OKI

nX-8/500S Core

Instruction Manual

CMOS 16-bit microcontroller

SECOND EDITION
ISSUE DATE: JUNE, 1999

NOTICE

1. The information contained herein can change without notice owing to product and/or technical improvements. Please make sure before using the product that the information you are referring to is up-to-date.
2. The outline of action and examples of application circuits described herein have been chosen as an explanation of the standard action and performance of the product. When you actually plan to use the product, please ensure that the outside conditions are reflected in the actual circuit and assembly designs.
3. NO RESPONSIBILITY IS ASSUMED BY US FOR ANY CONSEQUENCE RESULTING FROM ANY WRONG OR IMPROPER USE OR OPERATION, ETC. OF THE PRODUCT.
4. Neither indemnity against nor license of a third party's industrial and intellectual property right, etc. is granted by us in connection with the use of the product and/or the information and drawings contained herein. No responsibility is assumed by us for any infringement of a third party's right which may result from the use thereof.
5. The products described herein fall within the category of strategical goods, etc. under the Foreign Exchange and Foreign Trade Control Law. Accordingly, before exporting the product you are required under the Law to file the application for the export license by your local Government.
6. No part of the contents contained herein may be reprinted or reproduced without our prior permission.

Table of Contents

Chapter 0. Preface

Chapter 1. Architecture

Chapter 2. Addressing Modes

Chapter 3. Instruction Details

X

Chapter 0. Preface

This chapter explains the configuration and usage of this manual.

Preface

This manual describes the instruction set of the nX-8/500S core. The nX-8/500S core is used as the CPU core of Oki Electric's original CMOS 8/16-bit single-chip microcontrollers. As one of the OLMS-66K Series cores, the nX-8/500S core is higher end than nX-8/200 and nX-8/400. The first device to use the nX-8/500S core is the MSM66556/589.

The explanations in this manual presume the basic architecture of the nX-8/500S core. The basic architecture incorporates the maximum functionality of the nX-8/500S core. In this basic architecture data memory space and code memory space each have a capacity of 16M bytes (64K bytes×256 segments), and the architecture provides instructions for manipulating these spaces. Depending on the device you actually use, the actual capacity and instruction set may be subsets of the basic architecture. Refer to the user's manual of your device for information on any such limitations.

The following manuals are for products related to the nX-8/500S core. Please read them as well.

■MSM665xx User's Manual
The MSM665xx User's Manual describes the hardware of your target device.

■MAC66K Assembler Package User's Manual
The MAC66K Assembler Package User's Manual explains assembly language syntax and the use of the relocatable assembler, linker, librarian, and object converter.

■Macroprocessor MP User's Manual
The Macroprocessor MP User's Manual explains macroprocessing language syntax and the use of the general-purpose macroprocessor.

■EASE665xx User's Manual
The EASE665xx User's Manual describes the EASE665xx emulator and SID665xx debugger.

This manual consists of three chapters.

Chapter 1 describes the basic architecture of the nX-8/500S core.
This chapter explains how programs make use of major resources, such as registers and memory. It then describes particular features and restrictions of programming. This chapter provides the basic knowledge needed to understand Chapter 2 and Chapter 3.

Chapter 2 describes addressing modes.
This chapter explains the coding syntax to access register and memory resources. It also explains the operation of these accesses in detail.

Chapter 3 describes the functions of each instruction.
This chapter explains the functions and detailed operation of instructions, and provides instruction codes. It presents instructions in alphabetic order, so it can be used for reference.

This manual uses the following terminology.

■Values

Numeric expressions and address expressions are basically the same as those used with RAS66K.　Refer to the manual for the assembler package for details.

■Ranges

A-B represents a range of values that includes A and B.　A-B is used in some places where it clearly will not be confused with subtraction.

■Addresses

Complete address expressions for the nX-8/500S are coded using a physical segment number (#0 to #255) and an offset within the segment (0 to 0FFFFH), as shown below.

physical_segment_number : offset_within_segment

■Examples■

0:0	Offset address 0 in physical segment #0.
0FFH:0FFFFH	Offset address 65535 in physical segment #255.
CSR:1000H	Address 1000H in the code segment indicated by CSR.
TSR:1000H	Address 1000H in the table segment indicated by TSR.
DSR:1000H	Address 1000H in the data segment indicated by DSR.

However, the offset within a segment is sometimes coded alone as an address where there is no chance for confusion.　In particular, an address and an offset within a segment are the same thing when programming for a device that does not access multiple segments or when a program exists entirely within one segment.

Physical segments and logical segments

For the nX-8/500S, blocks of 64K bytes in memory space are called physical segments, but this manual often simply calls them segments.　Blocks allocated to memory by a program are also called segments, but these are specifically logical segments.

Chapter 1. Architecture

This chapter explains the basic architecture of the nX-8/500S. The basic architecture is the major functional specification of the nX-8/500S. Any microcontroller utilizing this core will have the same functions or a subset of them.

1-1. Overview

1-1-1. Overview Of OLMS-66K Series And nX-8/500S Core

The OLMS-66K Series of devices are single-chip microcontrollers that integrate Oki Electric's original 16-bit CPU as their core with various peripheral circuits. Currently the OLMS-66K Series provides the target cores listed below. This series has expanded with improvements in processing efficiency in the CPU cores while program compatibility has been maintained.

The nX-8/500S core maintains upward compatibility at the basic assembler level with the nX-8/200 and nX-8/400 cores, but adds instructions and speeds up frequently used instructions. At the same time is extends the accessible memory space and adds addressing modes.

Core	Device	Description
nX-8/100	MSM66101	Reduced instruction version of nX-8/200.
nX-8/200	MSM66201/207	Reduced instruction version of nX-8/300.
nX-8/300	MSM66301	First core of OLMS-66K series.
nX-8/400	MSM66417	High-speed version of nX-8/200.
nX-8/500S	MSM66556/589	Basic assembly language level upward compatibility with nX-8/200 to nX-8/400.

The nX-8/500S centers its processing around its accumulator and register set. It provides nearly identical functions for byte data processing and word data processing. A flag (the data descriptor) determines which type of data is being calculated in the accumulator. Thus the same instruction codes provide functions that are the same for byte data and word data calculations, but are switched by the state of the data descriptor flag.

Instruction codes are configured in 8-bit units, with lengths of 1 to 6 bytes. Highly efficient programs can be coded by making use of both native instructions for frequent types of processing, and composite instructions for a wide variety of addressing modes.

Memory of the nX-8/500S is split into program memory space and data memory space. Each space can be 16M bytes, configured as 256 physical segments of 64K bytes each. Segments are specified by three segment registers. Code memory also has a vector type area for resets, interrupts, and 1-byte calls, and an ACAL area for 2-byte calls. Segments of data memory are configured as 256 pages of 256 bytes each. More efficient addressing is provided for the SFR page, in which peripheral function control registers are located, and the fixed page and current page.

1-2. CPU Resources And Programming Model

This section describe registers and memory configurations and their roles as CPU resources used in programming.

1-2-1. Registers

The nX-8/500S utilizes processing methods centered around an accumulator and register sets. The register sets includes a local register set for storing mainly data and a pointing register set for mainly storing addresses. In addition to these, the nX-8/500S has registers for controlling program flow and registers for controlling memory, which together make up the programming model for registers. This section lists the registers used in programs and then describes the functions of each in detail.

■ Accumulator

		15 – 8	7 – 0
Needed for calculations.	A(ACC)		

■ Control registers (CR)

This register group controls program flow and stores its current state.

		15 – 8	7 – 0
Program Status Word	PSW		
Program Counter	PC		
Local Register Base	LRB		
System Stack Pointer	SSP		

■ Pointing registers (PR)

There are eight pointing register sets, each with four 16-bit registers X1, X2, DP, and USP. The pointing register sets store memory addresses for indirect addressing. They also provide the same functions for word calculations as extended local registers, so they can be used as data registers too.

		15 – 8	7 – 0
Index Register 1	X1		
Index Register 2	X2		
Data Pointer	DP		
User Stack Pointer	USP		

■ Local registers (ER)

There are 256 local register sets, each with eight 8-bit registers. Each two adjacent 8-bit registers comprise an extended local register (ERn) for processing word data. This data register group is used for storage and calculations of byte and word data.

		15 – 8	7 – 0
Extended local register #0	ER0	R1	R0
Extended local register #1	ER1	R3	R2
Extended local register #2	ER2	R5	R4
Extended local register #3	ER3	R7	R6

■ Segment registers

These three 8-bit registers each select a physical segment that contains program code, read-only data, and read/write data respectively. For devices with limited memory capacity, the number of bits implemented in the actual registers may be correspondingly limited. Some devices do not even implement segment registers.

		7 … 0
Code Segment Register	CSR	
Table Segment Register	TSR	
Data Segment Register	DSR	

■ ROM window control register

This 8-bit register is used to open a ROM window.

	7 … 4	3 … 0
ROMWIN		

1-2-1-1. Accumulator (A)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	AH								AL							

The accumulator is a 16-bit register around which calculations are centered. It can process words and bytes data. The low byte of the accumulator (AL) can also specify a bit in a bit array. The accumulator is normally accessed by accumulator addressing. However, because it is allocated as a word register in SFR space, it can also be manipulated with SFR addressing (sfr ACC). The accumulator's value immediately after a reset is 0. After an interrupt, the accumulator's value is automatically pushed on the stack. When an RTI instruction is executed, that value is popped from the stack and stored back in A.

■ Example■Accumulator usage

```
L      A,WORD_VAR     ; Word instruction         (A←WORD_VAR)
LB     A,BYTE_VAR     ; Byte instruction         (AL←BYTE_VAR)
MB     C,A.3          ; Bit instruction          (C←A.3)
SBR    BIT_ARRAY      ; Bit array instruction    (AL is bit specifier)
MOV    ACC,BASE[X2]   ; SFR addressing           (ACC←(BASE+X2))
```

1-2-1-2. Control Registers (CR)

The control register group controls program flow and stores its current state. Each 16-bit register has a specific function. The information stored in these registers is often collectively called the program context.

1-2-1-2-1. Program Status Word (PSW)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSW	C	Z	HC	DD	S	MIP	OV	MIE	MAB	F1	BCB1	BCB0	F0	SCB2	SCB1	SCB0
	PSWH								PSWL							

The PSW is configured as flags and fields that store and specify program status. The flag states can be tested with conditional branch instructions. The PSW is allocated as a word register in the SFR area, so it can also be accessed with SFR addressing (sfr APSW). After an interrupt, PSW contents are automatically pushed on the stack. When an RTI instruction is executed, those contents are popped from the stack and stored back in the PSW.

The high byte of the program status word (PSWH) consists of five flags that store the states of CPU calculation results, one flag that indicates the data type in the accumulator, and two flags that control interrupts.

The low byte of the program status word (PSWL) consists of a flag for multiply-accumulate calculations, a field that specifies the size of the common area, a field that selects the pointing register set, and two flags that are for the user.

The operation of each flag and field is described below.

C Carry flag (bit 15)

The carry flag stores the carry or borrow from unsigned calculations. It is set to 1 when the most significant bit in a arithmetic or comparison instruction generated a carry or borrow. It is reset to 0 in all other cases. The most significant bit is bit 15 for word calculations and bit 7 for byte calculations. The carry flag is also used as a bit accumulator for bit moves and bit logical operations. The SC and RC instructions are provided to set and reset the carry flag.

Z Zero flag (bit 14)

The zero flag indicates if the result of a calculation was 0. It is set to 1 when the execution result of any calculation instruction (such as arithmetic, logical, comparison, and accumulator data move instructions) or the object bit of any bit manipulation is zero. It is reset to 0 in all other cases.

HC Half-Carry flag (bit 13)

The half-carry flag is provided for implementing decimal arithmetic. It is set to 1 when bit 3 in a arithmetic or comparison instruction generated a carry or borrow. It is reset to 0 in all other cases.

DD Data Descriptor (bit 12)

The data descriptor indicates the type of data in the accumulator (A). It is a flag that determines the type of calculation for which the accumulator (A) will be used. It indicates word data when 1, and byte data when 0. The SDD and RDD instructions are provided to set and reset the data descriptor.

S Sign flag (bit 11)

The sign flag indicates the sign of calculation results. It is set to 1 when the sign bit (most significant bit) of the execution result of an arithmetic, comparison, or logical calculation was 1. It is reset to 0 in all other cases. The most significant bit is bit 15 for word calculations and bit 7 for byte calculations.

MIP Mask Interrupt Priority flag (bit 10)

The mask interrupt priority flag controls the priority function of maskable interrupts. It enables the priority function when 1, and disables the priority function when 0.

OV Overflow flag (bit 9)

The overflow flag stores the carry or borrow from signed calculations. It is set to 1 when the result of a arithmetic or comparison instruction exceeds the range that can be expressed with 2's complement numbers. It is reset to 0 in all other cases. The range is −32767 to +32767 for word data, and −128 to +127 for byte data.

MIE Mask Interrupt Enable flag (bit 8)

The mask interrupt enable flag controls whether all maskable interrupts are enabled or disabled. It enables interrupts when 1, and disables interrupts when 0. The EI and DI instructions are provided to set and reset MIE.

MAB Multiply-Accumulate Register Bank flag (bit 7)

The multiply-accumulate register bank flag specifies the bank of registers used for multiply-accumulate calculations (MAC instruction).

F1,F0 User flags 1, 0 (bit 6, bit 3)

The user flags are available for the user in programs. Programs can be written such that these flags are automatically updated in the PSW after interrupts.

BCB_{1-0} Bank Common Base (bit 5 to 4)

The bank common base specifies the last address of the area that is common between segments. The table below shows the relation between these bits and the selected common area.

No.	BCB Value 1	BCB Value 0	Common Area Range
0	0	0	0 to 03FFH
1	0	1	0 to 1FFFH
2	1	0	0 to 3FFFH
3	1	1	0 to 7FFFH

$SCB_{2\text{-}0}$ System Control Base (bit 2 to 0)

The system control base selects the pointing register set. The table below shows the relation between these bits and the selected pointing register set.

No.	SCB Value 2	1	0	Addresses of Pointing Register Set
0	0	0	0	0200H to 0207H
1	0	0	1	0208H to 020FH
2	0	1	0	0210H to 0217H
3	0	1	1	0218H to 021FH
4	1	0	0	0220H to 0227H
5	1	0	1	0228H to 022FH
6	1	1	0	0230H to 0237H
7	1	1	1	0238H to 023FH

1-2-1-2-1-1. How Instructions Change PSW Flags

The next page lists the instructions that change PSW flags when executed. However, the list basically excludes instructions that directly write to PSW or PSWH (such as instructions with sfr addressing). The table shows the flag name where the flag changes. It shows 1 where the flag is set and 0 where the flag is reset. It is blank where the flag does not change.

■ How instructions change PSW flags

Instruction Type	Mnemonics		Flag Changed C	Z	S	OV	HC	DD
Move								
	L, LB CLR, CLRB	(if destination is A)		Z				DD
	LC, LCB			Z				
Increment/Decrement								
	INC, INCB, DEC, DECB			Z	S	OV	HC	
Multiplication								
	MUL, MULB, SQR, SQRB			Z				
Division								
	DIV, DIVB		C	Z				
	DIVQ		C	Z		OV		
Arithmetic/Comparison								
	NEG, ADD, ADC, SUB, SBC NEGB, ADDB, ADCB, SUBB, SBCB CMP, CMPB, CMPC, CMPCB		C	Z	S	OV	HC	
Logical								
	AND, OR, XOR ANDB, ORB, XORB			Z	S			
Sign Extend								
	EXTND				S			1
Bit Manipulation/Bit Test								
	SB, RB, SBR, RBR, TBR			Z				
DD Manipulation								
	SDD, RDD							DD
Carry Manipulation								
	SC, RC		C					
Bit Move To Carry								
	MB, MBR	(if destination is C)	C					
Logical With Carry								
	BAND, BOR, BXOR BANDN, BORN		C					
Rotate/Shift With Carry								
	ROL, ROR, SLL, SRL, SRA ROLB, RORB, SLLB, SRLB, SRAB		C					
Return From Interrupt								
	RTI		C	Z	S	OV	HC	DD
Pop Data To PSW								
	POP	(if operand is PSW or CR)	C	Z	S	OV	HC	DD
Reset								
	BRK		0	0	0	0	0	0

1-2-1-2-2. Program Counter (PC)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC																

The PC is a 16-bit counter that stores the address of the program code to be executed next. It increments immediately after the program code is fetched from program memory. Repetition of this operation causes the flow of program execution. Branch instructions set the PC to new addresses of program code.

The PC exists as an independent register, and is not allocated in SFR space. The PC is overwritten by execution of branch instructions, but you do not need to be especially aware of the PC.

Immediately after a reset, the PC value will become the contents of the reset vector. After an interrupt, the address at which execution is to resume will be automatically pushed on the stack. That value will be popped back into the PC when an RTI instruction is executed.

1-2-1-2-3. Local Register Base (LRB)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LRB	LRBH								LRBL							

The LRB is a 16-bit register. Its high 8 bits and low 8 bits have independent functions.

The high 8 bits of the LRB (LRBH) specify the location of the current page. The current page is one of the 256 pages in the data segment specified by DSR. A single page is a 256-byte space that starts at a page boundary. The starting address of the current page is given by LRBH×100H. Current page addressing (off Dadr) and current page SBA area addressing (sbaoff Badr) are provided for accessing the 256 bytes of the current page specified by LRBH.

The low 8 bits of the LRB (LRBL) specify the location of the local register set. The local register set is allocated in 8-bit units within the 2K bytes between offset 200H and 9FFH of physical segment #0 (0:200H to 0:9FFH). The starting address of the local register is given by LRBL×8+200H. Local registers are allocated in order R0, R1, R2, ..., R7 from this starting address. Local register addressing (Rn, ERn) is provided for accessing the local registers specified by LRBL.

LRB is allocated as a word register in SFR space, so it can be manipulated using SFR addressing. The value of LRB is undefined after reset, so its value should be set soon after program execution begins. If local register addressing or current page addressing is used before this, then an undefined memory address will be accessed. After an interrupt, the LRB's value is automatically pushed on the stack. When an RTI instruction is executed, that value is popped from the stack and stored back in LRB.

1-2-1-2-4. System Stack Pointer (SSP)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSP																

The SSP is a 16-bit register that stores the top stack address of the hardware stack. The hardware stack is a pushdown stack for pushing and popping registers upon execution of interrupt process transfers/returns, calls/returns, and PUSHS/POPS instructions. The SSP stores the top (lowest) address of this stack. The SSP is automatically decremented and incremented during execution processing.

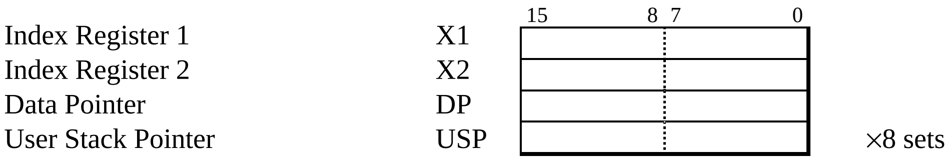
Data is normally pushed on and popped off the stack in word units. When a word value is pushed on the stack, the word data is written to the stack address specified by SSP, and then SSP is decremented by 2. When a word value is popped off the stack, SSP is incremented by 2, and then the word data is read from the stack address specified by SSP. Reads and writes to the memory of this word data are affected by word boundaries, so even if the SSP value is odd, the word data handled will be at the next lower even address. Pushing and popping the stack through SSP is always performed in accordance with these rules.

The hardware stack pointed to by SSP is always allocated in data segment #0 (0:0 to 0:0FFFFH). To access the stack with RAM addressing other than that of stack manipulation instructions, the DSR must be set to 0.

SSP is allocated as a word register in SFR space, so it can also be manipulated with SFR addressing. Immediately after reset, the value of SSP is 0FFFFH, the last address of memory. If there is no memory up to address 0FFFFH, then the actual value for SSP must be set soon after program execution begins. If instructions that manipulate the stack are executed before then, program operation will not be predictable.

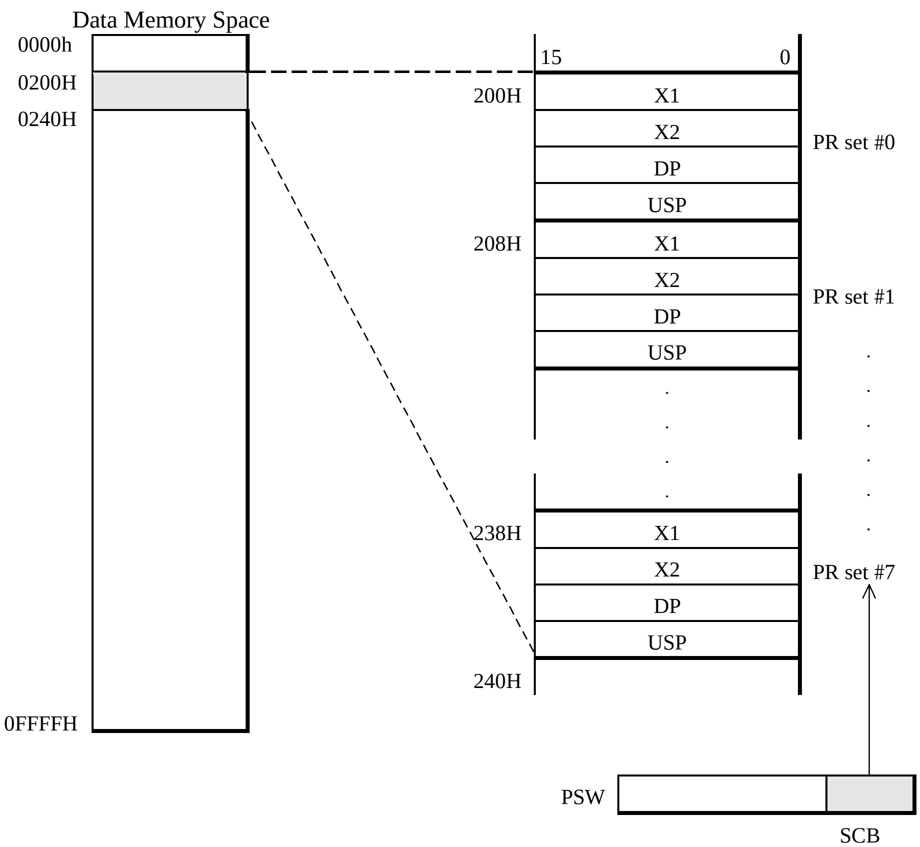
1-2-1-3. Pointing Registers (PR)

There are eight pointing register sets, each with four 16-bit registers X1, X2, DP, and USP. The pointing register sets store memory addresses for indirect addressing. They also provide the same functions for word calculations as extended local registers, so they can be used as data registers too.



