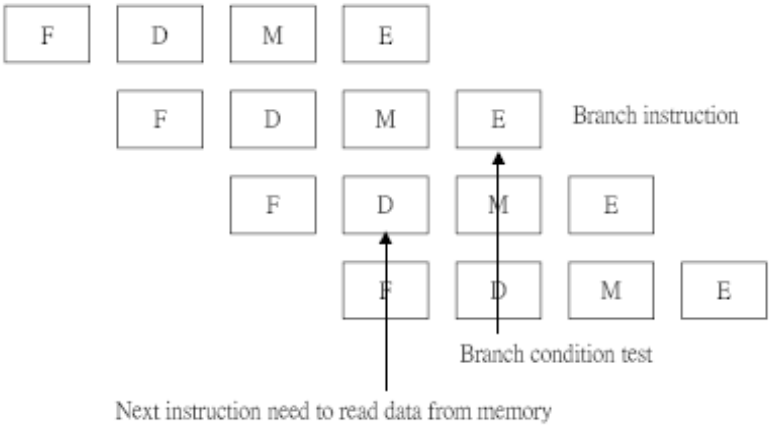


Figure 6.2

Example:

JNE label1; // if there are memory read access instruction after branch will stall 1 cycle.

R2 = [R3];

3. **IAG input RF not ready:** If IAG use RF value as next address source, but the referenced RF value is not ready or is updating in EXE stage. *unSP* 2.0 will stall until RF value is ready. We don't forward register write back from EXE stage to decode stage in this situation to cut critical path.

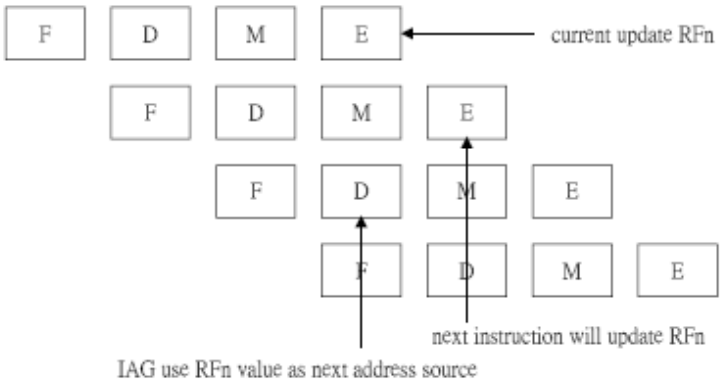


Figure 6.3

Example:

R3 = 0x1234;

GOTO MR; // use register which is destination register of previous instruction

// as next instruction address source will stall 1 cycle

R2 = R1; // instruction A

NOP;

MR = R1 * R2; // MULS will send Rs to IAG Bus as address, and the previous instruction A

// will update Rs in decode stage of MULS , stall 1 cycle to cut critical path

4. **DAG input RF not ready:** If DAG use RF value as next address source, but the referenced RF value is not ready or is updating in EXE stage. *unSP* 2.0 will stall until RF value is ready. We don't forward register write back from EXE stage to decode stage in this situation to cut critical path.

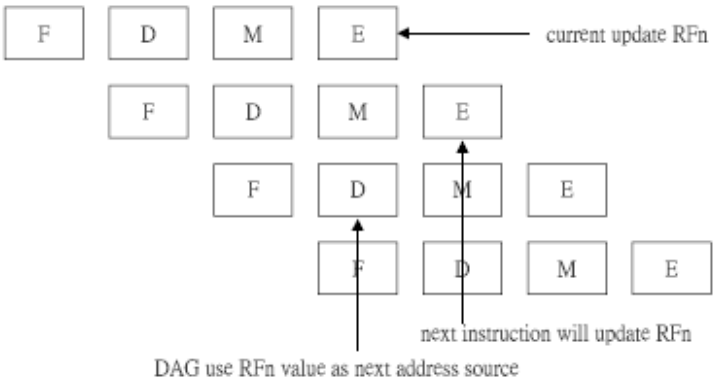


Figure 6.4

Example:

```
R2=R1+2;
R3=[R2]; // If use register which is the destination register of previous instruction // as memory access source
                register will add 1 stall cycle
```