The pointing register sets are allocated in the 64 bytes starting from address 200H in the fixed page of data memory space. They are allocated in order #0, #1,..., #7 from low address to high. Within each pointing register set, X1, X2, DP, and USP are allocated to memory in that order from low address to high.

■ Pointing register sets in data memory



The pointing register set to be used is selected by the SCB field in the PSW. The following table shows the relation between SCB field values and the pointing register set selected. Immediately after reset, pointing register set #0 will be selected. The initial values of all pointing registers are undefined.

■ SCB field and pointing register set addresses

No.	SCB Value 2	1	0	Pointing Register Set Addresses
0	0	0	0	0200H to 0207H
1	0	0	1	0208H to 020FH
2	0	1	0	0210H to 0217H
3	0	1	1	0218H to 021FH
4	1	0	0	0220H to 0227H
5	1	0	1	0228H to 022FH
6	1	1	0	0230H to 0237H
7	1	1	1	0238H to 023FH

The pointing register sets overlap the first eight local register sets (R0, R1, ..., R7), which also start from address 200H. To ensure proper program execution, set SCB and LRBL appropriately, such that the pointing registers and local registers do not overlap.

1-2-1-3-1. Addressing With Pointing Registers

Pointing register addressing modes are provided to access the contents of pointing registers.

■Example■Pointing register addressing

```
L       A,X1          ; A←X1
ADD     A,X2          ; A←A+X2
CMP     DP,#1234H     ; DP-1234H
ST      A,USP         ; A→USP
```

Index register 1 (X1) is used for indirect addressing ([X1]) where X1 itself specifies an address, indirect addressing with 16-bit base (D16[X1]) where an optional address within 64K bytes specifies a base address with X1 specifying an offset, and indirect addressing with 8-bit register displacement ([X1+A], [X1+R0]) where X1 specifies a base address anywhere in 64K bytes with an 8-bit register specifying an offset.

■Example■X1 indirect addressing

```
L       A,[X1]        ; X1 indirect addressing
ADD     A,1234[X1]    ; X1 indirect addressing with 16-bit base
SUB     A,[X1+A]      ; X1 indirect addressing with AL register displacement
AND     A,[X1+R0]     ; X1 indirect addressing with R0 register displacement
```

Index register 2 (X2) is used for indirect addressing with 16-bit base (D16[X2]) where an optional address within 64K bytes specifies a base address with X2 specifying an offset.

■Example■X2 indirect addressing

```
ADD     A,1234H[X2]     ; X2 indirect addressing with 16-bit base
```

The data pointer (DP) is used for indirect addressing ([DP]) where DP itself specifies an address, indirect addressing with post-increment/decrement ([DP+],[DP-]) where DP is automatically incremented or decremented after the data access, and indirect addressing with 7-bit displacement (n7[DP]) where DP specifies a base address anywhere in 64K bytes with an offset –64 to +63.

■Example■DP indirect addressing

```
L       A,[DP]          ; DP indirect addressing
ADD     A,[DP+]         ; DP indirect addressing with post-increment
SUB     A,[DP-]         ; DP indirect addressing with post-decrement
ADD     A,-12[DP]       ; DP indirect addressing with 7-bit displacement
```

The user stack pointer (USP) is used for indirect addressing with 7-bit displacement (n7[USP]) where USP specifies a base address anywhere in 64K bytes with an offset –64 to +63.

■Example■USP indirect addressing

```
L       A,-12[USP]      ; USP indirect addressing with 7-bit displacement
```

Like other byte objects, the low bytes of X1, X2, DP, and USP can be used as loop counter that specify 1 to 256 loops.

■Example■Loop counter usage

```
DJNZ    X1L,LOOP        ; X1 low byte (X1L) is loop counter
DJNZ    X2L,LOOP        ; X2 low byte (X2L) is loop counter
DJNZ    DPL,LOOP        ; DP low byte (DPL) is loop counter
DJNZ    USPL,LOOP       ; USP low byte (USPL) is loop counter
```

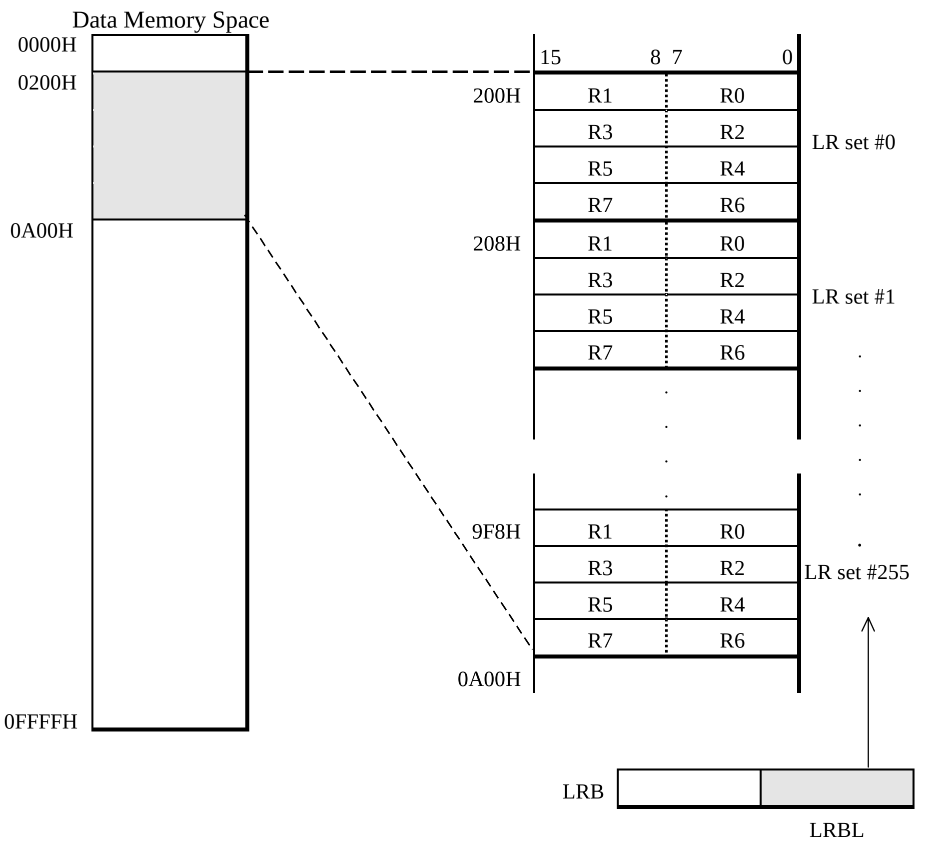
1-2-1-4. Local Registers (ER)

There are 256 local register sets, each with eight 8-bit registers. Each two adjacent 8-bit registers comprise an extended local register (ERn) for processing word data. This data register group is used for storage and calculations of byte and word data.

		15 … 8	7 … 0	
Extended local register #0	ER0	R1	R0	
Extended local register #1	ER1	R3	R2	
Extended local register #2	ER2	R5	R4	
Extended local register #3	ER3	R7	R6	×256 sets

The local register sets are allocated in the 2048 bytes starting from address 200H in the fixed page of data memory space. They are allocated in order #0, #1,..., #255 from low address to high. Within each local register set, R0 to R7 are allocated to memory in that order from low address to high.

■Local register sets in data memory



The local register set to be used is selected by the low byte of LRB (LRBL). The starting address of the local register set selected is given by LRBL× 8 + 200H. Immediately after reset, the value of LRBL is undefined, so there is no way to tell which local register set is selected. The initial values of all local registers are undefined.

■LRBL value and local register set addresses

No.	LRBL Value	Local Register Set Addresses
0	0	0200H to 0207H
1	1	0208H to 020FH
2	2	0210H to 0217H
3	3	0218H to 021FH
.	.	.
.	.	.
.	.	.
254	254	09F0H to 09F7H
255	255	09F8H to 09FFH

The first eight local register sets overlap the pointing register sets (X1, X2, DP, USP), which also start from address 200H. To ensure proper program execution, set LRBL and SCB appropriately, such that the local registers and pointing registers do not overlap.

1-2-1-4-1. Addressing With Local Registers

A byte-oriented local register addressing mode and word-oriented extended local register addressing mode are provided to access the contents of local registers.

■Example■Local register addressing

```
LB      A,R0          ; AL←R0
ADDB    A,R3          ; AL←AL+R3
CMPB    R6,#12        ; R6-12
STB     A,R7          ; A→R7
```

■Example■Extended local register addressing

```
L       A,ER0         ; A←ER0
ADD     A,ER1         ; A←A+ER1
CMP     ER2,#1234H    ; ER2-1234H
ST      A,ER3         ; A→ER3
```

For INCB and DECB instructions, R0 to R3 give more efficient instruction codes than R4 to R7.

■Example■INCB/DECB instructions

```
INCB    R0            ; 1-byte instruction
DECB    R3            ; 1-byte instruction
INCB    R4            ; 2-byte instruction
DECB    R7            ; 2-byte instruction
```

For DJNZ instructions, R4 and R5 give more efficient instruction codes for jumps in the range -128 to -1.

■Example■Loop instructions

```
LOOP:
          DJNZ    R4,LOOP          ; 2-byte instruction
          DJNZ    R0,LOOP          ; 3-byte instruction
          DJNZ    R5,NEXT          ; 3-byte instruction
NEXT:
```

For multiplication and division instructions, ER0, ER1, and R1 are used to store products, dividends, quotients, and remainders.

■Example■Multiplication and division instruction

```
MUL     obj                        ; <A,ER0>←A×obj
DIV     obj                        ; <A,ER0>←A÷ obj
                                   ; ER1←<A,ER0> mod obj
MULB    obj                        ; A←A×obj
DIVB    obj                        ; A←A÷ obj
                                   ; R1←A mod obj
```

R0 is used as a 1-byte unsigned displacement for addressing with X1 as a base.

■Example■X1 indirect addressing with R0 register displacement

```
MOV     A,[X1+R0]                  ; A←(X1+R0)
INCB    [X1+R0]                    ; (X1+R0)←(X1+R0)+1
```

1-2-1-5. Segment Registers

These 8-bit registers each select one of the 256 physical segments. CSR and TSR point to program memory space. CSR and TSR do not exist in devices with just one segment in program memory space. DSR points to data memory space. DSR does not exist in devices with just one segment in data memory space.

1-2-1-5-1. Code Segment Registers (CSR)

	7	6	5	4	3	2	1	0
CSR								

The CSR specifies which segment in program memory space contains the program code that is currently executing. It exists as an independent 8-bit register, so it is not allocated in SFR space. Writes to the CSR are performed by interrupts and by FJ, FCAL, FRT, and RTI instructions. The CSR cannot be written to by other methods.

A single segment has offset addresses 0 to 0FFFFH. Address calculations to determine the addressing of objects are performed with 16-bit offset addresses; overflows and underflows are ignored. Therefore, addressing alone will not change the CSR. Similarly, the CSR will not be changed if the PC overflows. Thus, program execution cannot proceed across code segment boundaries by any method other than those mentioned in the previous paragraph. Immediately after reset the CSR value will be 0.

When an interrupt occurs under the medium or large memory model, the current CSR will be automatically pushed on the stack along with the PC. The popped value will be restored to the CSR upon execution of an RTI instruction. (Refer to memory models.)

1-2-1-5-2. Table Segment Registers (TSR)

	7	6	5	4	3	2	1	0
TSR								

The TSR specifies which segment in program memory space contains table data. It is an 8-bit register allocated in SFR space, so it can be written by instructions that have SFR addressing. Data in the table segment is accessed using ROM reference instructions (LC, LCB, CMPC, CMPCB). RAM addressing of the table segment can also be performed by using the ROM window function.

A single segment has offset addresses 0 to 0FFFFH. Address calculations to determine the addressing of objects are performed with 16-bit offset addresses; overflows and underflows are ignored. Therefore, addressing alone will not change the TSR. Immediately after reset the TSR value will be 0.

1-2-1-5-3. Data Segment Registers (DSR)

	7	6	5	4	3	2	1	0
DSR								

The DSR specifies which segment in data memory space contains data. It is an 8-bit register allocated in SFR space, so it can be written by instructions that have SFR addressing. Data in the data segment is accessed using RAM addressing. The ROM window function opens a window in this data segment through which the table segment can be accessed.

A single segment has offset addresses 0 to 0FFFFH. Address calculations to determine the addressing of objects are performed with 16-bit offset addresses; overflows and underflows are ignored. Therefore, addressing alone will not change the DSR. Immediately after reset the DSR value will be 0.

1-2-1-6. ROM Window Control Register (ROMWIN)

	7	6	5	4	3	2	1	0
ROMWIN								

ROMWIN has the function of opening a ROM window. It is an 8-bit register allocated in SFR space. The lower 4 bits specify the starting address of the ROM window, and the upper 4 bits specify the ending address. The starting address will be $\text{ROMWIN}_{3\text{-}0}\times$1000H, and the ending address will be $\text{ROMWIN}_{7\text{-}4}\times$1000H + 0FFFH. For example, if 71H is written to ROMWIN, then the ROM window will be 1000H to 7FFFH. If the value written to the lower 4 bits is 0, then the ROM window function will not operate.

ROMWIN may be written only once after reset. Second and later writes will be ignored. Immediately after reset, the value of ROMWIN will be 0, so the ROM window function will not operate. To use the ROM window function, it is recommended that you open the ROM window soon after reset.

1-2-1-7. Special Function Registers (SFR)

Special function registers are a register group for controlling peripheral functions. They are allocated to addresses 0 to 1FFH in data memory space. In other words, nX-8/500S utilizes the concept of memory-mapped I/O. Refer to the section on data memory space for details.

1-2-2. Memory Space

The memory of nX-8/500S is split into program space and data space. The configurations of each of these spaces are described below.

1-2-2-1. Program Memory Space

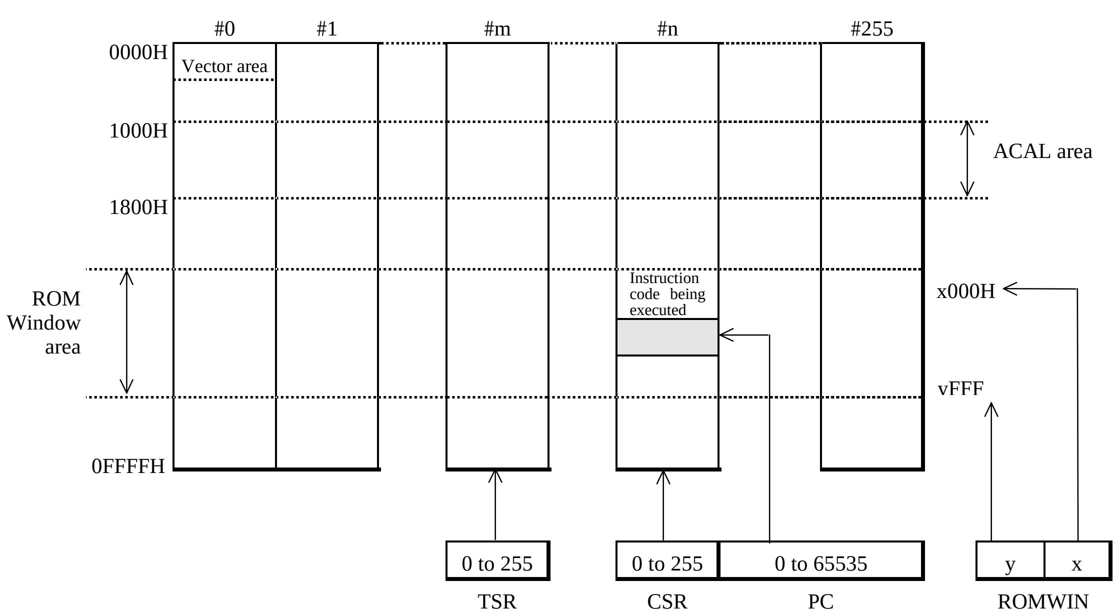
Program memory space of nX-8/500S has a total capacity of 16M bytes, configured as 256 segments of 64K bytes each. Program memory space contains executable instruction code (program code) and read-only data (table data).

The program code being executed is specified as 24 bits: CSR determines the high 8 bits, and PC determines the low 16 bits (CSR:PC). The segment selected by CSR is called the code segment. When instruction execution increments the PC or when relative jumps add displacements to the PC, overflows and underflows are ignored. This means that the CSR will not change.

The segment selected by TSR is called the table segment. The table segment can be accessed using table data addressing with the four instructions LC, LCB, CMPC, and CMPCB. RAM addressing can also access the table segment through use of the ROM window function.

A single segment has offset addresses 0 to 0FFFFH. Address calculations to determine the addressing of objects are performed with 16-bit offset addresses; overflows and underflows are ignored. Therefore, addressing alone will not change the TSR.

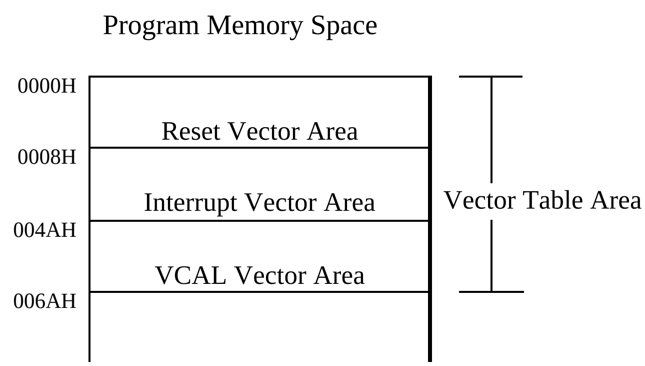
■Overview of program memory space



1-2-2-1-1. Vector Table Area

The 74 bytes from address 0 to 49H in segment #0 (0:0 to 0:49H) in program memory space are a vector table area for storing program process entry addresses (vectors) used after resets and interrupts. The 32 bytes from address 4AH to 69H (0:4AH to 0:69H) are a vector table area for storing program process entry addresses used when VCAL instructions are executed.

Each vector is a data word located at an even address. When control transfers to a program process, the CSR value is reset to 0 by hardware, selecting segment #0. Therefore entry addresses of program processes exist only in segment #0.



1-2-2-1-1-1. Reset Vector Area

The first four entries in the vector table are assigned as reset vectors corresponding to the sources of resets. Vector addresses and reset sources are as follows.

Vector Address	Reset Source
0000H	Reset pin (RES) input
0002H	System reset instruction (BRK) execution
0004H	Watchdog timer (WDT)
0006H	Op-code trap (OPTRP)

1-2-2-1-1-2. Interrupt Vector Area

Interrupt sources differ depending on the peripheral functions of each device. The interrupt vector area is assigned one non-maskable interrupt (NMI) and a maximum 32 maskable interrupts.

Vector Address	Interrupt Source
0008H	NMI pin input
000AH	Maskable interrupt #1
000CH	Maskable interrupt #2
.	.
.	.
.	.
0048H	Maskable interrupt #32

1-2-2-1-1-3. VCAL Table Area

The VCAL table area is a vector area for the 16 VCAL instructions (1-byte call instructions). Vector addresses and their corresponding VCAL instructions are as follows.