5. **MAC forward stall:** Cut critical path from MAC unit output in execution stage to register files read in decode stage

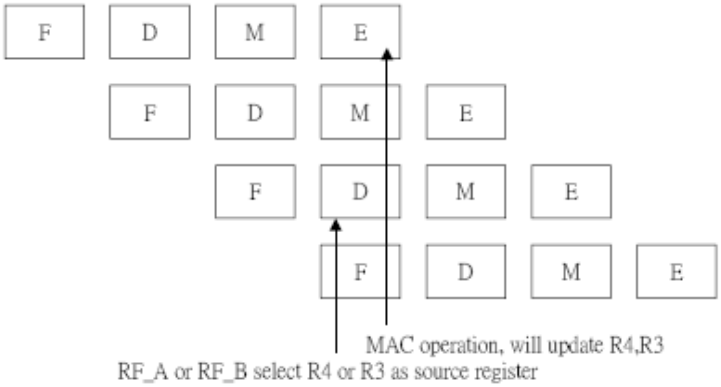


Figure 6.5

Example:

```
MR = R1 * R2;
R1 += 1;
R2 -= R4;          // If Rs=R3, R4 will stall 1 cycle.
```

6. **Bank change stall:** In *unSP* 2.0, registers value are read at decode stage, and bank flag is changed at execution stage, we must use bank flag to select which bank registers are read in decode

stage, so if bank flags will be changed by SETF or SET FR instruction, we must stall pipeline before bank flag is changed.

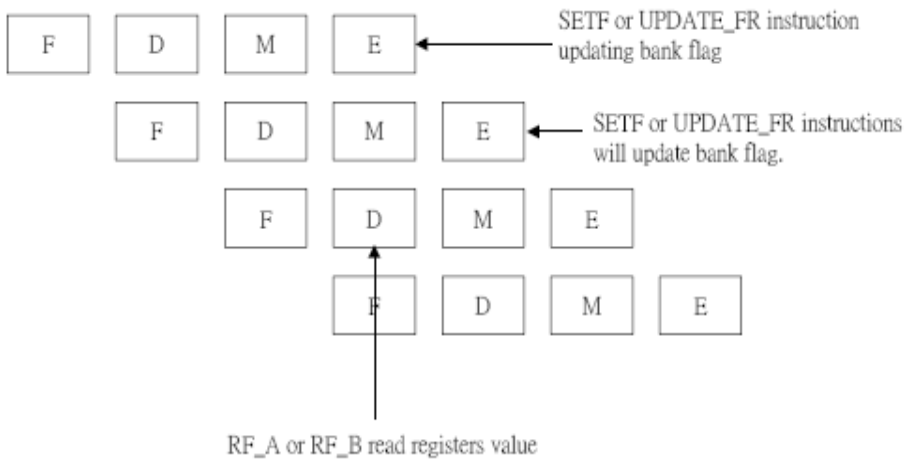


Figure 6.6

Example:

SECBANK ON; // if this instruction will change bank (SECBANK on/off or FR=Rs) and

// either the next 2 instruction use R1-R4 as source register will stall 1 cycle.

R2=R1+2;

FR = R2; // if this instruction will change bank (SECBANK on/off or FR=Rs) and

// either the next 2 instruction use R1-R4 as source register will stall 1 cycle.

NOP;

R1 = R3 - R2;

7. **Shifter source not ready stall:** Shifter unit is placed at MR stage, we need to stall pipeline if any resource needed by shift operation is not ready.

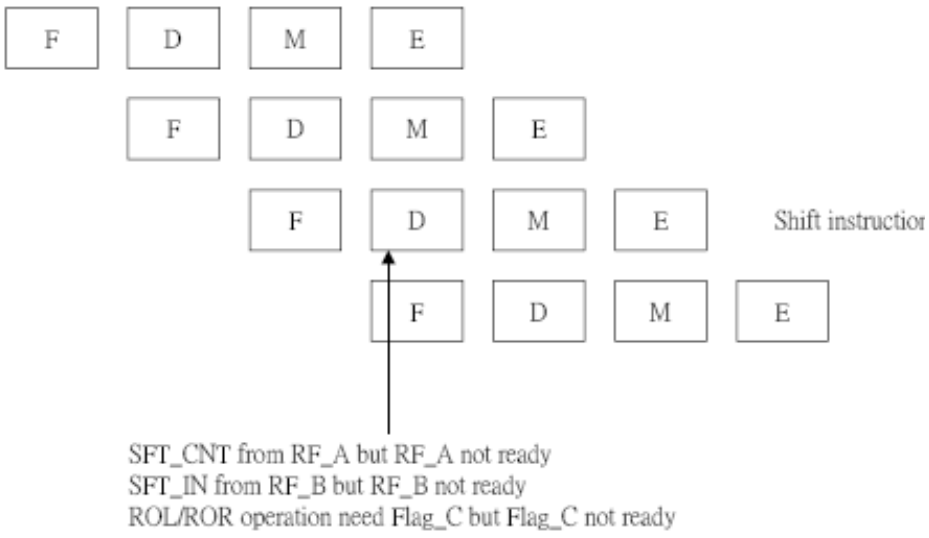


Figure 6.7

Example : R1 = 0x5;

```
R2 = R2 ASR R1; // SFT_CNT comes from register which is destination register of
                // previous instruction will stall 1 cycle
R2 = R3;
R4 = R2 LSL R1;          // SFT_IN comes from register which is destination register of // previous
                         instruction will stall 1 cycle
R1 = R1 + 0x1234;
R2 = R3 ROL R4;     // ROR/ROL operation needs C Flag which would be updated
                    // by previous instruction will stall 1 cycle
```

8. **Delay calculate register not ready:** DS_IND with ++Rs prefix, POP, BP+IM6 instructions need to read memory with address calculated by register and offset, To cut critical path, *unSP* 2.0 will delay 1 cycle and store register value into pipelined registers then send this register and offset to DAG to calculate address. If the source register value is not ready in decode stage, *unSP* 2.0 will stall until register is ready.

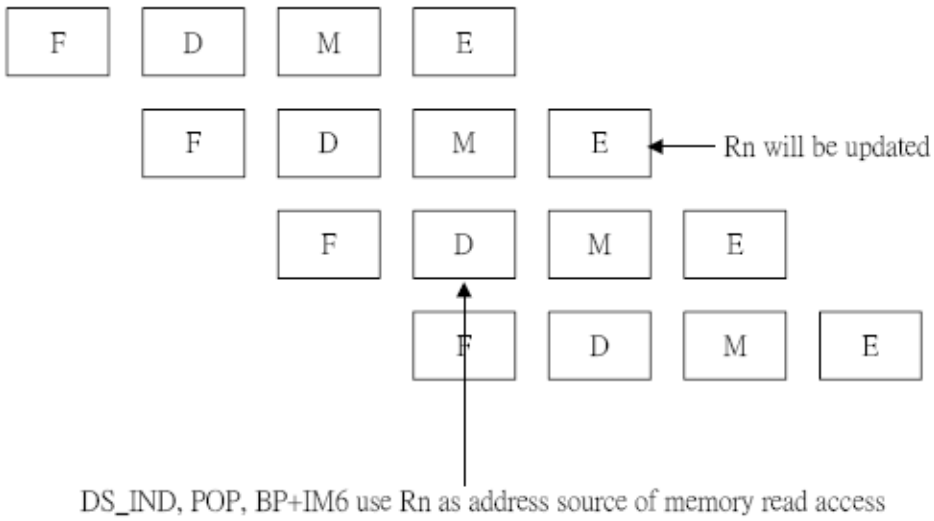


Figure 6.8

Example :

```
BP = 0x5678;
R2 = [BP+0x34];     // Use the register which will be updated by previous
                    instruction // as address source of memory read
```

9. **MAC Write Stall:** Stall for MULS operation to update memory content. If FIR_MOV flag is on, *unSP* 2.0 will shift right 1 word of the memory content indexed by Rd. We need to stall 1 cycle between continuous data read for data write back.