Vector Address	VCAL Instruction
004AH	VCAL 4AH
004CH	VCAL 4CH
004EH	VCAL 4EH
0050H	VCAL 50H
0052H	VCAL 52H
0054H	VCAL 54H
0056H	VCAL 56H
0058H	VCAL 58H
005AH	VCAL 5AH
005CH	VCAL 5CH
005EH	VCAL 5EH
0060H	VCAL 60H
0062H	VCAL 62H
0064H	VCAL 64H
0066H	VCAL 66H
0068H	VCAL 68H

1-2-2-1-1-4. Vector Table Coding Syntax

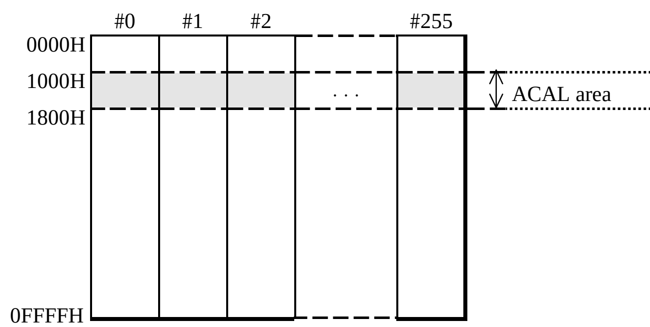
With the assembler, program process entry addresses are coded as labels in the operands of DW directives. An example program that defines the vector area is shown below. If the vector area other than the reset vector for reset pin (RES) input is not used for vectors, then it can be used for ordinary program code.

```
;
;Reset Vector Table
;
                CSEG    AT 0000H

                DW      START           ; Power on reset
                DW      BRK_RESET       ; BRK instruction
                DW      WDT_RESET       ; Watch dog timer overflow
                DW      OPTRP_RESET     ; Opecode trap
;
;Interrupt Vector Table
;
                DW      NMI_ENTRY       ; Non-maskable interrupt
                DW      INT0_ENTRY      ; Maskable interrupt #1
                         .
                         .
                         .
                DW      INTN_ENTRY      ; Maskable interrupt #n
;
;Vcal Vector Table
;
                CSEG    AT 004AH

VSUB0:          DW      SUB0            ; VCAL subroutine #0
VSUB1:          DW      SUB1            ; VCAL subroutine #1
                 .
                 .
                 .
VSUB15:         DW      SUB15           ; VCAL subroutine #15

;
; Start of main procedure
;
                EXTRN DATA:_$$SSP       ; Stack pointer initial address
START:
                MOV     SSP,#_$$SSP     ; Set system stack pointer
                 .
                 .
```

1-2-2-1-2 ACAL Area

The 2K bytes at addresses 1000H to 17FFH of each segment in program memory space (CSR:1000H to CSR:17FFH) are the ACAL area for placing the entry points of subroutines called by ACAL instructions. ACAL instructions are 2-byte instructions, so they are more efficient that 3-byte CAL instructions. An ACAL area exists in each physical segment.

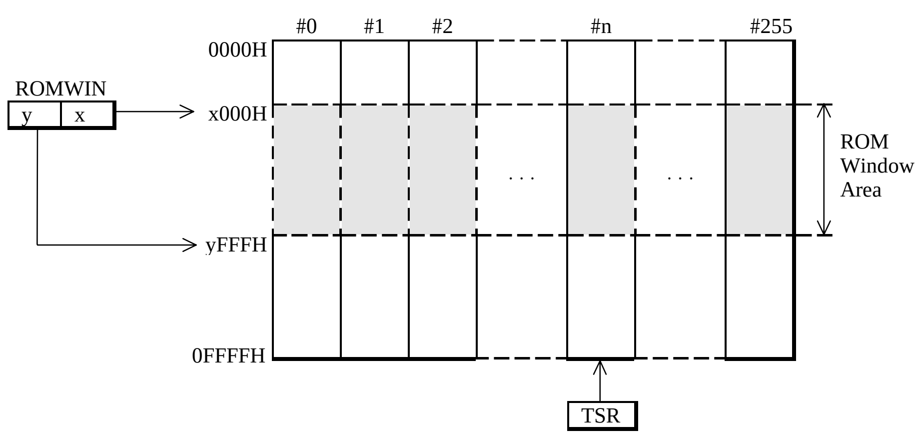
■ACAL area in program memory space



1-2-2-1-3. ROM Window Area In Program Memory Space

The ROM window area is allows data in the table segment specified by TSR to be accessed using RAM addressing (ROM window addressing). It is a program memory area that can be seen through a window opened in a data segment. Table data at the same address value can be read through the window, which can only be opened in areas that are not mapped to internal data memory. The range of the ROM window area is set with the ROM window function control register (ROMWIN).

■ROM window area in program memory space



1-2-2-1-4. Internal And External Program Memory Areas

There are no logical differences in programming when using internal and external program memory areas. Use the linker to place program code in internal program memory areas, which are implemented in the target device, and in external program memory areas, which are mounted in the target system.

Internal program memory size depends on the device. Refer to the user's manual of the target device for details.

1-2-2-2. Data Memory Space

Data memory space of nX-8/500S has a total capacity of 16M bytes, configured as 256 segments of 64K bytes each. Data memory space normally contains memory that is readable and writable.

The segment selected by DSR is called the data segment. The data segment can be accessed using RAM addressing. RAM addressing can also access the table segment through use of the ROM window function.

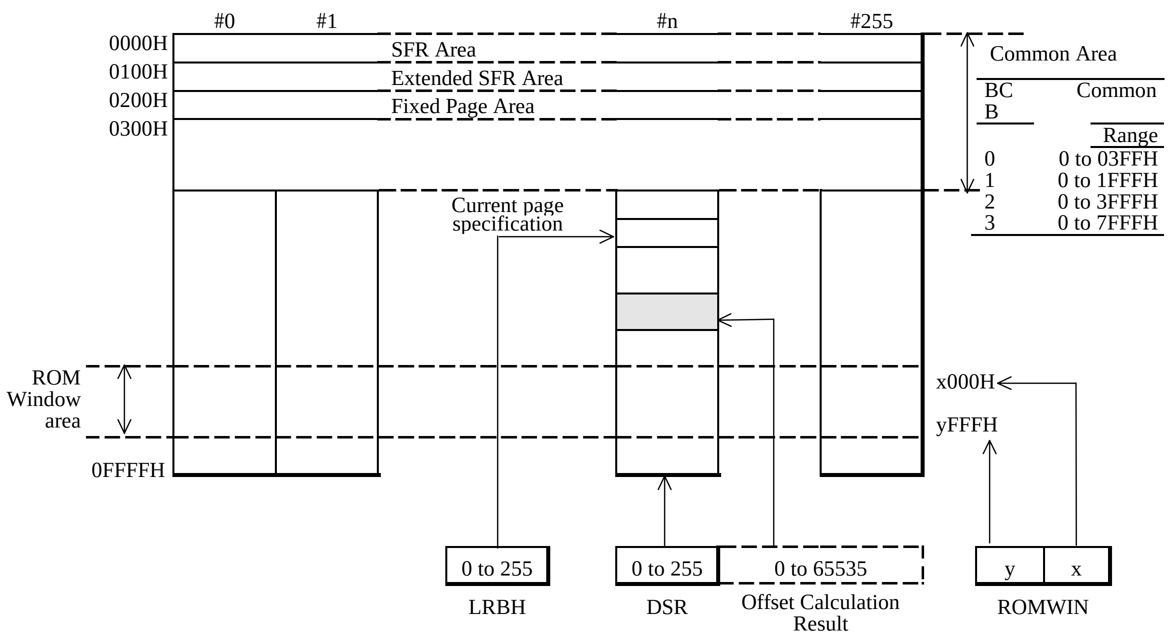
A single segment has offset addresses 0 to 0FFFFH. Address calculations to determine the addressing of objects are performed with 16-bit offset addresses; overflows and underflows are ignored. Therefore, addressing alone will not change the DSR.

The nX-8/500S provides several special areas in data memory space to raise coding efficiency. These areas include special pages, such as the SFR, fixed, and current page, which allow addresses to be specified as one-byte offsets within the page. There is also an SBA area, which provides very efficient code for the instructions SB, RB, JBS, and JBR. If the programmer defines variables with consideration to the location of data, then the assembler will select the optimal addressing for data accesses.

Applications that use multiple data segments may need to exchange data between segments. To enable this exchange, nX-8/500S has a common area starting from address 0 in data memory. The SFR area, extended SFR area, and fixed page area always reside in the common area.

Local registers and pointing registers are located in data memory space. These registers can also be accessed with address specifications.

■Overview of data memory space



1-2-2-2-1. SFR Area

The nX-8/500S maps special function registers (SFR) for controlling peripheral functions to memory (memory-mapped I/O). The SFR area is the area to which SFR are assigned. It covers addresses 0 to 0FFH (page 0) in data memory space. This area resides in the common area, so it can always be accessed regardless of the value of DSR. The SFR area can be read and written with ordinary RAM addressing, and it also allows SFR addressing for better coding efficiency.

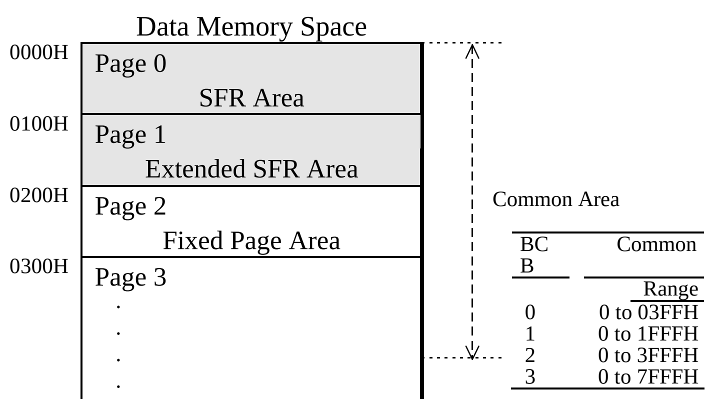
Special function registers include word registers, byte registers, bit registers, and combinations thereof. They also include read/write registers, read-only registers, and write-only registers. Many important special-purpose registers, such as the accumulator (A) and program status word (PSW), are also assigned to the SFR area. There are addresses in the SFR area to which no SFR is assigned, but the results of reading or writing these addresses are not guaranteed.

For details on the SFR and SFR functions in your target device, refer to the user's manual of that device.

1-2-2-2-2. Extended SFR Area

The 256 bytes at addresses 100H to 1FFH (page 1) in data memory space are called the extended SFR area. Like the SFR area, the extended SFR area is assigned SFR registers for controlling peripheral functions. Except that it cannot be used with SFR addressing, it is identical to the SFR area described above.

■SFR area and extended SFR area



1-2-2-2-3. Fixed Page

The 256 bytes at addresses 200H to 2FFH (page 2) in data memory space are called the fixed page. The fixed page is for efficient fixed page addressing (fix Dadr). The fixed page can also be read and written with ordinary RAM addressing. Along with the SFR area and extended SFR area, the fixed page area resides in the common area, so it can be accessed regardless of the value of DSR.

1-2-2-2-3-1. Area Available For Pointing Registers

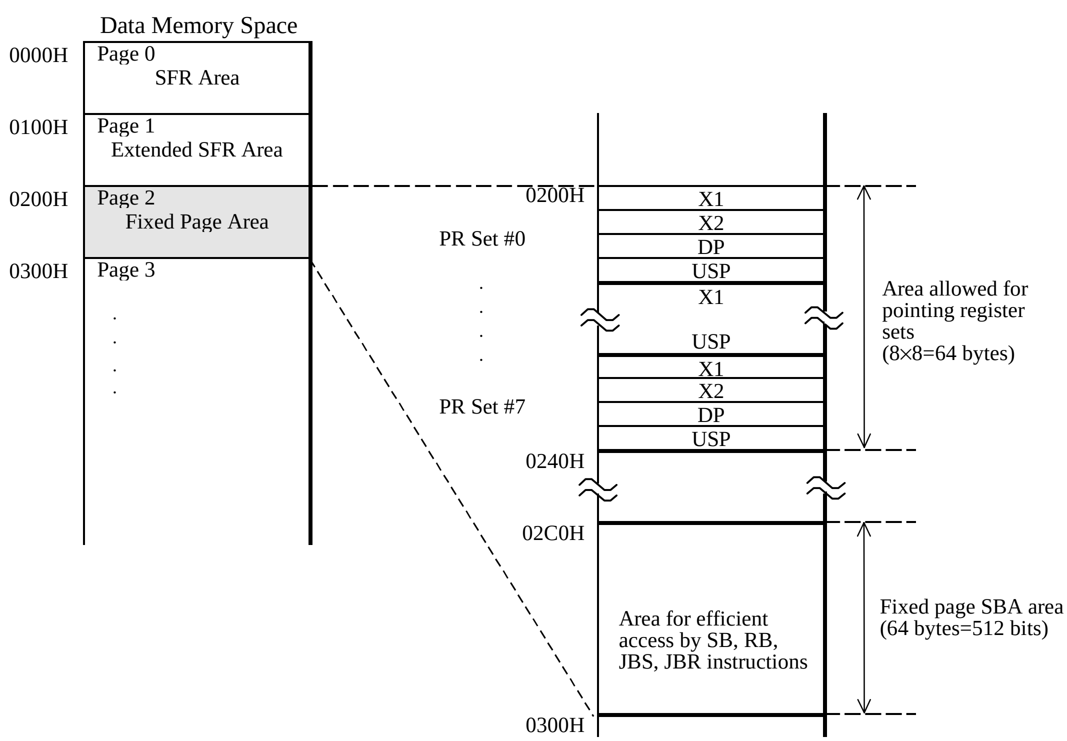
The 64 bytes starting from address 200H in the fixed page area are allocated eight pointing register sets. The pointing register area can also be used as ordinary memory when it is not being used as pointing registers.

The pointing register sets overlap the first eight local register sets, which also start at address 200H.

1-2-2-2-3-2. Fixed Page SBA Area

The 64 bytes at addresses 2C0H to 2FFH in the fixed page area are called the fixed page SBA area. As for the current page SBA area, the four instructions SB, RB, JBS, and JBR have efficient instruction codes for accessing the 512 bits in the fixed page SFR area.

■Fixed Page Configuration



1-2-2-2-4. Current Page

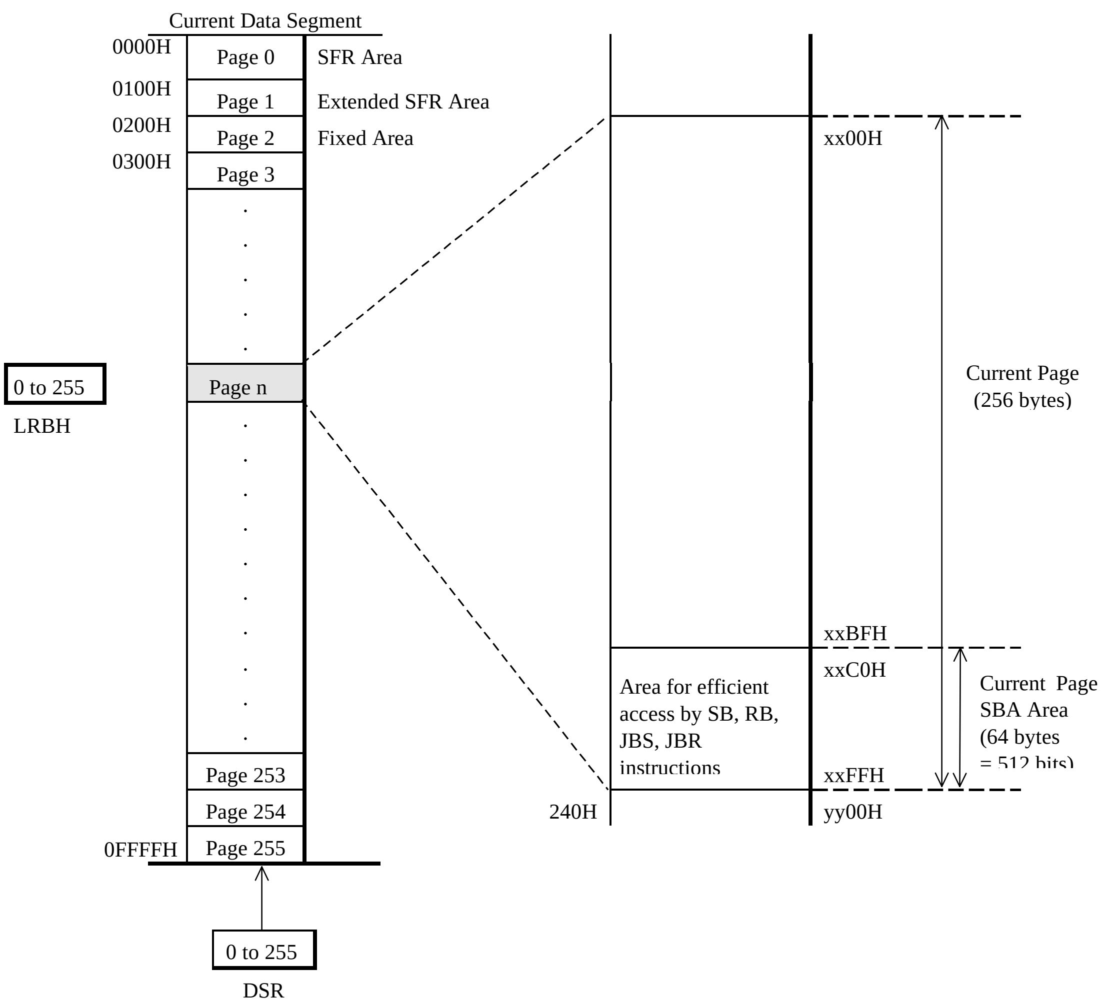
Each segment of data memory space is divided into 256 pages.　Each page is 256 bytes starting from a 256-byte boundary (xx00H).　Addresses for one page in the current data segment can be specified as 1-byte offsets within the page.　This page is called the current page.　The location of the current page in the data segment is specified by the high byte of the local register base (LRBH).

The nX-8/500S provides current page addressing (off Dadr or＼Dadr) and current page SBA area addressing (sbaoff Badr or＼Badr) for the 256 bytes of the current page.

1-2-2-2-4-1. Current Page SBA Area

The 64 bytes at addresses xxC0H to xxFFH in the current page are called the current page SBA area.　As for the fixed page SBA area, the four instructions SB, RB, JBS, and JBR have efficient instruction codes for accessing the 512 bits in the current page SFR area.

■Current Page Configuration

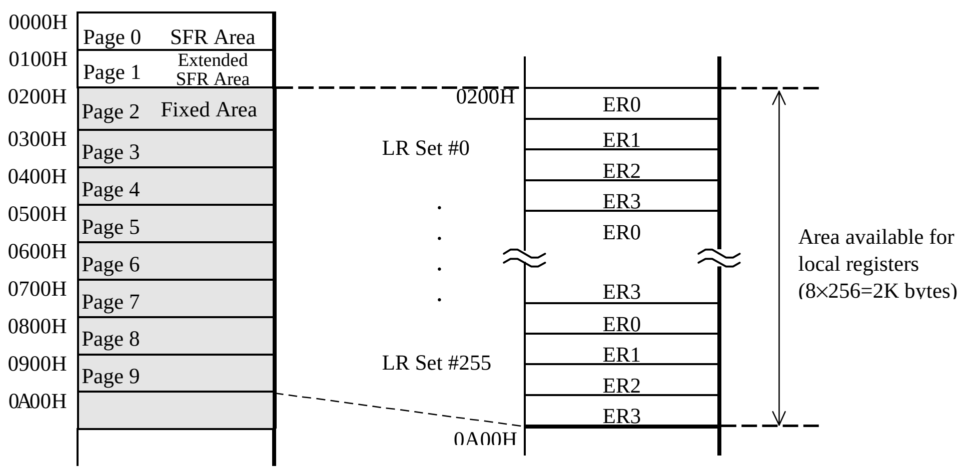


1-2-2-2-5. Area Available For Local Registers

The 256 local register sets are allocated to the 2,048 bytes starting from address 200H in data segment #0. Any one set can be used as local registers by setting LRBL. The local register area can also be used as ordinary memory when it is not used as local registers.

The first eight local register sets overlap the pointing register sets, which also start at address 200H.

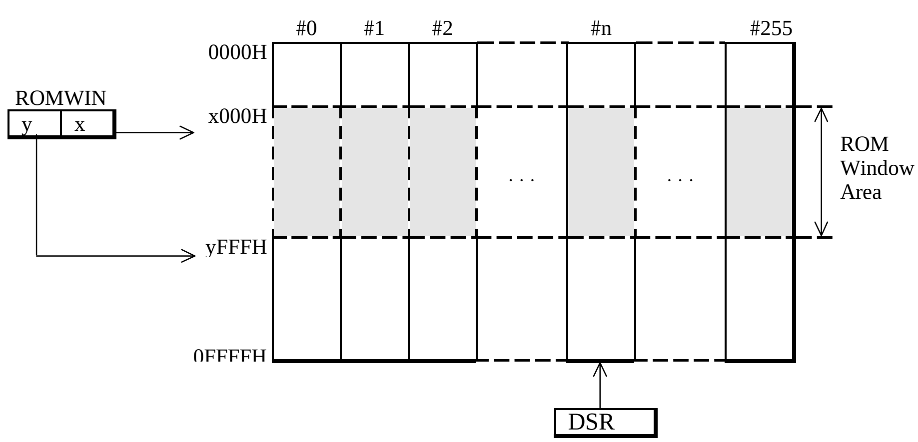
■Local register area in data memory space



1-2-2-2-6. ROM Window Area In Data Memory Space

For accessing data in the table segment specified by TSR using RAM addressing (ROM window addressing), an area exists as a window opened in the data segment. By opening a window in an area not mapped to internal data memory, a program can read table data at the same address values. The range of the ROM window area is set by the ROM window control register (ROMWIN).

■ROM window area in data memory space



1-2-2-2-7. Common Area

The nX-8/500S provides a common area in data memory space for exchanging data between segments. The common area is common to all segments. It is located in low memory of each segment starting from offset address 0. The range of the common area is set by the value in the BCB field of the PSW. The relation between the BCB value and the common area selected is as shown below.

No.	BCB value 1	0	Common Area Range
0	0	0	0 to 03FFH
1	0	1	0 to 1FFFH
2	1	0	0 to 3FFFH
3	1	1	0 to 7FFFH

The common area always includes the SFR area, extended SFR area, and fixed page, so they can be accessed regardless of the value of DSR.

1-2-2-2-8. Other Memory

1-2-2-2-8-1. EEPROM Area

Internal EEPROM may be allocated to addresses 4000H to 6000H of data segment #0. Refer to the user's manual of a target device that has EEPROM for its programming control functions.