10. **DATA Bus not ready:** *unSP* 2.0 will stall if data bus not ready.

11. **INST Bus not ready:** *unSP* 2.0 will stall if inst bus not ready.

12. **BUS Fighting stall:** *unSP* 2.0 will stall if INST Bus and Data Bus is fighting.

7 Appendix A Difference Between *unSP*-1.2 & *unSP*-1.3

- **New behavior of checking interrupt**

  In *unSP* 1.3, CPU does not check interrupt after RETI instruction. Besides, CPU dos not check interrupt after MDS access instructions

- **Configurable multiplier**

  The multiplication of *unSP* 1.2 needs 9 cycles to sum up partial products. In order to accelerate the multiplication, the multiplier comes with *unSP*1.3 can be configured to sum up these partial product in one-cycle. Nevertheless, extra 6K gates are needed for this multiplier

- **Stack access pin**

  The stack access pin output (high active) is used to indicate that CPU is accessing (reading or writing) stack. The system designer can use this signal to detect stack underflow or overflow. The system becomes more robust.

  A pin is added to indicate current instruction reads or writes the stack. These instructions are:

  - PUSH ... to [SP]
  - POP … from [SP]
  - The action of pushing SR and OC when an interrupt occurs
  - The action of pushing FR, SR and PC when higher priority interrupt occurs in nested interrupt mode
  - RETI
  - RETF
  - CALL

- **Modified nested interrupt supports**

  IRQ_NEST mode is always ON. Then, *unSP* 1.3 will save PC, SR and FR into memory when serving IRQ or FIQ, and restore them after IRQ or FIQ service routine.

  IRQ_ENABLE is turned off automatically when *unSP* 1.3 performing IRQ service routine. User can turn on IRQ_ENABLE in IRQ service routine to allow higher priority IRQ to interrupt. *unSP* 1.3 can **re-execute** FIQ service routine when serving FIQ if FIQ_ENABLE is on. Both FIQ_ENABLE and IRQ_ENABLE are turned off automatically when *unSP* 1.3 performing FIQ service routine.

- **Address change pin**

  The address change pin output (low active) is used to indicate that address is changing.

8 Appendix B Difference Between *unSP*-2.0 & *unSP*-1.2

- **The behavior of CS auto increase in memory bank boundary**

Since *unSP* 2.0 is a 4 stage pipelined architecture processor, it will use a 22-bit IAG unit to pre-fetch the instruction data and increase instruction address automatically, the behavior of CS auto increase in memory bank boundary is different from *unSP* 1.2 which designed by multi-cycle architecture.

For example:

INST address from 0x1 FFFE – 0x20000, the behavior of SR and PC

Table 8.1

INST address	*unSP*2.0		*unSP*1.2	
	SR	PC	SR	PC
0x1FFFE	0x0001	0xFFFE	0x0001	0xFFFE
0x1FFFF	0x0001	0xFFFF	0x0002	0xFFFF
0x20000	0x0002	0x0000	0x0002	0x0000
0x20001	0x0002	0x0001	0x0002	0x0001

- **Disable interrupt detect instructions.**

unSP 1.2 will check the interrupt signals(FIQ/IRQ) at the last cycle of every instruction except RA16 (Direct16 instruction with read) for semaphore implementation of the real-time operating system.

unSP 2.0 check the interrupt signals(FIQ/IRQ) at the decode stage of every instruction except RA16/RW16 (Direct16 instruction with read/write) and RETI instructions for semaphore implementation of the real-time operating system and make sure the user's program will be executed at least 1 instruction after leaving interrupt service routine to avoid infinite loop of interrupt servicing.