1-2-2-2-8-2. Dual Port RAM Area

Internal dual port RAM may be allocated to addresses 6000H to 8000H of data segment #0. Refer to the user's manual of a target device that has dual port RAM for its control functions.

1-2-2-2-9. Internal And External Data Memory Areas

There is no logical difference between programming for internal data memory and external data memory. Use the linker to optimally assign data to internal data memory areas of the target device and external data memory areas mounted in the target system.

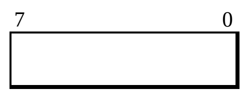
The size of internal data memory differs depending on the device. Refer to the user's manual of your target device for details.

1-3. Data Types

This section describes the types of data that can be used with nX-8/500S instructions.

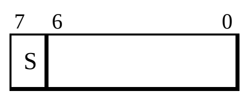
Unsigned byte

The unsigned byte data type can be handled by byte instructions. Its range is 0 to 255. When arithmetic calculations on unsigned byte data cause overflow or underflow from the 0 to 255 range, the carry (CY) will be set to 1 and the result will be the value of the modulo 256 operation. Logical calculations on unsigned byte data are performed on each bit. Bit positions in one byte of data are assigned numbers such that the MSB is bit 7 and the LSB is bit 0.



Signed byte

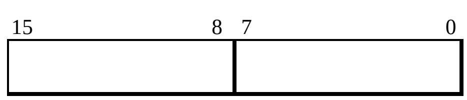
The signed byte data type can be handled by byte instructions. It is expressed as 2's complement, with the most significant bit recognized as the sign bit. Its range is -128 to 127. When arithmetic calculations on signed byte data cause overflow or underflow from the -128 to 127 range, the overflow flag (OV) will be set to 1.



Unsigned word

The unsigned word data type can be handled by word instructions. Its range is 0 to 65535. The low byte (bits 7-0) of a word is allocated to the lower address in memory, while the high byte (bits 15-8) is allocated to the higher address. In data memory space the lower address at which the low byte is located will always be an even address in order to keep word boundaries. In code memory space this restriction does not exist. The address of word data will be the address of that word's low byte.

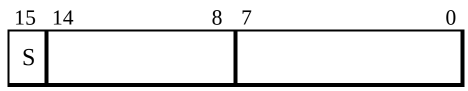
When arithmetic calculations on unsigned word data cause overflow or underflow from the 0 to 65535 range, the carry (CY) will be set to 1 and the result will be the value of a modulo 65536 operation. Logical calculations on unsigned word data are performed on each bit. Bit positions in one word of data are assigned numbers such that the MSB is bit 15 and the LSB is bit 0.



Signed word

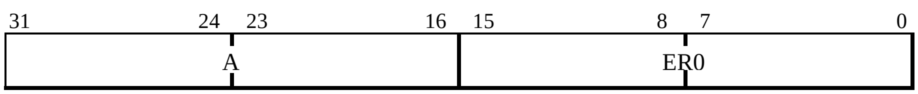
The signed word data type can be handled by word instructions. It is expressed as 2's complement, with the most significant bit recognized as the sign bit. Its range is –32768 to +32767. The low byte (bits 7-0) of a word is allocated to the lower address in memory, while the high byte (bits 15-8) is allocated to the higher address. In data memory space the lower address at which the low byte is located will always be an even address in order to keep word boundaries. In code memory space this restriction does not exist. The address of word data will be the address of that word's low byte.

When arithmetic calculations on signed word data cause overflow or underflow from the –32768 to +32767 range, the overflow flag (OV) will be set to 1.



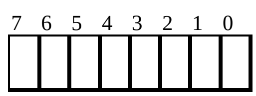
Unsigned long word

The unsigned long word data type is used for multiplication (MUL instruction) and division (DIV and DIVQ instruction). Its range is 0 to 4,294,967,295. It expresses the product of a 16-bit×16-bit multiplication or the dividend and quotient of a 32-bit / 16-bit division.



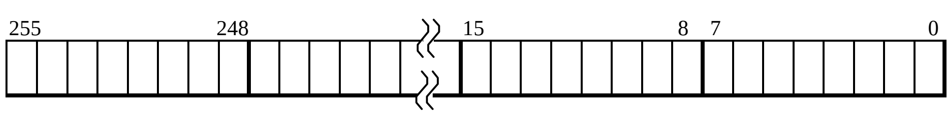
Bit

The bit data type is accessed by bit manipulation instructions. It takes the values 0 and 1. It can express all bits in memory and bit-type registers. Bit data is specified in operands by appending a bit position specifier 0 to 7 to addressing of a byte-type register or memory. Moves, logical calculations, and bit test and jump operations can be performed on accessed bits.



Bit array

The bit array data type is handled by bit manipulation instructions with register indirect bit specifications (MBR). A bit array is a maximum 256 bits (32 bytes) starting from a byte boundary in memory specified as the instruction operand. Each element of the array is bit data. The array is allocated to memory as bytes starting from bit 0 in 8-bit increments in the direction of higher addresses. The bits in each byte are allocated in sequence with the smallest specifier assigned to the LSB and the largest specifier assigned to the MSB.



1-4. Address Allocation

Address allocation in memory is performed in both byte units and bit units.

Byte addresses are individual addresses allocated to all bytes in memory. A 64K-byte space is allocated 65535 addresses from a low address of 0 to a high address of 0FFFFH. The range of complete addresses including segment addresses is 0:0 to 0FFH:0FFFFH.

Bit addresses are individual addresses allocated to all bits in memory. A 64K-byte space is allocated 524288 addresses from a low address of 0 to a high address of 7FFFFH. The bit addresses in each byte are assigned such that the lowest address is the LSB and the highest address is the MSB. If a byte is at byte address addr, then the bit address of its LSB is addr×8. The range of complete addresses including segment addresses is 0:0 to 0FFH:7FFFFH.

The nX-8/500S has two independent spaces, program memory space and data memory space. Each of these are allocated both byte addresses and bit addresses as explained above. Bit addresses in program memory space correspond to bits in the table segment opened through the ROM window.

■Byte addresses and bit addresses

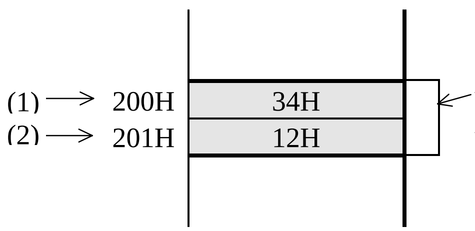
Byte address \ Bit position	7	6	5	4	3	2	1	0
0000H	7H	6H	5H	4H	3H	2H	1H	0H
0001H	0FH	0EH	0DH	0CH	0BH	0AH	9H	8H
0002H	17H	16H	15H	14H	13H	12H	11H	10H
0003H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
0004H	27H	26H	25H	24H	23H	22H	21H	20H
0005H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
		36H	35H	34H	33H	32H	31H	30H
0FFFCH	7FFE7H	7FFE6H			3BH	3AH	39H	38H
0FFFDH	7FFEFH	7FFEEH	7FFEDH	7FFECH	7FFEBH			7FFE8H
0FFFEH	7FFF7H	7FFF6H	7FFF5H	7FFF4H	7FFF3H	7FFF2H	7FFF1H	7FFF0H
0FFFFH	7FFFFH	7FFFEH	7FFFDH	7FFFCH	7FFFBH	7FFFAH	7FFF9H	7FFF8H

1-5. Word Boundaries

Data memory of the nX-8/500S has word boundaries (word alignment). Word boundaries restrict word memory accesses to even addresses. For the nX-8/500S, a word memory access to an odd address will actually access the word data located at the next lower address. In other words, word data that extends across a word boundary cannot be read. Looked at another way, word data in data memory space must be arranged to follow word boundaries.

■Word boundaries in data memory space

```
L    A, 200H    ; ①A←1234H (AH←contents of address 201H, AL←contents of address 200H)
L    A, 201H    ; ②A←1234H (AH←contents of address 201H, AL←contents of address 200H)
```



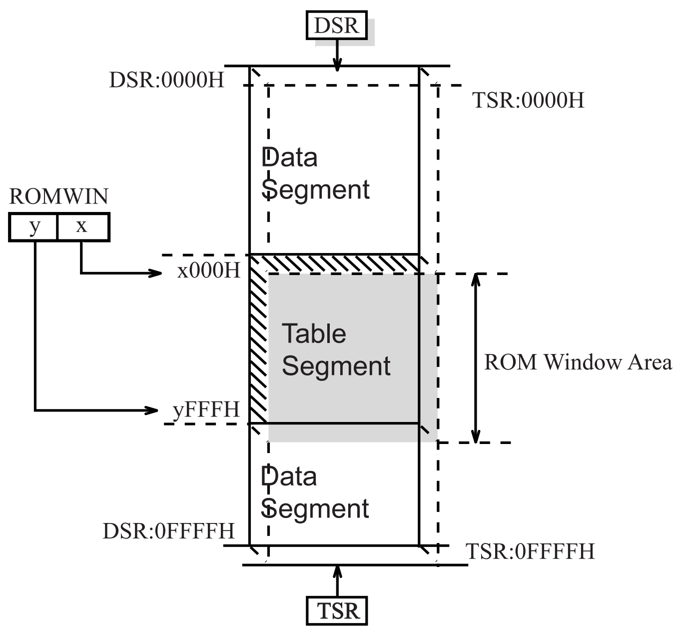
When memory at either 200H or 201H is read as a word, these contents (1234H) will be read, as explained above.

Word boundaries do not exist in program memory space. They also do not exist in program memory space accessed through the ROM window (table segment). This operational difference between data memory space and program memory space arises due to differences in address generation hardware.

1-6. ROM Window Function

Compared to the addressing modes and instructions available for accessing data memory, those available for accessing the table segment in program memory space are severely restricted. There are only four instructions: LC, LCB, CMPC, and CMPCB. To get around this restriction, the nX-8/500S provides a ROM window function.

The ROM window functions opens a window in an area that is not allocated to internal memory of the data segment, and then views the table segment through that window. When the ROM window is opened, the table segment can be accessed by using RAM addressing at the same offset address to read the data segment. The ROM window can only be accessed by reads. The results of a write operation to the ROM window are not guaranteed.



In order to open the ROM window, its lower and upper addresses must be set in the ROM window control register (ROMWIN). ROMWIN is an 8-bit register allocated in SFR space. The lower 4 bits specify the starting address of the ROM window, and the upper 4 bits specify the ending address. The starting address will be $ROMWIN_{3\text{-}0} \times 1000H$, and the ending address will be $ROMWIN_{7\text{-}4} \times 1000H + 0FFFH$. For example, if 71H is written to ROMWIN, then the ROM window will be 1000H to 7FFFH. If the value written to the lower 4 bits is 0, then the ROM window function will not operate.

ROMWIN may be written only once after reset. Second and later writes will be ignored. Immediately after reset, the value of ROMWIN will be 0, so the ROM window function will not operate. To use the ROM window function, it is recommended that you open the ROM window soon after reset.

1-7. Memory Models

The nX-8/500S implements the concept of hardware memory models. Depending on the memory model, accessible memory size, interrupt and corresponding RTI instruction operation, and VCAL instruction operation will differ. The hardware can also check whether or not FJ and FCAL instructions with far code addressing and corresponding FRT instructions can be executed.

The memory model chooses the maximum accessible memory size from two possibilities: 64K bytes and 64M bytes. The combination of both choices for code memory space and data memory space gives four memory models, as shown in the table below. When accessing a 16M-byte space, segment addresses will be valid for that space. Writes are permitted to segment registers that are not use by the specified memory model, but these values will not be used by the hardware to specify segments.

Under the medium or large model, with 16M-bytes of code memory space, interrupts and VCAL instructions will push both the PC and CSR on the stack. Then when an RTI instruction returns from processing an interrupt, the CSR will also be popped from the stack. Also, the FRT instruction must be used to return from a subroutine called by a VCAL instruction.

Under the small or compact model, with 64K-bytes of code memory space, FJ, FCAL, and FRT instructions will cause an op-code trap. The microcontroller will resume execution from the vector address corresponding to resets when an op-code trap occurs.

Devices that have only the small model do not have a configuration for setting the memory model. Only the first memory model setting made after reset is valid. All devices assume the small model by default immediately after reset. Refer to the user's manual of your target device regarding how to set the memory model.

The above information is summarized in the table below.

Model	Max. Memory		Segment Register			Interrupts	Instructions	
	Code	Data	CSR	TSR	DSR		VCAL	FJ,FCAL,FRT
Small	64K	64K	-	-	-	Near	Near	Op-code trap
Compact	64K	16M	-	-	Valid	Near	Near	Op-code trap
Medium	16M	64K	Valid	Valid	-	Far	Far	Executable
Large	16M	16M	Valid	Valid	Valid	Far	Far	Executable

1-8.　Data Descriptor (DD)

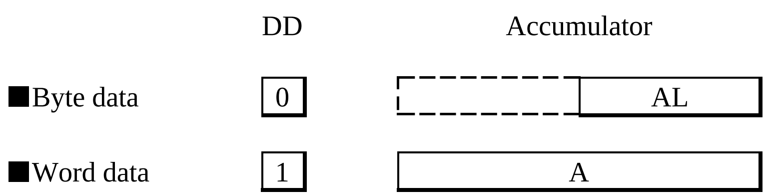
The nX-8/500S has a special flag called the data descriptor (DD).　The programmer must pay attention to the DD flag when the type of data being handled changes during the flow of the program.

This section first describes the meaning and use of DD, and then lists the instructions that affect DD and the instructions affected by DD.

The dependence of each instruction on DD is shown in the "Flags" section of each instruction's description in Chapter 3, under the heading "Flags affecting instruction execution."　The "Description" section will also include the statement, "Execution of this instruction is limited to when DD is 0/1."

1-8-1.　Description And Use OF DD

The data descriptor (DD) is allocated to bit 12 of the PSW.　It is a flag that indicates the type of data in the accumulator (A).　When DD is 0, it indicates byte data.　When DD is 1, it indicates word data.



The type of calculations that use the accumulator (A) is determined by DD.　Instructions that affect this flag are accumulator load instructions, clear instructions, and type conversion instructions, as well as instructions that directly set and reset the flag.　Instructions affected by this flag are basically those that leave calculation results in A and those that store the contents of A.

In general, a program is made up of blocks that load data of some type in the accumulator, perform several calculations of that type, and then store the results in memory.　If the type of data to be loaded in A and then calculated is determined once, then further calculations and stores should be performed with that same type.　In such cases, there is no need for the instruction codes of calculation and store instructions to contain information about data type.　This has allowed the nX-8/500S to efficiently increase the number of instructions implemented.

The flag that preserves the data type information determined by the accumulator load is DD. Thus, instructions that load word data to the accumulator set DD to 1.　Instructions that load byte data to the accumulator reset DD to 0.

```
L       A, #1234H       ; Sets DD to 1.
LB      A, #12H         ; Resets DD to 0.
```

Further calculations performed on the accumulator will be affected by DD. In the following example, the word data at address VAR is added to A, and the result is stored as a word in memory at VAR2.

```
...
L       A, #1234H      ; A←1234H              Sets DD to 1.
ADD     A,VAR          ; A←A+VAR              Executed when DD is 1.
ST      A,VAR2         ; A→VAR2               Executed when DD is 1.
...
```

The following example shows the handling of byte data. Byte data at address VAR is added to AL, and the result is stored as a byte in memory at VAR2.

```
...
LB      A, #12H        ; AL←12H               Sets DD to 0.
ADDB    A,VAR          ; AL←AL+VAR            Executed when DD is 0.
STB     A,VAR2         ; AL→VAR2              Executed when DD is 0.
...
```

In these two examples, ADD and ADDB actually have identical instruction codes. ST and STB also have identical instruction codes. The difference is only the value of DD. In the following example, the ADDB and STB mnemonics are expressed for byte instruction operation (or so the programmer hopes), but they will actually operate as word instructions.

```
...
L       A, #1234H      ; A←1234H              Sets DD to 1.
ADDB    A,VAR          ; A←A+VAR              Byte instruction operates as word.
STB     A,VAR2         ; A→VAR2               Byte instruction operates as word.
...
```

If the programmer truly wants ADDB and STB to operate as byte instructions, then he needs to change the value of DD as shown next.

```
...
L       A, #1234H      ; A←1234H              Sets DD to 1.
RDD                    ; DD←0                 Calculate with byte data.
ADDB    A,VAR          ; AL←AL+VAR            Operates as byte.
STB     A,VAR2         ; AL→VAR2              Operates as byte.
...
```

Conversely, to calculate with word data after loading byte data, the programmer can use the SDD instruction to set DD to 1, or he can use the sign-extension type conversion instruction as shown below.

```
...
LB      A, VAR         ; A←VAR                    Sets DD to 0.
EXTND                  ; A←(sign extension)AL     Signed type conversion. Sets DD to 1.
ADD     A,VAR2         ; A←A+VAR2                 Operates as word.
ST      A,VAR3         ; A→VAR3                   Operates as word.
...
```

The programmer must look closely at whether DD must be explicitly set or reset at points where calculations change between byte data and word data. The assembler provides the USING DATA directive in order to detect when DD is inappropriate for instructions that reference DD.

1-8-2. Instructions That Change DD

1-8-2-1. Instructions That Change DD As Part Of Their Function

In accordance with the philosophy explained in Section 1-8, "Description And Use Of DD," instructions that move data to the accumulator or sign-extend the accumulator will determine the accumulator's data type. Also, the programmer needs instructions that set and reset DD. The nX-8/500S instructions that change DD as part of their function are listed below.

■Instructions that set DD to 1

Mnemonic	Operand	CZSVHD D	Function
L	A,obj	. Z. . . 1 .	A ← obj, DD ← 1
CLR	A	. 1. . . 1 .	A ← 0, DD ← 1
SDD		. . . . . 1 .	DD ← 1
EXTND		. . S. . 1 .	$A_{15-7} \leftarrow A_7$, DD ← 1

■Instructions that reset DD to 0

Mnemonic	Operand	CZSVHD D	Function
LB	A,obj	. Z. . . 0 .	AL ← obj, DD ← 0
CLRB	A	. 1. . . 0 .	AL ← 0, DD ← 0
RDD		. . . . . 0 .	DD ← 0
BRK		000000 .	RESET, PC ←(Vector-table 0002H)

DD is always changed when these instructions are executed. Flag changes are clearly shown to be 0 or 1 under the “Flags” heading for each instruction of chapter 3 in the manual.

1-8-2-2. Other Instructions That Change DD

The PSW is allocated to SFR space. DD can be written by accessing PSW and DD using byte addresses and bit addresses.

■Example■Other instructions that change DD

```
MOV    APSW,#0    ; Write PSW using byte address.
MOVB   PSWH,#0    ; Write PSWH using byte address.
SB     DD         ; Set DD to 1 using bit address.
```

Depending on the operation, these instructions may or may not change DD. Writes to DD in these cases are not clarified in this manual's descriptions of flag changes ("Flags" heading for each instruction of chapter 3). These instruction are considered to just happen to have PSW as their object.

1-8-3. Instructions Affected By DD

The instructions that operate in accordance with the data type of the accumulator, as described in Section 1-8-1, "Description And Use Of DD," are shown in the table below. These are nearly all the instructions that have A as their first operand.

■Instruction executed when DD is 1 (word)

Mnemonic	Operand	CZSVHDD	Function
ST	A,obj	 1	obj ← A
FILL	A	 1	A←0FFFFH
XCHG	A, obj	 1	A←→obj
SLL	A, width	C. 1	C←[$_{15}$ A $_{0}$]←0
SRL			0→[$_{15}$ A $_{0}$]→C
SRA			A15[$_{15}$ A $_{0}$]→C
ROL			C←[$_{15}$ A $_{0}$]←C
ROR			C→[$_{15}$ A $_{0}$] →C
INC	A	. ZSVH. 1	A←A+1
DEC			A←A-1
SQR	A	. Z. . . . 1	<A,ER0> ← A × A
ADD	A,obj	CZSVH. 1	A←A+obj
ADC			A←A+obj+C
SUB			A ← A-obj
SBC			A ← A-obj-C
CMP			A-obj
NEG	A	CZSVH. 1	A ← -A
AND	A,obj	. ZS. . . 1	A←A ∩ obj
OR			A ← A ∪ obj
XOR			A ← A $\overline{\cup}$ obj
TJZ	A, radr		if A=0 then PC←radr
TJNZ			if A≠0 then PC←radr

■Instructions executed when DD is 0 (byte)

Mnemonic	Operand	CZSVHDD	Function
STB	A,obj	 0	obj ← AL
FILLB	A	 0	AL←0FFH
XCHGB	A, obj	 0	AL←→obj
SLLB	A, width	C. 0	C←[$_{7}$ AL $_{0}$]←0
SRLB			0→[$_{7}$ AL $_{0}$]→C
SRAB			A7→[$_{7}$ AL $_{0}$]→C
ROLB			C←[$_{7}$ AL $_{0}$]←C
RORB			C→[$_{7}$ AL $_{0}$]→C
INCB	A	. ZSVH. 0	AL←AL+1
DECB			AL←AL-1
SQRB	A	. Z. . . . 0	A ← AL × AL
ADDB	A,obj	CZSVH. 0	AL←AL+obj
ADCB			AL←AL+obj+C
SUBB			AL ← AL-obj
SBCB			AL ← AL-obj-C
CMPB			AL-obj
NEGB	A	CZSVH. 0	AL ← -AL
ANDB	A,obj	. ZS. . . 0	AL←AL ∩ obj
ORB			AL ← AL ∪ obj
XORB			AL ← AL $\overline{\cup}$ obj
TJZB	A, radr	 0	if AL=0 then PC←radr
TJNZB			if AL≠0 then PC←radr

1-8-4. Pre-Fetched Instructions And DD

If DD is changed using an instruction described in Section 1-8-2-2, "Other Instructions That Change DD," (for example, if DD is changed by performing a write with the address specification as PSW in SFR space), and if the next instruction is one that is affected by DD, then a NOP must be inserted before that next instruction.

■Example■NOP insertion

```
...
ANDB   PSWH,#05H      ; DD reset to 0 along with C, Z, HC, S, and OV.
NOP                   ; NOP is needed.
ADDB   A,#12H         ; Instruction is affected by DD.
...
```

Normal programming does not have many instances of changing DD using instructions described in Section 1-8-2-2, "Other Instructions That Change DD."　They are limited to cases like the example above, where other flag types are to be changed simultaneously.　Typically this would be to set the PSW to an initial value with a single instruction.

The reason that a NOP is needed is as follows.　Before execution of one instruction is finished, the nX-8/500S starts to fetch and decode the next instruction.　The value of DD is fetched along with the instruction code at this point, so if the previous instruction does not change DD until its last state, then the (final) value of DD when the previous instruction finishes executing will not be the same as the value of DD that is fetched.　If the next instruction is affected by the value of DD, then it will operate based on the DD value that was fetched.　If a NOP is inserted before that instruction, then it will operate based on the correctly changed value of DD.

How this affects a program is explained below.　In the next example, the instruction immediately after an RB instruction is an ADD instruction that references DD.　The programmer intends to load the immediate value 1234H in A, and then add the byte data at address 300H to AL. However, the ADDB instruction will operate not on byte data, but actually as an ADD instruction for word data.　As a result, the word data at address 300H will be added to the accumulator.

```
...
L      A, #1234H      ; A←1234H, DD←1
RB     DD             ; DD←0
ADDB   A, 300H        ; Operates as word instruction "ADD    A,300H"
...
```

In order to avoid this, use the RDD instruction instead of an RB instruction.　The RDD instruction was created specifically for manipulating DD.

```
...
L      A, #1234H      ; A←1234H, DD←1
RDD                   ; DD←0
ADDB   A, 300H        ; Operates as byte instruction as expected.
...
```

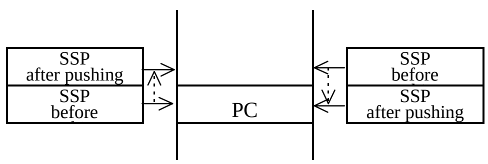
Alternatively, insert an NOP before the instruction that references DD.

```
...
L       A, #1234H    ; A←1234H, DD←1
RB      DD           ; DD←0
NOP                  ; Next instruction fetched while NOP execution.
ADDB    A, 300H      ; Operates as byte instruction as expected.
...
```

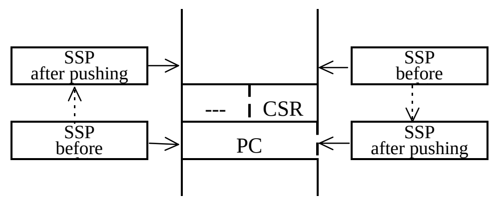
1-9. Changing The Stack

This section summarizes how the stack is changed by instructions and interrupts. Refer to Chapter 3 for details of each instruction. For pushing/popping the register sets CR, ER, and PR, this section illustrates only the case where the entire sets are pushed/popped at once. The sequence for pushing/popping the entire register set at once is identical to selecting the individual registers to be pushed/popped. The images shown are the locations in memory of registers when ER and PR are pushed as register sets, and when an interrupt pushes CR as a register set.

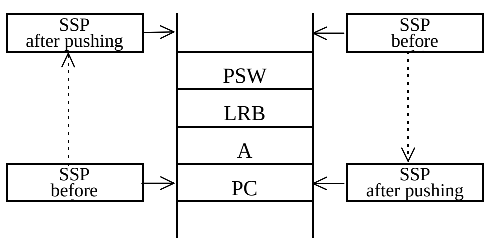
■ Push
CAL, SCAL, ACAL instructions
VCAL instruction (small/compact model)
Pop
RT instruction



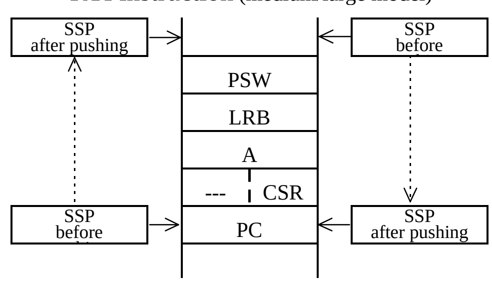
■ Push
FCAL instruction
VCAL instruction (medium/large model)
Pop
FRT instruction



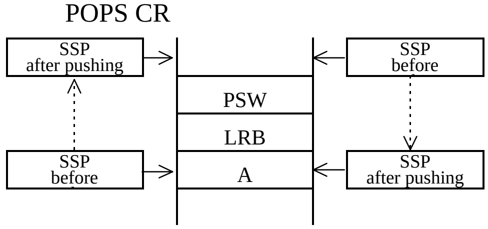
■ Push
Interrupt (small/compact model)
Pop
RTI instruction (small/compact model)



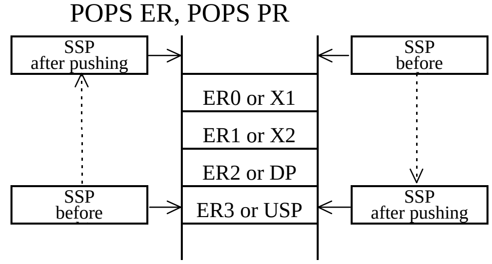
■ Push
Interrupt (medium/large model)
Pop
RTI instruction (medium/large model)



■ Push
PUSHS CR
Pop
POPS CR



■ Push
PUSHS ER, PUSHS PR
Pop
POPS ER, POPS PR



1-10. Instruction Code Format

This section explains native instructions and composite instructions, a feature of nX-8/500S instruction code format.

1-10-1. Native Instructions And Composite Instructions

Instructions of the nX-8/500S are classified as native instructions or composite instructions based on the background of their instruction codes. Instructions that require high coding efficiency and processing efficiency are implemented as native instructions. Composite instructions consist of a prefix code and suffix code. The prefix code specifies the address being accessed. The suffix code mainly specifies the operation. This was one idea for implementing a wide variety of addressing modes. By having both native instructions and composite instructions, the nX-8/500S instruction set is able to be both efficient and easy to code with.

Native instructions are instructions with 1 to 4 bytes of code.

1 to 4 bytes

Composite instructions consist of a 1 to 3 byte address specification field (prefix) and a 1 to 3 byte operation specification field (suffix).

Prefix 1 to 3 bytes	Suffix 1 to 3 bytes

Prefixes can be word type or byte type. Word prefix codes and byte prefix codes are listed below. Suffixes of word instructions are combined with word prefixes. Suffixes of byte instructions and bit instructions are combined with byte prefixes.

■Word Prefixes

**	<word> Word Prefix Instruction Code			Cycle (Internal)
	1st	2nd	3rd	
A	BC			2
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP-]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

■Byte Prefixes

*	<byte> Byte Prefix Instruction Code			Cycle (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP-]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

The instruction code table given for each instruction in Chapter 3 is shown as a single table for native instructions. For composite instructions, the suffix code table corresponding to the word or byte prefix code table is shown. When the function of an instruction that can be combined from a prefix and suffix is identical to the function of a native instruction, the assembler will generate the native instruction code.

1-11. Microcontrollers That Use The nX-8/500S Core

The functional specifications of microcontrollers the use the nX-8/500S core differ on the following points.　Devices without these functions also exist.

- Peripheral circuits and allocation of registers in SFR space to control them.
- Accessible memory ranges and bit length of segment registers to control them (CSR, TSR, DSR).
- Permitted memory models and structures for setting them.
- Internal program memory range.
- Internal data memory types and ranges.
- Control methods for special internal data memory (EEPROM programming methods, etc.).
- Multiply-Addition function (MAC instruction) and its flag in the PSW (MAB).

When program memory space only has segment #0, the device has the following limitations.

- CSR and TSR do not exist.
- FJ, FCAL, and FRT instructions, which transfer execution across code segments, do not exist.
- Medium and large memory models cannot be specified.

When data memory space only has segment #0, the device has the following limitations.

- DSR does not exist.
- The concept of common memory across data segments does not apply.
- BCB in the PSW can be used as user flags.
- Compact and large memory models cannot be specified.

Refer to the user's manual of your target device for the functional specifications of the above items when you need this information to write programs.　However, when a function or structure does not exist for your target device, it might not be alluded to in the user's manual if it seemed unnecessary for explanations.

Chapter 2. Addressing Modes

This Chapter explains how to access registers and memory using nX-8/500S core instructions. The specific methods for these accesses are called addressing modes. This chapter describes the types, functions, and syntax of addressing modes.

2-1. Addressing Mode Types

The nX-8/500S core has two independent memory spaces: a data memory space and a program memory space. The nX-8/500S core addressing modes can be classified broadly to correspond to these spaces.

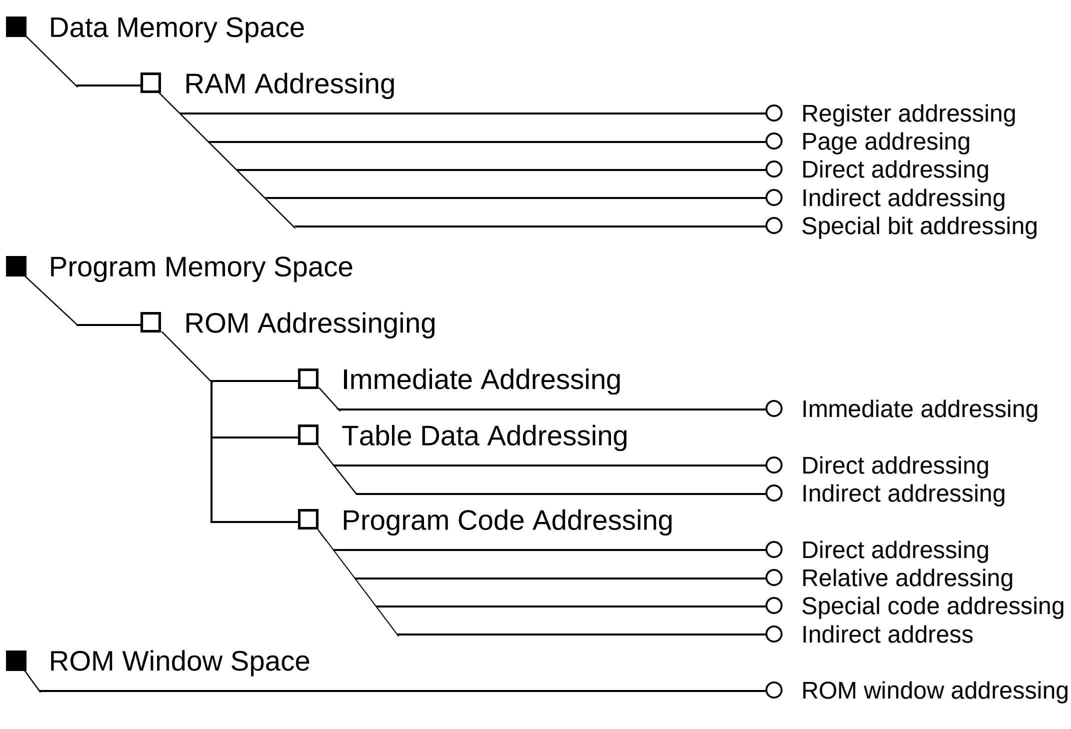
Data memory space is normally configured from read/write memory (RAM), so it is also called RAM space. Addressing to this space is called RAM addressing.

Program memory space is normal configured from read-only memory (ROM), so it is also called ROM space. Addressing to this space is called ROM addressing.

ROM addressing can be further divided into immediate addressing for accessing from instructions themselves, table data addressing for accessing data in ROM space, and program code addressing for accessing programs in ROM space.

In addition, the nX-8/500S core has a special addressing called ROM window addressing. This addressing mode accesses table data in ROM space using RAM addressing. It reads data in a table segment through a window in a data segment opened by the program.

The above addressing mode types and their addressing are summarized below.



RAM addressing, ROM addressing, and ROM window addressing are explained in order below.

2-2. RAM Addressing

RAM addressing is the addressing mode for addressing program variables in RAM space.

(1) Register Addressing

These various registers have dedicated addressing modes, and can also be addressed as data memory. These modes are classified as register addressing and RAM addressing.

(2) Page Addressing

(3) Direct Addressing

(4) Pinting register indirect addressing

(5) Special bit area (SBA) addressing

A Accumulator Addressing

Function

For word-type instructions, this addressing mode accesses the contents of the accumulator (A). For byte-type and bit-type instructions, this addressing mode accesses the contents of the low byte of the accumulator (AL).

Syntax

The instruction mnemonic determines whether the addressed object is the contents of the accumulator (A) or the contents of the low byte of the accumulator (AL).

Word format

L	**A** ,#1234H
ST	**A** ,VAR

Byte format

LB	**A** ,#12H
STB	**A** ,VAR

Bit format

MB	**C**, A.3
JBS	**A.3** ,LABEL

PSW / LRB / SSP

Control Register Addressing

Function

This addressing mode accesses the contents of registers.

Syntax

SSP	System stack pointer
LRB	Local register base
PSW	Program status word
PSWH	Program status word high byte
PSWL	Program status word low byte
C	Carry flag

The register name itself is coded as the operand.

Word format

FILL **SSP**
MOV **LRB** ,#401H
CLR **PSW**

Byte format

CLR **PSWH**
INC **PSWL**

Bit format

MB **C** ,BITVAR

X1 / X2 / DP / USP

Pointing Register Addressing

Function

This addressing mode accesses the contents of pointing registers.

In this addressing mode, value of System Control Base (SCB) field in the PSW specifies one of the 8 pointing registers (PR0 to PR7: every 8 bytes of 200H to 23FH in data memory.)

Syntax

X1	Index register 1
X2	Index register 2
DP	Data pointer
USP	User stack pointer
X1L	Index register 1 low byte
X2L	Index register 2 low byte
DPL	Data pointer low byte
DP*	Data pointer low byte
USPL	User stack pointer low byte

* Only for the "JRNZ DP,radr" instruction, provided for compatibility with nX-8/100-400.

The register name itself is coded as the operand.

Word format

L	A ,**X1**
ST	A ,**X2**
MOV	**DP** ,#2000H
CLR	**USP**

Byte format

DJNZ	**X1L** ,LOOP
DJNZ	**X2L** ,LOOP
DJNZ	**DPL** ,LOOP
DJNZ	**USPL** ,LOOP
JRNZ	**DP** ,LOOP

ER*n* / R*n* Local Register Addressing

Function

This addressing mode accesses the contents of local registers.

In this addressing mode, value of the low byte of Local Register Base (LRB) specifies one of 256 local registers (every 8 bytes of 200H to 9FFH of data memory.)

Syntax

ER0 to ER3	Extended local registers
R0 to R7	Local registers

The register name itself is coded as the operand.

Word format

```
L       A,ER0
MOV     ER2, ER1
CLR     ER3
```

Byte format

```
LB      A, R0
ADDB    R1, A
CMPB    R2, #12H
INCB    R3
RORB    R4
MOVB    R5, R6
```

Bit format

```
SB      R0.0
RB      R1.7
JBRS    R7.3, LABEL
```

sfr *Dadr* — SFR Page Addressing

Function

This addressing mode specifies an offset within the SFR page (data memory addresses 0-0FFH) with one byte in an instruction code. The specified address can be accessed as word, byte, or bit data.

Syntax

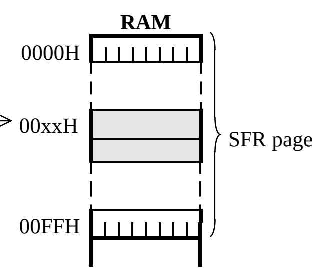
sfr *address_expression* *address_expression*

An expression with the "sfr" addressing specifier is coded as the operand. The "sfr" can be omitted, but SFR page addressing will result only when the assembler recognizes that the expression is an address in the SFR page.

Address symbols for each type of device are provided in the SFR. Usually these symbols are used for SFR accesses.

Word format

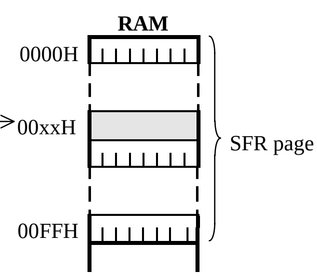
L A, **sfr P0**
L A, **P0**



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary). However, there may be exceptions depending on the SFR.

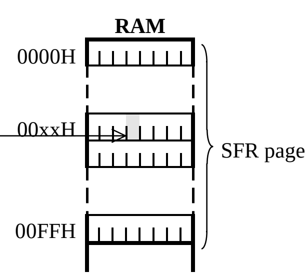
Byte format

LB A, **sfr P0**
LB A, **P0**



Bit format

SB **sfr P0.3**
SB **P0.3**



fix *Dadr* — Fixed Page Addressing

Function

This addressing mode specifies an offset within the fixed page (data memory addresses 200-2FFH) with one byte in an instruction code. The specified address can be accessed as word, byte, or bit data.

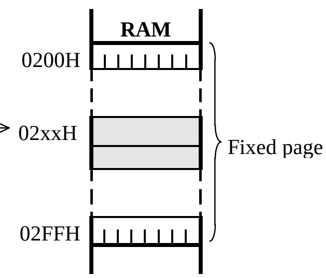
Syntax

fix *address_expression* *address_expression*

An expression with the "fix" addressing specifier is coded as the operand. The "fix" can be omitted, but fixed page addressing will result only when the assembler recognizes that the expression is an address in the fixed page.

Word format

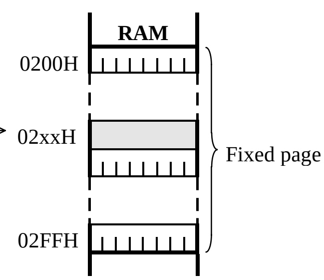
```
L    A, fix FIXVAR
L    A, FIXVAR
```



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

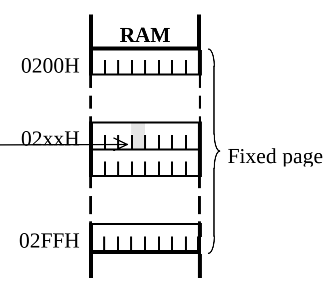
Byte format

```
LB   A, fix FIXVAR
LB   A, FIXVAR
```



Bit format

```
SB   fix FIXVAR.3
SB   FIXVAR.3
```



off *Dadr* — Current Page Addressing

Function

This addressing mode specifies an offset within the current page (data memory of one page of 256 as specified by the value of LRBH) with one byte in an instruction code. The specified address can be accessed as word, byte, or bit data.

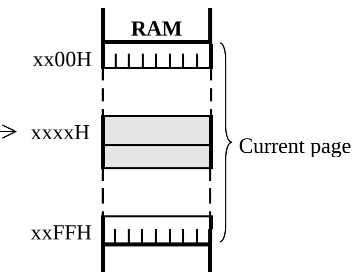
Syntax

off *address_expression* ＼*address_expression*

An expression with the "off" addressing specifier is coded as the operand. A backslash "＼" can be coded instead of "off", but the meaning is slightly different when accessing bit data in the SBA area (→sbaoff Badr).

Word format

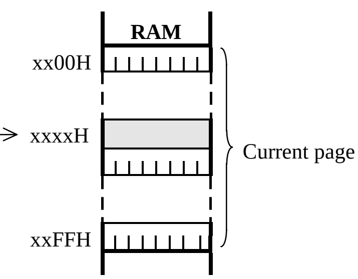
```
L     A, off VAR
L     A, ＼VAR
```



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

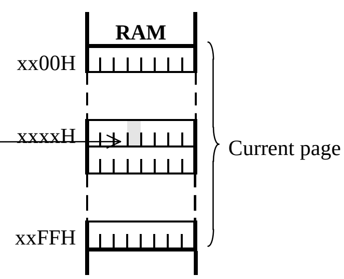
Byte format

```
LB    A, off VAR
LB    A, ＼VAR
```



Bit format

```
SB    off VAR.3
SB    ＼VAR.3
```



dir *Dadr* Direct Addressing

Function

This addressing mode specifies an address in the current physical segment of data memory (addresses 0-0FFFFH = 64K bytes) with two bytes in an instruction code. The specified address can be accessed as word, byte, or bit data.

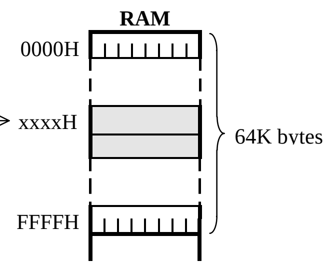
Syntax

dir *address_expression* *address_expression*

An expression with the "dir" addressing specifier is coded as the operand. The "dir" can be omitted, but the assembler may select SFR page addressing or fixed page addressing when the specified address is in the SFR page or fixed page.