9 Appendix C Comparison Between *unSP* Versions

	unSP 1.0	*unSP* 1.1	*unSP* 1.2	*unSP* 1.3	*unSP* 2.0
Memory Bus	Single Bus	Single Bus	Single Bus	Single Bus	Separate Inst/Data
Address depth	22-bit	22-bit	22-bit	22-bit	22-bit / 22-bit
Data width	16-bit	16-bit	16-bit	16-bit	16-bit / 16-bit
Pipeline	No	No	No	No	4 stage pipeline
General Registers (R1-R4)	Yes	Yes	Yes	Yes	Yes
System Registers (SP, BP, SR, PC)	Yes	Yes	Yes	Yes	Yes
Second Bank Registers (SR1-SR4)	No	No	Yes	Yes	Yes
Inner Flag Register (FR)	No	No	Yes	Yes	Yes
Extend Registers (R8-R15)	No	No	No	No	Yes
Interrupt Sources	10 (FIQ,IRQ,BRK)	10 (FIQ,IRQ,BRK)	10 (FIQ,IRQ,BRK)	10 (FIQ,IRQ,BRK)	10 (FIQ,IRQ,BRK)
Nested IRQ	No	No	Yes	Yes	Yes
Average CPI	6	5	5	5	2
Area (TSMC 0.35um)	-	750umx650um	2220umx488.4um CPU: 9K ICE: 3K	CPU: 14.3K ICE: 3.7K	2100umx950um CPU: 19.5K ICE: 5K
Speed	-	80MHz(TC)	111.1 MHz(TC)	58.8 MHz(WC)	109.6 MHz(TC)
Power	-	0.16 mA/MHz	0.3 mA/MHz	-	-
MAC operation	Signed x signed Signed x unsigned	Signed x signed Signed x unsigned	Signed x signed Signed x unsigned Unsigned x unsigned	Signed x signed Signed x unsigned Unsigned x unsigned	Signed x signed Signed x unsigned Unsigned x unsigned
MAC Cycles	13	12	12/13(uxu)	12/13(uxu)	1
Guard bits	No	No	4	4	4

	unSP 1.0	*unSP* 1.1	*unSP* 1.2	*unSP* 1.3	*unSP* 2.0
Fraction mode	No	No	Yes	Yes	Yes
Division	No	No	1- bit Non-restoring division (DIVS/DIVQ)	1- bit Non-restoring division (DIVS/DIVQ) and single instruction division (DIVUU/DIVSS)	1 bit Non-restoring division (DIVS/DIVQ)
EXP	No	No	Yes	Yes	Yes
Bit operation	No	No	Register / Memory	Register / Memory	Register / Memory
16 bits shifter	No	No	Yes	Yes	Log Shifter
DS access	No	No	Yes	Yes	Yes
FR access	No	No	Yes	Yes	Yes
SS access	No	No	No	Yes	No
MDS access	No	No	No	Yes	No
Far jump	Yes	Yes	Yes	Yes	Yes
Far indirect jump	No	No	Yes	Yes	Yes
Far indirect call	No	No	Yes	Yes	Yes
Extend Operations	No	No	No	No	Yes
Immediate (I6 / I16)	Yes	Yes	Yes	Yes	Yes
Direct (A6 / A16)	Yes	Yes	Yes	Yes	Yes
Indirect (DS indirect)	Yes	Yes	Yes	Yes	Yes
Relative (BP+IM6)	Yes	Yes	Yes	Yes	Yes
Multiple indirect (Push/Pop)	Yes	Yes	Yes	Yes	Yes
Byte Register Indirect	No	No	No	Yes	No
Byte Indexed Address	No	No	No	Yes	No
Byte Register Indexed	No	No	No	Yes	No

10 Appendix D CPU Cycle Formula and Examples

c.1 unSP 1.2 Cycle Formula

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		

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

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	
Push/ Pop					
PUSH R_H, R_L to [Rs]	1	4+2*N+N*DM+PM	20		PUSH BP, R1 to [SP]; SP points to DM
POP R_H, R_L to [Rs]	1	4+2*N+N*DM+PM	20		POP R1, BP from [SP]; SP points to DM
Multiplication					
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