Word format

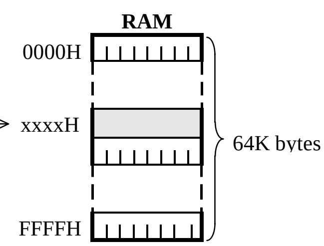
```
L    A, dir VAR
L    A, VAR
```



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

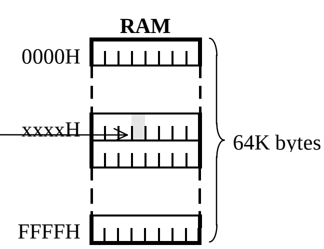
Byte format

```
LB   A, dir VAR
LB   A, VAR
```



Bit format

```
SB   dir VAR.3
SB   VAR.3
```



[DP] / [X1] Pointing Register Indirect Addressing

Function

This addressing mode specifies an address in the current physical segment of data memory (addresses 0-0FFFFH = 64K bytes) with the contents of a pointing register. The specified address can be accessed as word, byte, or bit data.

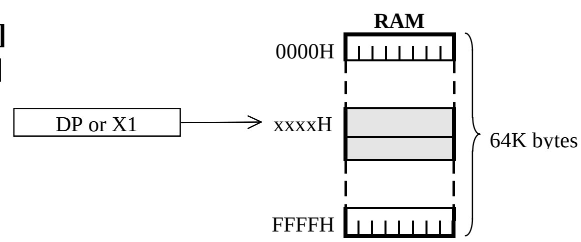
Syntax

[DP]	DP indirect addressing
[X1]	X1 indirect addressing

Only [DP] can be used with nX-8/100-400.

Word format

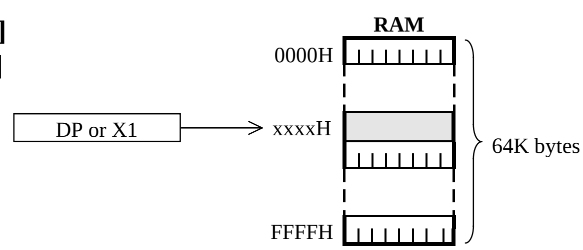
L A, **[DP]**
L A, **[X1]**



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

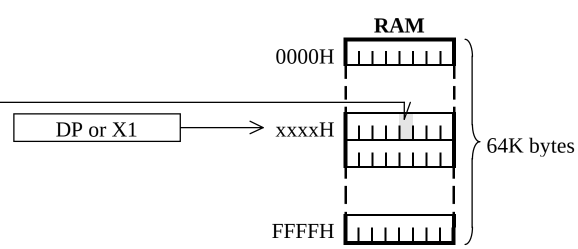
Byte format

LB A, **[DP]**
LB A, **[X1]**



Bit format

SB **[DP]**.3
RB **[X1]**.3



[DP+] DP Indirect Addressing With Post-Increment

Function

This addressing mode specifies an address in the current physical segment of data memory (addresses 0-0FFFFH = 64K bytes) with the contents of a pointing register. The specified address can be accessed as word, byte, or bit data.
After the address has been accessed, the contents of the pointing register are incremented. For word-type instructions, the contents are increased by 2. For byte-type and bit-type instructions, the contents are increased by 1.

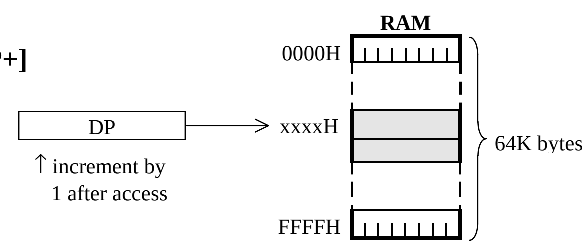
Syntax

[DP+]	DP indirect addressing with post-increment

This addressing mode does not exist for nX-8/100-400.

Word format

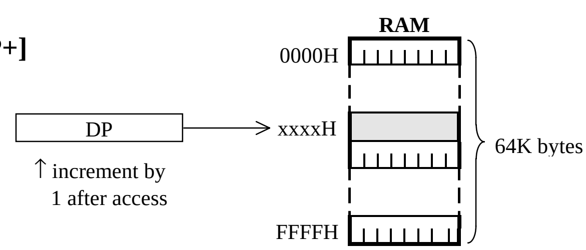
L A, **[DP+]**



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

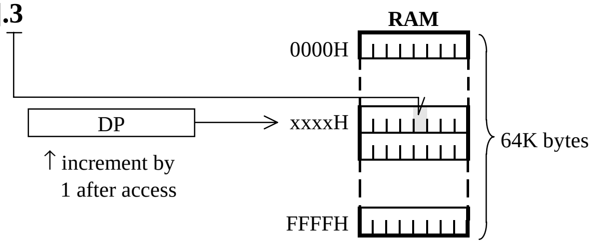
Byte format

LB A, **[DP+]**



Bit format

SB **[DP+].3**



[DP-] DP Indirect Addressing With Post-Decrement

Function

This addressing mode specifies an address in the current physical segment of data memory (addresses 0-0FFFH = 64K bytes) with the contents of a pointing register. The specified address can be accessed as word, byte, or bit data.

After the address has been accessed, the contents of the pointing register are decremented. For word-type instructions, the contents are reduced by 2. For byte-type and bit-type instructions, the contents are reduced by 1.

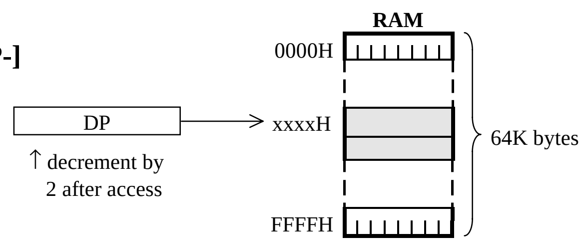
Syntax

[DP-]	DP indirect addressing with post-decrement

This addressing mode does not exist for nX-8/100-400.

Word format

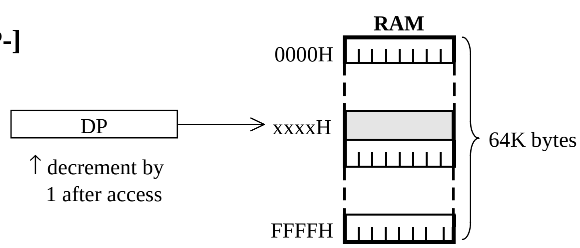
L A, **[DP-]**



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

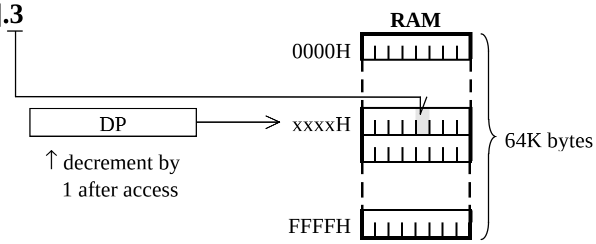
Byte format

LB A, **[DP-]**



Bit format

SB **[DP-].3**



n7 [DP] / *n7* [USP] DP/USP Indirect Addressing With 7-Bit Displacement

Function

This addressing mode specifies an address in the current physical segment of data memory (addresses 0-0FFFFH = 64K bytes) with the contents of a pointing register as the base and a 7-bit signed displacement (bits 6-0, with bit 6 the sign bit) in the instruction code. Addresses within a range -64 to +63 of the contents of a pointing register can be accessed. The specified address can be accessed as word, byte, or bit data.

Syntax

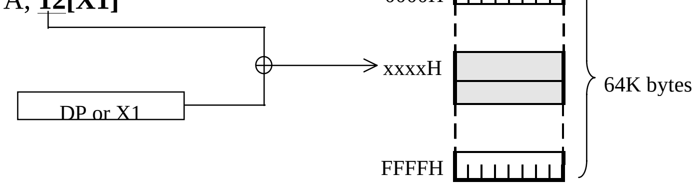
constant_expression[DP]	DP indirect addressing with 7-bit displacement
constant_expression[USP]	USP indirect addressing with 7-bit displacement

The constant_expression is a value in the range -64 to +63.
DP and USP can be used as the pointing register.

Only [DP] can be used with nX-8/100-400. For these, addresses within a range -128 to +127 of the contents of the pointing register can be accessed.

Word format

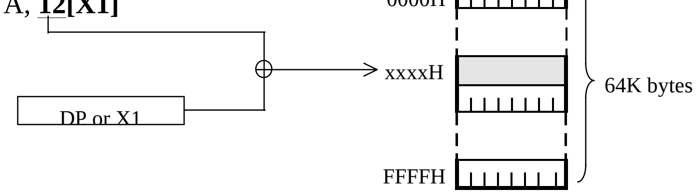
```
L       A, 12[DP]
L       A, 12[X1]
```



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

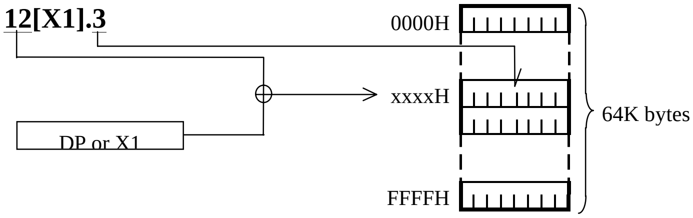
Byte format

```
LB      A, 12[DP]
LB      A, 12[X1]
```



Bit format

```
SB      12[DP].3
RB      12[X1].3
```



D16 [X1] / *D16* [X2] X1/X2 Indirect Addressing With 16-Bit Base

Function

This addressing mode specifies an address in the current physical segment of data memory (addresses 0-0FFFFH = 64K bytes) with two bytes in the instruction code (D16) as a base added to the contents of a pointing register (X1 or X2). The addition to generate the address is word (16-bit) addition with overflows ignored. Accordingly, the address generated will be 0 to 0FFFFH. The specified address can be accessed as word, byte, or bit data.

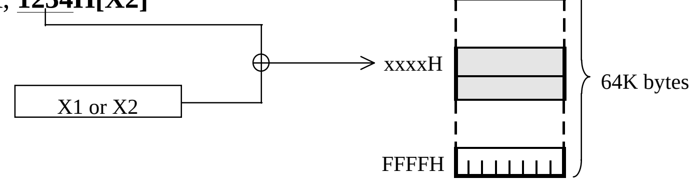
Syntax

address_expression[X1]	X1 indirect addressing with 16-bit base
address_expression[X2]	X2 indirect addressing with 16-bit base

The address_expression is a value in the range 0 to 0FFFFH. However, the assembler permits a range -8000H to +0FFFFH. D16 could be thought of not as a base address, but as a displacement.

Word format

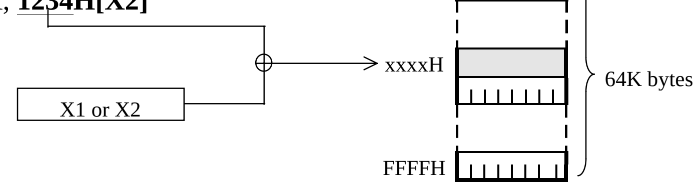
```
L     A, 1234H[X1]
ST    A, 1234H[X2]
```



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

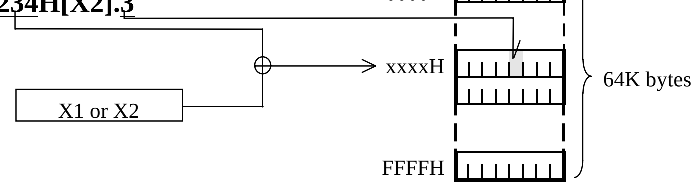
Byte format

```
LB    A, 1234H[X1]
STB   A, 1234H[X2]
```



Bit format

```
SB    1234H[X1].3
RB    1234H[X2].3
```



[X1+A] / [X1+R0] X1 Indirect Addressing With 8-Bit Register Displacement

Function

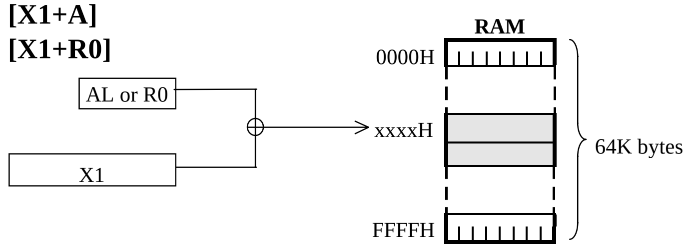
This addressing mode specifies an address in the current physical segment of data memory (addresses 0-0FFFFH = 64K bytes) with the contents of a pointing register as the base added to the contents of the low byte of the accumulator (AL) or local register 0 (R0). The addition to generate the address is word (16-bit) addition, with the 8-bit displacement from the register extended without sign. Overflow from this addition is ignored, so the generated value will be 0 to 0FFFFH. The specified address can be accessed as word, byte, or bit data.

Syntax

[X1+A]	X1 indirect addressing with 8-bit register displacement (AL)
[X1+R0]	X1 indirect addressing with 8-bit register displacement (R0)

Word format

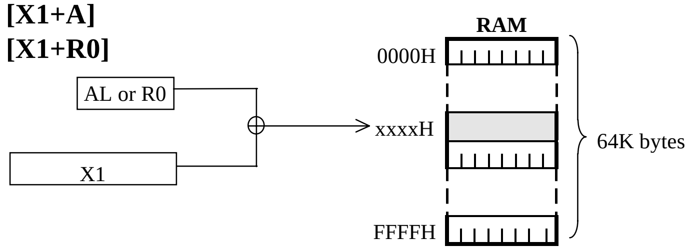
MOV A, **[X1+A]**
MOV A, **[X1+R0]**



If an odd address is specified, then the data word starting at the even address immediately below it will be accessed (→word boundary).

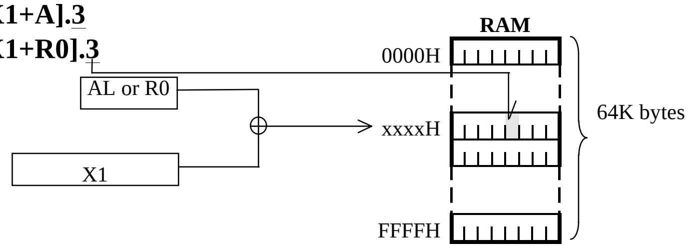
Byte format

MOVB A, **[X1+A]**
MOVB A, **[X1+R0]**



Bit format

SB **[X1+A].3**
RB **[X1+R0].3**



sbafix *Badr* — Fixed page SBA area addressing

Function

This addressing mode specifies a bit address in the 512-bit SBA area (2C0H.0-2FFH.7) in the fixed page. The specified address is accessed as bit data.

Syntax

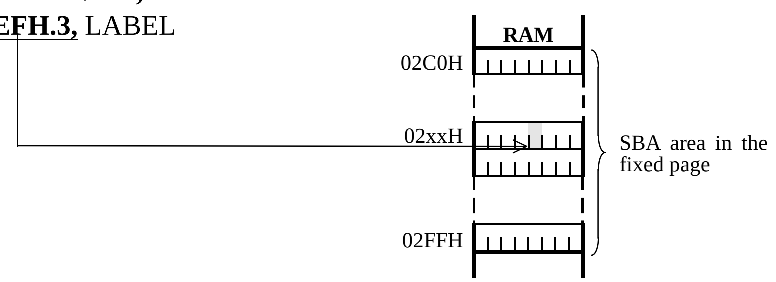
sbafix *address_expression* *address_expression*

Four instructions can be coded with this addressing mode: SB, RB, JBS, and JBR.

Bit format

SB	**sbafix 2C0H.0**
RB	**sbafix 1600H**
JBS	**sbafix FIXBITVAR,** LABEL
JBR	**sbafix 2EFH.7,** LABEL

SB	**2C0H.0**
RB	**1600H**
JBS	**FIXBITVAR,** LABEL
JBR	**2EFH.3,** LABEL



sbaoff *Badr* — Current page SBA area addressing

Function

This addressing mode specifies a bit address in the 512-bit SBA area (xxC0H.0-xxFFH.7) in the current page. The specified address is accessed as bit data.

Syntax

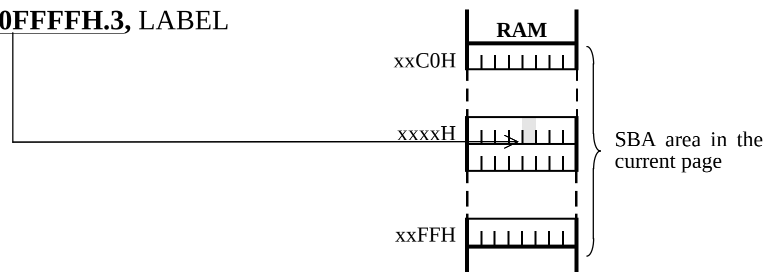
sbaoff *address_expression*
＼*address_expression*

Four instructions can be coded with this addressing mode: SB, RB, JBS, and JBR.

Bit format

SB	**sbaoff 4C0H.0**
RB	**sbaoff 2E80H**
JBS	**sbaoff VAR,** LABEL
JBR	**sbaoff 0FFFFH.3,** LABEL

SB	**＼2C0H.0**
RB	**＼2E80H**
JBS	**＼VAR,** LABEL
JBR	**＼0FFFFH.3,** LABEL



2-3. ROM Addressing

2-3-1. Immediate Addressing

These addressing modes access immediate data including in the instruction code.

2-3-2. Table Data Addressing

These addressing modes access the 64K bytes in the current table segment area in ROM space.

(1) Direct addressing

(2) Indirect addressing

3-3-3. Program Code Addressing

These addressing modes access the current program code in ROM space. They are used for operands in branch instructions.

(1) Direct addressing

(2) Relative addressing

(3) Special code addressing for particular instructions

(4) Indirect addressing

#N16　/　#N8　　Word/Byte Immediate Addressing

Function

For words, this addressing mode accesses two bytes (N16) in the instruction code.　For bytes, it accesses one byte (N8) in the instruction code.

Syntax

#expression

For words, the expression has a value in the range 0-0FFFFH.　For bytes, it has a value in the range 0-0FFH.　However, the assembler permits values in the ranges covered by both signed and unsigned expressions.　For words, this range is -8000H to +0FFFFH.　For bytes, it is -80H to +0FFH.

Word format

```
L       A, #1234H
MOV     X1, #WORD_ARRAY_BASE
```

Byte format

```
LB      A, #12H
MOVB    X1, #BYTE_ARRAY_BASE
```

Tadr Direct Table Addressing

Function

This addressing mode specifies an address in the current table segment (0-0FFFFH: 64K bytes) with two bytes in the instruction code. The specified address can be accessed as word or byte bit data.

Four instructions can use this addressing mode: LC, LCB, CMPC, and CMPCB.

Syntax

address_expression

The expression indicates and table address and is coded as the operand.

Word format

LC	A, **VAR**
CMPC	A, **VAR**

Byte format

LCB	A, **VAR**
CMPCB	A, **VAR**

[**] RAM Address Indirect Table Addressing

Function

This indirect addressing mode uses word data specified by RAM addressing as a pointer to the current table segment. Table memory can thus be accessed by placing a pointer to table memory in a register or in data memory.

Four instructions can use this addressing mode: LC, LCB, CMPC, and CMPCB.

Syntax

[RAM_*address_specification*]

A word RAM address specification is entered in the brackets.

Word format

LC	A, **[A]**
CMPC	A, **[1234[X1]]**

Byte format

LCB	A, **[ER0]**
CMPCB	A, **[VAR]**

T16[**] RAM Address Indirect Addressing With 16-Bit Base

Function

This addressing mode specifies an address in the current table segment (0-0FFFFH: 64K bytes) with a two-byte base (T16) in the instruction code added to the word data specified by RAM addressing. The addition to generate the address is word (16-bit) addition with overflows ignored. Accordingly, the address generated will be 0 to 0FFFFH. The specified address can be accessed as word or byte data.

Four instructions can use this addressing mode: LC, LCB, CMPC, and CMPCB.

Syntax

address_expression[*RAM_address_specification*]

A word RAM address specification is entered in the brackets.

Word format

LC	A, **2000H[A]**
CMPC	A, **2000H[1234[X1]]**

Byte format

LCB	A, **2000H[ER0]**
CMPCB	A, **2000H[VAR]**

Cadr Near Code Addressing

Function

This addressing mode specifies an address in the current code segment (0-0FFFFH: 64K bytes) with two bytes in the instruction code.

Two instructions can use this addressing mode: J and CAL.

Syntax

address_expression

The expression indicates a code address as the operand.

Example use

```
J      3000H
CAL    LABEL
```

Fadr Far Code Addressing

Function

This addressing mode specifies an address anywhere in program memory (0:0-0FFH:0FFFFH: 16M bytes) with three bytes in the instruction code.

Two instructions can use this addressing mode: FJ and FCAL.

Syntax

address_expression

The expression indicates a code address as the operand

Example use

FJ	**20H:3000H**
FCAL	**FARLABEL**

radr Relative Code Addressing

Function

This addressing mode specifies an address in the current code segment (0-0FFFFH: 64K bytes) with the current program counter (PC) as a base added to an 8-bit or 7-bit sign-extended value in the instruction code. The addition to generate the address is word (16-bit) addition with overflows ignored. Accordingly, the address generated will be 0 to 0FFFFH.

Instructions that can use this addressing mode are the SJ instruction and conditional branch instructions.

Syntax

address_expression

The expression indicates a code address as the operand.

Example use

SJ	**LABEL**
DJNZ	R0, **LABEL**
JC	LT, **LABEL**

Cadr11 ACAL Code Addressing

Function

This addressing mode specifies an address in the ACAL area current code segment (1000H-17FFH: 2K bytes) with 11 bits in the instruction code.

Only the ACAL instruction can use this addressing mode.

Syntax

address_expression

The expression indicates a code address as the operand.