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	
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]

c.2 unSP 1.3 Cycle Formula

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		
Jcond (not-taken)	1	2+PM	3		
Jcond (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					

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

			Example (SPCE060)		
Instruction	**Code Width**	**CPU Cycle Formula** **Rd is NOT**	**Program Memory**	**1**	**Example Condition**
	(unit: word)	**(if pc)**	**Data Memory**	**1**	
			Counter(N)	**5**	
PUSH R_H, R_L to [Rs]	1	4+2*N+N*DM+PM	20		PUSH BP, R1 to [SP]; SP points to DM
POP R_H, R_L to [Rs]	1	4+2*N+N*DM+PM	20		POP R1, BP from [SP]; SP points to DM
Multiplication					
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
DIVUU MR,R2	1	48+PM	49		DIVUU MR,R2
DIVSS MR,R2	1	47+PM	48		DIVSS 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
Bitop D:[A16],offset	2	8+PM+2*DM	11		CLRB [0x1234],0x3
Flag Setting					
DS=Rs / Rs=DS	1	2+PM	3		
FR=Rs / Rs=FR	1	2+PM	3		
SS=Rs / Rs=SS	1	2+PM	3		
MDS=R3 / R3=MDS	1	2+PM	3		
ALU Operation					

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	
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]
Rd ÙB/W:[Rs@]	1	10+PM+DM	12		
B:[Rs@]=IMM8	2	10+PM+DM	12		
W:[Rs@]=IMM 16	2	10+PM+DM	12		
Rd ÙB/W:[BP+IM6]	1	6+PM+DM	8		
B:[BP+IM6]=IMM8	2	6+PM+DM	8		
W:[BP+IM6]=IMM16	2	6+PM+DM	8		
Rd ÙB/W:[Rs@]	1	9+PM+DM	11		
B:[Rs@]=IMM8	2	9+PM+DM	11		
W:[Rs@]=IMM16	3	9+PM+DM	11		

c.2 unSP 2.0 Cycle Formula

Instruction	Code Width (unit: word)	CPU Cycle Formula (if Rd is NOT pc)	Example (SPCE060)		Example Condition (may not reveal to customers)
			Program Memory	1	
			Data Memory	1	
			Counter(N)	5	
Control Flow					
CALL	2	3+2*PM+2*DM	7		call func_1
RETF	1	2+PM+2*DM	5		
RETI (IRQ, FIQ, BRK)	1	2+PM+2*DM	5		
RETI (Nested IRQ)	1	3+PM+3*DM	7		
BREAK	1	4+2*PM+2*DM	8		
GOTO	2	3+2*PM	5		
Jcond (not-taken)	1	1+PM	2		
Jcond (taken)	1	4+2*PM	6		
GOTO MR	1	3+2*PM	5		
CALL MR	1	4+2*PM+2*DM	8		
NOP	1	1+PM	2		
Operation Mode					
INT IRQ, FIQ	1	1+PM	2		
INT OFF	1	1+PM	2		
INT FIQ	1	1+PM	2		
INT IRQ	1	1+PM	2		
IRQ ON/OFF	1	1+PM	2		
FIQ ON/OFF	1	1+PM	2		
FIR_MOV ON/OFF	1	1+PM	2		
FRACTION ON/OFF	1	1+PM	2		
SECBANK ON/OFF	1	1+PM	2		
IRQNEST ON/OFF	1	1+PM	2		
Push/ Pop					
PUSH R_H, R_L to [Rs]	1	1+N+N*DM+PM	12		PUSH R1, BP to [SP], SP points to DM

Instruction	Code Width (unit: word)	CPU Cycle Formula (if Rd is NOT pc)	Example (SPCE060)		Example Condition (may not reveal to customers)
			Program Memory	1	
			Data Memory	1	
			Counter(N)	5	
POP R_H, R_L from [Rs]	1	2+N+N*DM+PM	13		POP R1, BP from [SP], SP points to DM
Multiplication					
MR = Rd* Rs, {ss,us}	1	1+PM	2		MR=R1*R2
MR = [Rd]*[Rs], {ss,us}, N	1	DM/PM no conflict, FIR_MOV OFF 2+N+N*MAX(PM,DM)+PM	13		MR=[R1],[R2], 5 Rd points to OM, Rs points to PM
		DM/PM no conflict, FIR_MOV ON 1+2*N+N*MAX(PM,DM)+N*DM+PM	22		MR=[R1],[R2], 5 Rd points to OM, Rs points to PM
		DM/PM conflict, FIR_MOV OFF 2+2*N+N*(PM+DM)+PM	23		MR=[R1],[R2], 5 Rd points to OM, Rs points to PM
		DM/PM conflict, FIR_MOV ON 3*N+(N+1)*PM+(2N-1)*DM	30		MR=[R1],[R2], 5 Rd points to OM, Rs points to PM
Division					
DIVS MR,R2	1	1+PM	2		DIVS MR,R2
DIVQ MR,R2	1	1+PM	2		DIVQ MR,R2
Exponential Detect					
R2 = EXP R4	1	1+PM	2		R2 = EXP R4
Shift Operation					
Rd = Rd LSFT Rs	1	1+PM	2		R1 = R1 ASR R2
Bit Operation					
Bitop Rd, Rs	1	1+PM	2		TSTB R1,R2
Bitop Rd,offset	1	1+PM	2		SETB R2,0x3
Bitop D:[Rd], Rs	1	1+PM+DM	3		CLRB [R1],R3

Instruction	Code Width (unit: word)	CPU Cycle Formula (if Rd is NOT pc)	Example (SPCE06		Example Condition (may not reveal to customers)
			Program Memory	1	
			Data Memory	1	
			Counter(N)	5	
Bitop D:[Rd],offset	1	1+PM+DM	3		INVB [R2],0x4
Bitop D:[A16],offset	2	2+2*PM+DM	5		SETB D:[0x7016],0x8
Flag Setting					
DS=Rs / Rs=DS	1	1+PM	2		
FR=Rs / Rs=FR	1	1+PM	2		
ALU Operation					
Rd= Rd op [BP+IM6]	1	2+PM(read) / 1+PM(stall)	3	2	R1=R1+[BP+3]
Rd=Rd op D:[Rs@]	1	2+PM+DM / 3+PM+DM ([++Rs])	4	5	R3=R3+D:[R1++]/ R3=R3+D:[++R1]
Rd = Rd op Rs SFT sfc	1	1+PM	2		R2=R2-R3 ASR 2
Rd=Rd op IM6	1	1+PM	2		R1=R1+0x8
Rd=Rd op IM16	2	2+2*PM	4		R2=R1+0x5678
Rd=Rd op [A6]	1	1+PM+DM	3		R3=[0x20]
Rd=Rs op [A16]	2	2+2*PM+DM	5		R4=[0x7600]
Extend Instruction					
Ra=Ra op Rb	2	2+2*PM	4		R2=R8+R3
PUSH Rx,Ry to [Rb]	2	2+N+N*DM+2*PM	14		PUSH R8,R13 to [SP], SP points to DM
POP Rx,Ry from [Rb]	2	3+N+N*DM+2*PM	15		POP R8,R13 from [SP], SP points to DM
Ra=Rb op IM16	3	3+3*PM	6		R9=R4+0x5678
Ra=Rb op [A16]	3	3+3*PM+DM	7		R10=R1+[0x7016]
Rx=Rx op D:[Ry@]	2	3+2*PM+DM / 4+2*PM+DM ([++Rs])	6	7	R8=R8+D:[R10++] / R8=R8+D:[++R10]
Rx=Rx op IM6	2	2+2*PM	4		R9=R9+0x8
Rx=Rx op [BP+IM6]	2	3+2*PM(read) / 2+2*PM(stall)	5	4	R8=R8+[BP+3]
Rx=Rx op [A6]	2	2+2*PM+DM	5		R15=[0x20]