Example use

```
ACAL    1000H
ACAL    ACALLABEL
```

Vadr VCAL Code Addressing

Function

This addressing mode specifies a vector (word data) for the VCAL instruction with four bits in the instruction code.

Only the VCAL instruction can use this addressing mode.

Syntax

address_expression

The expression indicates a code address as the operand.

Example use

```
VCAL    4AH
VCAL    0:4AH
VCAL    VECTOR
```

[**] RAM Address Indirect Code Addressing

Function

This indirect addressing mode uses word data specified by RAM addressing as a pointer to the current code segment. Indirect jumps and calls can be executed by placing a pointer to code memory in a register or in data memory.

Two instructions can use this addressing mode: J and CAL.

Syntax

[*RAM_address_specification*]

A word RAM address specification is entered in the brackets.

Example use

```
J       [A]
CAL     [1234[X1]]
```

2-4.   ROM Window Addressing

ROM window addressing accessed table data in ROM space using RAM addressing. It reads data in a table segment through a window in a data segment opened by the program.

Addressing to data memory in the ROM window area is permitted, but if an instruction that writes to the ROM window is executed, then the results are not guaranteed.

Chapter 3. Instruction Details

This chapter explains the functions of each nX-8/500S core instruction in detail.

nX-8/500S Instruction Set Listed By Function

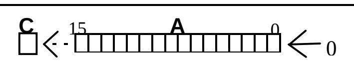
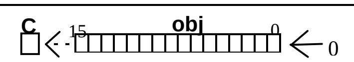
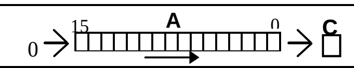
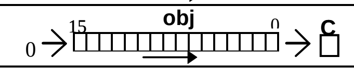
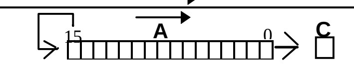
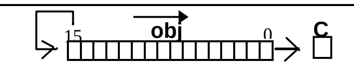
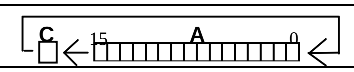
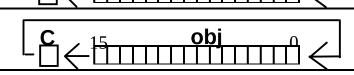
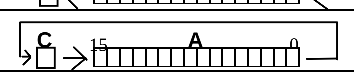
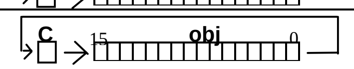
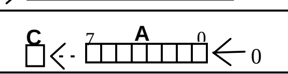
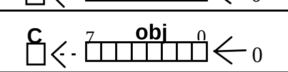
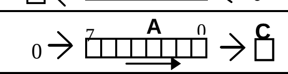
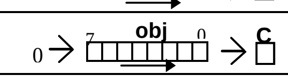
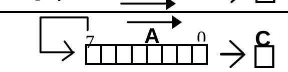
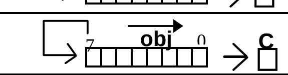
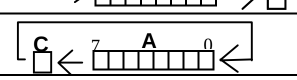
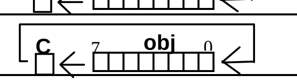
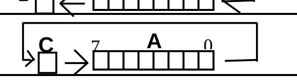
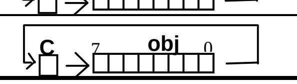
Data Move

Mnemonic	Operand	CZSVHD D	Function	
L	A,obj	. z. . .1 .	Word move (Word load)	A←obj, DD←1
ST	A,obj	 1	Word move (Word store)	obj ← A
MOV	obj1, obj2	. . . .	Word move	obj1 ← obj2
CLR	A	. 1. . . 1 .	Word clear	A←0, DD←1
	obj	. . . .	Word clear	obj←0
FILL	A	 1	Word fill	A←0FFFFH
	obj	. . . .	Word fill	obj←0FFFFH
XCHG	A,obj	 1	Word exchange	A ↔ obj
SWAP		. . . .	High/low byte swap	AH ↔ AL
LB	A,obj	. z. . . 0 .	Byte move (Byte load)	AL←obj, DD←0
STB	A,obj	 0	Byte move (Byte store)	obj ← AL
MOVB	obj1, obj2	. . . .	Byte move	obj1 ←obj2
CLRB	A	. 1. . . 0 .	Byte clear	AL←0, DD←0
	obj	. . . .	Byte clear	obj←0
FILLB	A	 0	Byte fill	AL←0FFH
	obj	. . . .	Byte fill	obj←0FFH
XCHGB	A,obj	 0	Byte exchange	AL ↔ obj

Stack Manipulation

Mnemonic	Operand	CZSVHD D	Function	
PUSHS	register_list	. . . .	Push on system stack	System stack←Register group
POPS	register_list	CZSVHD .	Pop off system stack	Register group←System stack

Shift/Rotate

Mnemonic	Operand	CZSVHD	D	Function	
SLL	A,width A	C.	1	Word left shift (with carry) width=1 to 4	
	obj,width obj	C.	.	Word left shift (with carry) width=1 to 4	
SRL	A,width A	C.	1	Word right shift (with carry) width=1 to 4	
	obj,width obj	C.	.	Word right shift (with carry) width=1 to 4	
SRA	A,width A	C.	1	Word arithmetic right shift (with carry) width=1 to 4	
	obj,width obj	C.	.	Word arithmetic right shift (with carry) width=1 to 4	
ROL	A,width A	C.	1	Word left rotate (with carry) width=1 to 4	
	obj,width obj	C.	.	Word left rotate (with carry) width=1 to 4	
ROR	A,width A	C.	1	Word right rotate (with carry) width=1 to 4	
	obj,width obj	C.	.	Word right rotate (with carry) width=1 to 4	
SLLB	A,width A	C.	0	Byte left shift (with carry) width=1 to 4	
	obj,width obj	C.	.	Byte left shift (with carry) width=1 to 4	
SRLB	A,width A	C.	0	Byte right shift (with carry) width=1 to 4	
	obj,width obj	C.	.	Byte right shift (with carry) width=1 to 4	
SRAB	A,width A	C.	0	Byte arithmetic right shift (with carry) width=1 to 4	
	obj,width obj	C.	.	Byte arithmetic right shift (with carry) width=1 to 4	
ROLB	A,width A	C.	0	Byte left rotate (with carry) width=1 to 4	
	obj,width obj	C.	.	Byte left rotate (with carry) width=1 to 4	
RORB	A,width A	C.	0	Byte right rotate (with carry) width=1 to 4	
	obj,width obj	C.	.	Byte right rotate (with carry) width=1 to 4	

Increment/Decrement

Mnemonic	Operand	CZSVHD D	Function	
INC	A	. ZSVH. . .1	Word increment	A←A+1
	obj	. ZSVH. . ..	Word increment	obj←obj+1
DEC	A	. ZSVH. . .1	Word decrement	A←A-1
	obj	. ZSVH. . ..	Word decrement	obj←obj-1
INCB	A	. ZSVH. . .0	Byte increment	AL←AL+1
	obj	. ZSVH. . ..	Byte increment	obj←obj+1
DECB	A	. ZSVH. . .0	Byte decrement	AL←AL-1
	obj	. ZSVH. . ..	Byte decrement	obj←obj-1

Arithmetic Calculation

Mnemonic	Operand	CZSVHD D	Function	
MUL	obj	. Z.	Word multiplication	<A,ER0> ← A × obj
SQR	A	. Z. . . . 1	Word square	<A,ER0> ← A × A
DIV	obj	CZ.	Word division	<A,ER0> ← <A,ER0> ÷ obj, ER1 ← <A,ER0> mod obj
DIVQ	obj	CZ. V. . .	Word quick division	A ← <A,ER0> ÷ obj, ER1 ← <A,ER0> mod obj
ADD	A,obj	CZSVH 1	Word addition	A←A+obj
	obj1, obj2	CZSVH .	Word addition	obj1←obj1+obj2
ADC	A,obj	CZSVH 1	Word addition with carry	A←A+obj+C
	obj1, obj2	CZSVH .	Word addition with carry	obj1←obj1+obj2+C
SUB	A,obj	CZSVH 1	Word subtraction	A←A-obj
	obj1, obj2	CZSVH .	Word subtraction	obj1←obj1-obj2
SBC	A,obj	CZSVH 1	Word subtraction with carry	A ← A-obj-C
	obj1, obj2	CZSVH .	Word subtraction with carry	obj1 ← obj1-obj2-C
CMP	A,obj	CZSVH 1	Word comparison	A-obj
	obj1, obj2	CZSVH .	Word comparison	obj1-obj2
NEG	A	CZSVH 1	Word negation	A ← -A
MULB	obj	. Z.	Byte multiplication	A ← AL × obj
SQRB	A	. Z. . . . 0	Byte square	A ← AL × AL
DIVB	obj	CZ.	Byte division	A ← A ÷ obj, R1 ←A mod obj
ADDB	A,obj	CZSVH 0	Byte addition	AL←AL+obj
	obj1, obj2	CZSVH .	Byte addition	obj1←obj1+obj2
ADCB	A,obj	CZSVH 0	Byte addition with carry	AL←AL+obj+C
	obj1, obj2	CZSVH .	Byte addition with carry	obj1←obj1+obj2+C
SUBB	A,obj	CZSVH 0	Byte subtraction	AL ← AL-obj
	obj1, obj2	CZSVH .	Byte subtraction	obj1←obj1-obj2
SBCB	A,obj	CZSVH 0	Byte subtraction with carry	AL ← AL-obj-C
	obj1, obj2	CZSVH .	Byte subtraction with carry	obj1 ← obj1-obj2-C
CMPB	A,obj	CZSVH 0	Byte comparison	AL-obj
	obj1, obj2	CZSVH .	Byte comparison	obj1-obj2
NEGB	A	CZSVH 0	Byte negation	AL ← -AL

Logical Calculation

Mnemonic	Operand	CZSVHD D	Function	
AND	A,obj	. ZS. . . 1	Word logical AND	A← A ∩ obj
	obj1,obj2	. ZS. . . .	Word logical AND	obj1 ← obj1 ∩ obj2
OR	A,obj	. ZS. . . 1	Word logical OR	A ← A ∪ obj
	obj1,obj2	. ZS. . . .	Word logical OR	obj1 ← obj1 ∪ obj2
XOR	A,obj	. ZS. . . 1	Word logical exclusive OR	A ← A ⊕ obj
	obj1,obj2	. ZS. . . .	Word logical exclusive OR	obj1 ← obj1 ⊕ obj2
ANDB	A,obj	. ZS. . . 0	Byte logical AND	AL←AL∩obj
	obj1,obj2	. ZS. . . .	Byte logical AND	obj1←obj1∩obj2
ORB	A,obj	. ZS. . . 0	Byte logical OR	AL ← AL ∪ obj
	obj1,obj2	. ZS. . . .	Byte logical OR	obj1 ← obj1 ∪ obj2
XORB	A,obj	. ZS. . . 0	Byte logical exclusive OR	AL ← AL ⊕ obj
	obj1,obj2	. ZS. . . .	Byte logical exclusive OR	obj1 ← obj1 ⊕ obj2

ROM table Reference

Mnemonic	Operand	CZSVHD D	Function	
LC	A,[obj]	. Z. . . .	Word ROM data move (indirect)	A ← TSR:(obj)
	A,T16[obj]	. Z. . . .	Word ROM data move (indirect with base)	A ← TSR:(T16 + obj)
	A,Tadr	. Z. . . .	Word ROM data move (direct)	A ← TSR:Tadr
CMPC	A,[obj]	CZSVH .	Word ROM comparison (indirect)	A - TSR:(obj)
	A,T16[obj]	CZSVH .	Word ROM comparison (indirect with base)	A - TSR:(T16 + obj)
	A,Tadr	CZSVH .	Word ROM comparison (direct)	AL - TSR:Tadr
LCB	A,[obj]	. Z. . . .	Byte ROM data move (indirect)	AL ← TSR:(obj)
	A,T16[obj]	. Z. . . .	Byte ROM data move (indirect with base)	AL ← TSR:(T16 + obj)
	A,Tadr	. Z. . . .	Byte ROM data move (direct)	AL ← TSR:Tadr
CMPCB	A,[obj]	CZSVH .	Byte ROM data move (indirect)	AL - TSR:(obj)
	A, [obj]	CZSVH .	Byte ROM data move (indirect with base)	AL - TSR:(obj)
	A,T16[obj]	CZSVH .	Byte ROM data move (direct)	AL - TSR:(T16 + obj)

Bit Manipulation

Mnemonic	Operand	CZSVHD D	Function	
SBR	obj	. Z. . . .	Set bit (register indirect bit specification)	obj.(AL) ← 1
RBR	obj	. Z. . . .	Reset bit (register indirect bit specification)	obj.(AL) ← 0
TBR	obj	. Z. . . .	Test bit (register indirect bit specification)	if obj.(AL)=0 then Z←1 else Z←0
MBR	C,obj	C.	Bit move (register indirect bit specification)	C ← obj.(AL)
	obj,C	. . . .	Bit move (register indirect bit specification)	obj.(AL) ← C
SB	obj.bit	. Z. . . .	Set bit (bit position direct specification)	if obj.bit = 1 then Z←1 else Z←0, obj.bit←1
RB	obj.bit	. Z. . . .	Reset bit (bit position direct specification)	if obj.bit = 0 then Z←1 else Z←0, obj.bit←0
MB	C,obj.bit	C.	Bit move	C ← obj.bit
	obj.bit,C	. . . .	Bit move	obj.bit ← C
BAND	C,obj.bit	C.	Bit logical AND	C←C ∩ obj.bit
BOR	C,obj.bit	C.	Bit logical OR	C←C ∪ obj.bit
BXOR	C,obj.bit	C.	Bit logical exclusive OR	C←C ⊕ obj.bit
BANDN	C,obj.bit	C.	Bit logical AND with bit complement	C←C ∩ $\overline{\text{obj.bit}}$
BORN	C,obj.bit	C.	Bit logical OR with bit complement	C←C ∪ $\overline{\text{obj.bit}}$

Jump/Call

Mnemonic	Operand	CZSVHD D	Function	
JBS	obj.bit,radr	. . . .	Bit test & jump	if obj.bit=1 then PC←radr
JBR	obj.bit,radr	. . . .	Bit test & jump	if obj.bit=0 then PC←radr
JBSR	obj.bit,radr	. . . .	Bit test & jump (with bit reset)	if obj.bit=1 then obj.bit←0, PC←radr
JBRS	obj.bit,radr	. . . .	Bit test & jump (with bit set)	if obj.bit=0 then obj.bit←1, PC←radr
TJZ	A,radr	 1	Word test & jump (jump if zero)	if A=0 then PC←radr
	obj,radr	. . . .	Word test & jump (jump if zero)	if obj=0 then PC←radr
TJNZ	A,radr	 1	Word test & jump (jump if non-zero)	if A ≠ 0 then PC←radr
	obj,radr	. . . .	Word test & jump (jump if non-zero)	if obj ≠ 0 then PC←radr
TJZB	A,radr	 0	Byte test & jump (jump if zero)	if AL=0 then PC←radr
	obj,radr	. . . .	Byte test & jump (jump if non-zero)	if obj=0 then PC←radr
TJNZB	A,radr	 0	Byte test & jump (jump if non-zero)	if AL ≠ 0 then PC←radr
	obj,radr	. . . .	Byte test & jump (jump if zero)	if obj ≠ 0 then PC←radr
J*cond*	radr	. . . .	Conditional jump	if *cond* is true then PC←radr
DJNZ	obj,radr	. . . .	Loop	obj←obj-1,if obj ≠ 0 then PC←radr
JRNZ	DP,radr	. . . .	Loop	DPL←DPL-1,if DPL ≠ 0 then PC←radr
SJ	radr	. . . .	Short jump	PC←radr
J	Cadr	. . . .	64K-byte space (within current physical code segment) direct jump	
	[obj]	. . . .	64K-byte space (within current physical code segment) indirect jump	
CAL	Cadr	. . . .	64K-byte space (within current physical code segment) direct call	
	[obj]	. . . .	64K-byte space (within current physical code segment) indirect call	
VCAL	Vadr	. . . .	Vector call	
ACAL	Cadr11	. . . .	Special area call	
SCAL	Cadr	. . . .	64K-byte space (within current physical code segment) direct call	
RT		. . . .	Return from subroutine	
RTI		CZSVHD .	Return from interrupt	
FCAL	Fadr	. . . .	24-bit space (16M bytes: entire program area) direct call	
FJ	Fadr	. . . .	24-bit space (16M bytes: entire program area) direct jump	
FRT		. . . .	Return from far subroutine	

Other Instructions

Mnemonic	Operand	CZSVHD D	Function	
SC		1	Set carry	C ← 1
RC		0	Reset carry	C ← 0
CPL	C	C	Complement carry	C ← $\overline{C}$
SDD		. . . . . 1 .	Set DD	DD ← 1
RDD		. . . . . 0 .	Reset DD	DD ← 0
EI		. . . .	Enable interrupts	MIE ← 1
DI		. . . .	Disable interrupts	MIE ← 0
EXTND		. . S . . 1 .	Extend byte to word with sign	A_{15-7} ← A_7, DD ← 1
NOP		. . . .	No operation	NO OPERATION
BRK		0 0 0 0 0 0 .	Break (system reset)	RESET, PC ←(Vector-table 0002H)
MAC		. . . .	Multiply-accumulate	Multiply-accumulate start bit ← 1

Symbols Used In Operand Expressions Of Instruction

Operand Expressions			
Symbol	Syntax	Meaning	Permitted range of value
Registers			
ERn	ER0, ER1, ER2, ER3	Word local register	
PRn	X1, X2, DP, USP	Pointing register (PR0,PR1,PR2,PR3 correspond to X1,X2,DP,USP respectively)	
Rn	R0,R1, R2,R3, R4,R5, R6,R7	Byte local register	
Expressions representing data addresses (represents table addresses (TSR base) within the ROM window)			
D16	expression	Index indirect base data address	DSR:0H to DSR:0FFFFH
off	off expression \ expression	Current page data address	DSR:0H to DSR:0FFFFH
sfr	sfr expression expression	SFR page data address	DSR:0H to DSR:0FFH
fix	fix expression expression	Fixed page data address	DSR:200H to DSR:2FFH
dir	dir expression expression	Direct data address	DSR:0 to DSR:0FFFFH
sbafix	sbafix expression expression	Fixed page SBA bit address	DSR:2C0H.0 to DSR:2FFH.7
sbaoff	sbaoff expression \ expression	Current page SBA address	DSR:xxC0H.0 to DSR:xxFFH.7
\ , off, sfr, fix, dir, sbafix, and sbaoff are address specifiers.			
Expressions representing code addresses			
Cadr	expression	Code address within code segment	CSR:0H to CSR:0FFFFH
Cadr11	expression	ACAL code address	CSR:1000H to CSR:17FFH
Vadr	expression	VCAL vector address	0:4AH to 0:69H
Fadr	expression	FAR code address	0:0H to 0FFH:0FFFFH
radr	expression	Relative code address	CSR:0H to CSR:0FFFFH
Expressions representing ROM table addresses			
Tadr	expression	Address within table segment	TSR:0H to TSR:0FFFFH
Expressions representing constants			
N16	expression	Word immediate value	-8000H to +0FFFFH
N8	expression	Byte immediate value	-80H to +0FFH
n7	expression	Signed 6-bit displacement	-40H to +3FH
Operands specifying prefix codes			
**		Word prefix code group	
*		Byte prefix code group	
Other			
bit	expression	Bit position	0 to 7
width	expression	Shift width	1 to 4

*All character expressions in instruction tables other than those above are used as is.

Symbols Used In Instruction Code Expressions Of Instruction

Code Expressions		
Symbol	**Meaning**	**Field**
Instruction codes that specify registers		7 6 5 4 3 2 1 0
xx +n	ERn (n:0-3)	
	PRn (n:0-3)	
	Rn (n:0-7)	
Instruction codes that indicate data addresses		7 6 5 4 3 2 1 0
D16L	Low byte of D16 expression value	
D16H	High byte of D16 expression value	
off8	Low byte of expression value	
sfr8	Low byte of expression value	
fix8	Low byte of expression value	
dirL	Low byte of dir	
dirH	High byte of dir	
Instruction codes that indicate code addresses		7 6 5 4 3 2 1 0
CadrL	Low byte of Cadr expression value	
CadrH	High byte of Cadr expression value	
Cadr11L	Low byte of Cadr11 expression value	
Cadr11H	3 bits (bit 10 to bit 8) of Cadr11 expression value (0-7)	
Vno4	Vector number (0-15)	
FadrL	Low byte of expression value	
FadrM	High byte of expression value	
FadrH	Physical code segment of expression	
rdiff8	Difference between radr and the next PC (-128 to +127)	
rdiff7	Difference between radr and the PC (-128 to -1)	
Instruction codes that indicate ROM table addresses		7 6 5 4 3 2 1 0
TadrL	Low byte of Tadr expression value	
TadrH	High byte of Tadr expression value	
T16L	Low byte of T16 expression value	
T16H	High byte of T16 expression value	
Instruction codes that indicate constants		
N16L	Word immediate value low byte	
N16H	Word immediate value high byte	
N8	Byte immediate value	
n7	Signed 6-bit displacement (-40H to 3FH)	
Other		7 6 5 4 3 2 1 0
bit	Bit position (0-7)	
width	Shift width 1-4 corresponding to code 0-3	
Prefix codes		7 6 5 4 3 2 1 0
<word>	Word prefix codes (1-3 bytes)	
<byte>	Byte prefix codes (1-3 bytes)	
<dumyW>	Dummy word prefix (X1 prefix code: 60H)	
<dumyB>	Dummy byte prefix (PSWL prefix code: 8AH)	

* All values other than those above are expressed as hexadecimal constants.

The following pages contain a reference of instruction details. Instructions are presented in alphabetical order, with the following as a general example. Basically one instruction is explained on one page.

□ General format of instruction
Shows general format of mnemonic and operands with symbols.

□ Metaformat of instruction operation
Shows instruction operation using easy-to-understand symbols.

□ Datailed description of instruction functions
Explains operation details, operand coding, restrictions,etc.

Simple meaning of instruction □

□ Flag changes
Classifies changes in flag states from instuction execution;
blank No change
0 Reset to 0
1 Set to 1
* Changes according to result

□ Necessary conditions for execution
Shows the necessarystate of the data descriptor for executing the instruction;
blank State of DD is irrelative
0 Reset to 0
1 Set to 1

□ Instruction code table
Shows instruction code and execution cycles with internal memory for the addressing groups permitted as operands.

Instruction code prefix table □
Each item in the left column can be coded as operand in the above instruction code table (* or **). The combined cycle count is found by summing both cycle count. A table entry isvalid when combined with the same instruction.

Chapter 3 Instruction Details
Instruction Set

ADDB A,obj — Byte Addition

Function

AL←AL+obj

Description

- This instruction performs byte addition, adding the contents of obj (byte length) to the accumulator low byte (AL).
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ADCB	A	Rn	28+n						3
		#N8	AE	N8					4
		fix	AC	fix8					4
		off	AD	off8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
ADDB	A	*	<byte>	F5					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP−]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

A-8 Chapter 3 nX-8/500S Instruction Manual

ACAL Cadr11 — Special Area Call

Function

(SSP)←PC+2
SSP←SSP-2
PC←Cadr11
However, CSR:1000H ≤ Cadr11 ≤ CSR:17FFH

Description

- This instruction calls the ACAL area in the current physical segment. The ACAL area is the 2K-bytes starting from address 1000H in code space.
- The state of the stack after execution of an ACAL instruction is identical to that after execution of a CAL instruction. Subroutines called with an ACAL instruction return using an RT instruction.
- The first address of the subroutine is coded in Cadr11.
- ACAL area subroutines can be called more efficiently with the ACAL instruction than the CAL instruction.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
ACAL	Cadr11	44+Cadr11H	Cadr11L					7

ADC A,obj

Word Addition With Carry

Function

A←A+obj+C

Description

- This instruction performs word addition, adding the contents of obj (word length) and the carry flag to the accumulator (A).
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ADC	A	#N16	BC	F3	N16L	N16H			8

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	(Internal)
ADC	A	**	<word>	F5					+2

**	<word> Word Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

ADC obj1,obj2 — Word Addition With Carry

Function

obj1←obj1+obj2+C

Description

- This instruction performs word addition, adding the contents of obj2 (word length) and the carry flag to obj1 (word length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
ADC	**	fix	<word>	F0	fix8				+5
		off	<word>	F1	off8				+5
		sfr	<word>	F2	sfr8				+5
		#N16	<word>	F3	N16L	N16H			+6
		A	<word>	F4					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

ADCB A,obj

Byte Addition With Carry

Function

AL←AL+obj+C

Description

- This instruction performs byte addition, adding the contents of obj (byte length) and the carry flag to the accumulator low byte (AL).
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ADCB	A	#N8	BC	F3	N8				6

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
ADCB	A	*	<byte>	F5					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1th	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

ADCB obj1,obj2

Byte Addition With Carry

Function

obj1←obj1+obj2+C

Description

- This instruction performs byte addition, adding the contents of obj2 (byte length) and the carry flag to obj1 (byte length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
ADCB	*	fix	<byte>	F0	fix8				+5
		off	<byte>	F1	off8				+5
		sfr	<byte>	F2	sfr8				+5
		#N8	<byte>	F3	N8				+4
		A	<byte>	F4					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

ADD A,obj

Word Addition

Function

A←A+obj

Description

- This instruction performs word addition, adding the contents of obj (word length) to the accumulator (A).
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ADD	A	ERn	28+n						3
		PRn	2C+n						3
		#N16	AE	N16L	N16H				6
		fix	AC	fix8					4
		off	AD	off8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	(Internal)
ADD	A	**	<word>	A5					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

ADD obj1,obj2

Word Addition

Function

obj1←obj1+obj2

Description

- This instruction performs word addition, adding the contents of obj2 (word length) to obj1 (word length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
ADD	**	fix	<word>	A0	fix8				+5
		off	<word>	A1	off8				+5
		sfr	<word>	A2	sfr8				+5
		#N16	<word>	A3	N16L	N16H			+6
		A	<word>	A4					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

ADDB A,obj

Byte Addition

Function

AL←AL+obj

Description

- This instruction performs byte addition, adding the contents of obj (byte length) to the accumulator low byte (AL).
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ADDB	A	Rn	28+n						3
		#N8	AE	N8					4
		fix	AC	fix8					4
		off	AD	off8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
ADDB	A	*	<byte>	A5					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

ADDB obj1,obj2

Byte Addition

Function

obj1←obj1+obj2

Description

- This instruction performs byte addition, adding the contents of obj2 (byte length) to obj1 (byte length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
ADDB	*	fix	<byte>	A0	fix8				+5
		off	<byte>	A1	off8				+5
		sfr	<byte>	A2	sfr8				+5
		#N8	<byte>	A3	N8				+4
		A	<byte>	A4					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

AND A,obj

Word Logical AND

Function

A←A ∩ obj

Description

- This instruction takes the word logical AND of the contents of obj (word length) and the accumulator (A), and stores the result in the accumulator.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
AND	A	off	BD	off8					4
		#N16	BE	N16L	N16H				6

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
AND	A	**	<word>	B5					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

AND obj1,obj2

Word Logical AND

Function

obj1←obj1 ∩ obj2

Description

- This instruction takes the word logical AND of the contents of obj1 (word length) and obj2 (word length), and stores the result in obj1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
AND	**	fix	<word>	B0	fix8				+5
		off	<word>	B1	off8				+5
		sfr	<word>	B2	sfr8				+5
		#N16	<word>	B3	N16L	N16H			+6
		A	<word>	B4					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

ANDB A,obj

Byte Logical AND

Function

AL←AL ∩ obj

Description

- This instruction takes the word logical AND of the contents of obj (byte length) and the accumulator low byte (AL), and stores the result in the accumulator.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ANDB	A	off	BD	off8					4
		#N8	BE	N8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
ANDB	A	*	<byte>	B5					+2

**	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

ANDB obj1,obj2

Byte Logical AND

Function

obj1←obj1 ∩ obj2

Description

- This instruction takes the word logical AND of the contents of obj1 (byte length) and obj2 (byte length), and stores the result in obj1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
ANDB	*	fix	<byte>	B0	fix8				+5
		off	<byte>	B1	off8				+5
		sfr	<byte>	B2	sfr8				+5
		#N8	<byte>	B3	N8				+4
		A	<byte>	B4					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

BAND　C,obj.bit

Bit Logical AND

Function

C←C ∩ obj.bit

Description

- This instruction takes the logical AND of the specified bit in obj (byte length) and the carry (C), and stores the result in carry.　The bit has a value of 0-7.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
BAND	C	*.bit	<byte>	40+bit					+3

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68　+n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80　+n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

BANDN C,obj.bit

Bit Complement and Bit Logical

Function

$C \leftarrow C \cap \overline{\text{obj.bit}}$

Description

• This instruction takes the logical AND of the complement of the specified bit in obj (byte length) and the carry (C), and stores the result in carry. The bit has a value of 0-7.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
BANDN	C	*.bit	<byte>	48+bit					+3

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

BOR C,obj.bit

Bit Logical OR

Function

C←C ∪ obj.bit

Description

- This instruction takes the logical OR of the specified bit in obj (byte length) and the carry (C), and stores the result in carry. The bit has a value of 0-7.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
BOR	C	*.bit	<byte>	50+bit					+3

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

BORN C,obj.bit

Bit Complement and Bit Logical OR

Function

$C \leftarrow C \cup \overline{obj.bit}$

Description

- This instruction takes the logical OR of the complement of the specified bit in obj (byte length) and the carry (C), and stores the result in carry. The bit has a value of 0-7.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
BORN	C	*.bit	<byte>	58+bit					+3

*	<byte> Byte Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

BRK

Break (System Reset)

Function

SYSTEM RESET
PC←(Vector-table 0002H)

Description

- This instruction performs a software system reset.
- The CPU first performs system reset processing. Next the word data at address 2 in the code space reset vector table (the first address of the break processing routine) is moved to the PC.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
0	0	0	0	0	0

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
BRK		FF						2

BXOR C,obj.bit

Bit Logical Exclusive OR

Function

C←C ⊻ obj.bit

Description

- This instruction takes the logical exclusive OR of the specified bit in obj (byte length) and the carry (C), and stores the result in carry. The bit has a value of 0-7.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
BXOR	C	*.bit	<byte>	60+bit					+3

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

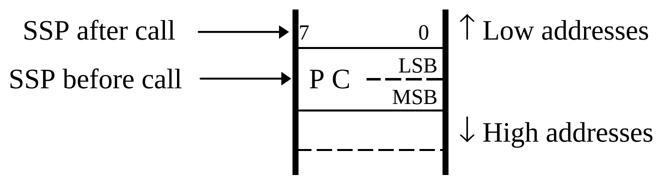
CAL Cadr 64K-Byte Space (Within Current Physical Code Segment) Direct Call

Function

(SSP) ←PC+3
SSP←SSP-2
PC←Cadr
However, CSR:0000H ≤ Cadr ≤ CSR:0FFFFH

Description

- This instruction calls any addresss in the 64K bytes in the current physical segment.
- The first address of the subroutine is coded in Cadr. The subroutine must exist within the current physical segment.
- The state of the stack after execution of a CAL instruction is shown below. Subroutines called with a CAL instruction return using an RT instruction.



Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
CAL	Cadr	FE	CadrL	CadrH				9

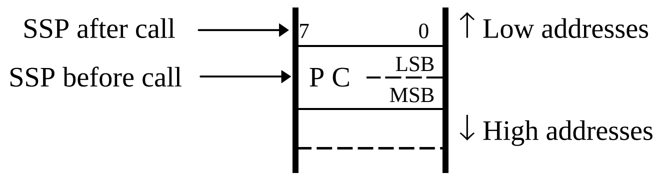
CAL [obj] 64K-Byte Space (Within Current Physical Code Segment) Indirect Call

Function

(SSP) ←PC+n
SSP←SSP-2
PC←obj
However, n is the number of bytes in this instruction and differs depending on obj.

Description

- This instruction is a 64K-byte space indirect call based on the contents of obj (word length).
- This instruction calls any addresss in the 64K bytes in the current physical segment.
- obj is the word-length contents of data memory or a register. The first address of the subroutine must be set in obj prior to executing this instruction. The subroutine must exist within the current physical segment.
- The state of the stack after execution of a CAL instruction is shown below. Subroutines called with a CAL instruction return using an RT instruction.



Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	prefix	+1st	+2nd	+3rd	+4th	+5th	(Internal)
CAL	[**]	<word>	EB					+5

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CLR A

Word Clear

Function

A←0
DD←1

Description

- This instruction clears the accumulator (word length).
- This instruction also sets DD to 1 (word).
- This instruction is functionally identical to the "L A,#0" instruction, including the effect on flags. However, this instruction requires fewer bytes and cycles.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	1				1

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
CLR	A	FA						2

CLR obj

Word Clear

Function

obj←0

Description

- This instruction clears obj (word length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd			
CLR	**	<word>	C7					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CLRB A

Byte Clear

Function

AL←0
DD←0

Description

- This instruction clears the accumulator (byte length).
- This instruction also sets DD to 0 (byte).
- This instruction is functionally identical to the "LB A,#0" instruction, including the effect on flags. However, this instruction requires fewer bytes and cycles.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	1				0

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
CLRB	A	FB						2

CLRB obj

Byte Clear

Function

obj←0

Description

- This instruction clears obj (byte length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd			
CLRB	*	<byte>	C7					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

CMP　A,obj

Word Comparison

Function

A-obj

Description

- This instruction compares the contents of obj (word length) to the accumulator (A).
- Actually the contents of obj are subtracted from the contents of the accumulator, and the result is used to set the PSW flags.　This result can be referenced using conditional branch instructions.　The accumulator contents do not change.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
CMP	A	ERn	18　+n						3
		PRn	1C　+n						3
		#N16	9E	N16L	N16H				6
		fix	9C	fix8					4
		off	9D	off8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
CMP	A	**	<word>	95					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64　+n			2
PRn	60　+n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80　+n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CMP obj1,obj2 — Word Comparison

Function

obj1-obj2

Description

- This instruction compares the contents of obj1 to obj2 (word length).
- Actually the contents of obj2 are subtracted from the contents of obj1, and the result is used to set the PSW flags. This result can be referenced using conditional branch instructions. The contents of obj1 do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
CMP	fix	#N16	C4	fix8	N16L	N16H			8
	off		C5	off8	N161	N16H			8

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
CMP	**	fix	<word>	90	fix8				+5
		off	<word>	91	off8				+5
		sfr	<word>	92	sfr8				+5
		#N16	<word>	93	N16L	N16H			+6
		A	<word>	94					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CMPB A,obj

Byte Comparison

Function

AL-obj

Description

- This instruction compares the contents of obj (byte length) to the accumulator low byte (AL).
- Actually the contents of obj are subtracted from the contents of the accumulator, and the result is used to set the PSW flags. This result can be referenced using conditional branch instructions. The accumulator contents do not change.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
CMPB	A	Rn	18+n						3
		#N8	9E	N8					4
		fix	9C	fix8					4
		off	9D	off8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
CMPB	A	*	<byte>	95					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

CMPB　obj1,obj2　　Byte Comparison

Function

obj1-obj2

Description

- This instruction compares the contents of obj1 to obj2 (byte length).
- Actually the contents of obj2 are subtracted from the contents of obj1, and the result is used to set the PSW flags.　This result can be referenced using conditional branch instructions.　The contents of obj1 do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
CMPB	fix	#N8	D4	fix8	N8				6
	off		D5	off8	N8				6

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
CMPB	*	fix	<byte>	90	fix8				+5
		off	<byte>	91	off8				+5
		sfr	<byte>	92	sfr8				+5
		#N8	<byte>	93	N8				+4
		A	<byte>	94					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68　+n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80　+n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

CMPC A, [obj] Word ROM Comparison (Indirect)

Function

A - TSR:(obj)

Description

- This instruction compares ROM data (word length) to the accumulator (A).
- The ROM data is word data in the current table segment, with the contents of obj as the address.
- Actually the ROM data is subtracted from the contents of the accumulator, and the result is used to set the PSW flags. This result can be referenced using conditional branch instructions. The accumulator contents do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
CMPC	A	[**]	<word>	D8					+9

**	<word> Word Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CMPC A,T16[obj] Word ROM Comparison (Indirect With 16-Bit Base)

Function

A - TSR:(T16 +obj)

Description

- This instruction compares ROM data (word length) to the accumulator (A).
- The ROM data is word data in the current table segment, with the contents of obj added to the base address of the data table (T16) as the address.
- Actually the ROM data is subtracted from the contents of the accumulator, and the result is used to set the PSW flags. This result can be referenced using conditional branch instructions. The accumulator contents do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
CMPC	A	T16[**]	<word>	E6	T16L	T16H			+13

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CMPC A,Tadr — Word ROM Comparison (Direct)

Function

A - TSR:Tadr

Description

- This instruction compares ROM data (word length) to the accumulator (A).
- The ROM data is the word data in the current table segment indicated by Tadr.
- Actually the ROM data is subtracted from the contents of the accumulator, and the result is used to set the PSW flags. This result can be referenced using conditional branch instructions. The accumulator contents do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
CMPC	A	Tadr	<dumyW>	B6	TadrL	TadrH			+15

CMPCB　A,[obj]　　Byte ROM Comparison (Indirect)

Function

AL - TSR:(obj)

Description

- This instruction compares ROM data (byte length) to the accumulator low byte (AL).
- The ROM data is byte data in the current table segment, with the contents of obj as the address.
- Actually the ROM data is subtracted from the contents of the accumulator, and the result is used to set the PSW flags.　This result can be referenced using conditional branch instructions.　The accumulator contents do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
CMPCB	A	[**]	<word>	D9					+6

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64　+n			2
PRn	60　+n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80　+n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CMPCB A,T16[obj] Byte ROM Comparison (Indirect With 16-Bit Base)

Function

AL - TSR:(T16 +obj)

Description

- This instruction compares ROM data (byte length) to the accumulator low byte (AL).
- The ROM data is byte data in the current table segment, with the contents of obj added to the base address of the data table (T16) as the address.
- Actually the ROM data is subtracted from the contents of the accumulator, and the result is used to set the PSW flags. This result can be referenced using conditional branch instructions. The accumulator contents do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
CMPCB	A	T16[**]	<word>	F6	T16L	T16H			+10

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

CMPCB A,Tadr

Byte ROM Comparison (Direct)

Function

AL - TSR:Tadr

Description

- This instruction compares ROM data (byte length) to the accumulator low byte (AL).
- The ROM data is the byte data in the current table segment indicated by Tadr.
- Actually the ROM data is subtracted from the contents of the accumulator, and the result is used to set the PSW flags. This result can be referenced using conditional branch instructions. The accumulator contents do not change.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
CMPCB	A	Tadr	<dumyB>	B6	TadrL	TadrH			12

CPL C

Complement Carry

Function

$C \leftarrow \overline{C}$

Description

- This instruction complements the carry flag.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
CPL	C	FD						2

DEC A

Word Decrement

Function

A←A-1

Description

- This instruction decrements the word-length accumulator by 1.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
DEC	A	DC						2

DEC obj — Word Decrement

Function

obj←obj-1

Description

- This instruction decrements the word-length obj by 1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
DEC	PRn	50 +n						3

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd			
DEC	**	<word>	D6					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

DECB A

Byte Decrement

Function

AL←AL-1

Description

- This instruction decrements the accumulator low byte (AL) by 1.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
DECB	A	DC						2

DECB　obj

Byte Decrement

Function

obj←obj-1

Description

- This instruction decrements the byte-length obj by 1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
DECB	Rn	(n=0-3)	D0 +n						3

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	prefix	+1st	+2nd	+3rd			(Internal)
DECB	*	<byte>	D6					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68　+n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80　+n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

DI

Disable Interrupts

Function

MIE←0

Description

- This instruction disables all maskable interrupts.
- This instruction resets MIE (mask interrupt enable flag: PSW bit 8) to 0.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
DI			DA						2

DIV obj

Word Division

Function

<A,ER0> ← <A,ER0> ÷ obj
ER1 ← <A,ER0> mod obj

Description

- This instruction divides a 32-bit number by a 16-bit number, giving a 32-bit quotient and 16-bit remainder.
- The dividend is 32 bits, formed with the accumulator (A) as the upper word and extended local register 0 (ER0) as the lower word. The divisor is the word data indicated by obj. For the results of the division, the quotient is stored in the A and ER0 pair, and the remainder is stored in extended local register 1 (ER1).
- This instruction functions differently than previous devices (nX-8/100-400) in the way registers are used. Care should be exercised.
- IF the divisor is 0, the carry flag will be set to 1. In this case, the quotient and the remainder will be undefined.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*				

Flags affecting instruction execution

DD

C :The carry flag will be 1 if the divisor is 0, and will be 0 otherwise.
Z :The zero flag will be 1 if the quotient is 0, and will be 0 otherwise.

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd			
DIV	**	<word>	A8					+42

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

DIVB obj

Byte Division

Function

A ← A ÷ obj
R1 ← A mod obj

Description

- This instruction divides a 16-bit number by a 8-bit number, giving a 16-bit quotient and 8-bit remainder.
- The dividend is the 16-bit accumulator (A). The divisor is the byte data indicated by obj. For the results of the division, the quotient is stored in A, and the remainder is stored in local register 1 (R1).
- IF the divisor is 0, the carry flag will be set to 1. In this case, the quotient and the remainder will be undefined.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*				

Flags affecting instruction execution

DD

C : The carry flag will be 1 if the divisor is 0, and will be 0 otherwise.
Z : The zero flag will be 1 if the quotient is 0, and will be 0 otherwise.

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd			
DIVB	*	<byte>	A8					+22

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

DIVQ obj

Word Quick Division

Function

A ← <A,ER0> ÷ obj
ER1 ← <A,ER0> mod obj

Description

- This instruction divides a 32-bit number by a 16-bit number, giving a 16-bit quotient and 16-bit remainder.
- The dividend is 32 bits, formed with the accumulator (A) as the upper word and extended local register 0 (ER0) as the lower word. The divisor is the word data indicated by obj. For the results of the division, the quotient is stored in A, and the remainder is stored in extended local register 1 (ER1).
- Except for when the quotient needs more than 16-bit precision, this instruction is functionally the same as the "DIV obj" instruction, but execution time is approximately half.
- IF the divisor is 0, the carry flag will be set to 1. In this case, the quotient and the remainder will be undefined.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*		*		

Flags affecting instruction execution

DD

C : The carry flag will be 1 if the divisor is 0, and will be 0 otherwise.
Z : The zero flag will be 1 if the quotient is 0, and will be 0 otherwise. However, it is undefined when OV is 1.
OV: The overflow flag will be 1 if the quotient is greater than 65535, and will be 0 otherwise.

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd			
DIVQ	**	<word>	FB					+24

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

DJNZ obj,radr

Loop

Function

obj←obj-1
if obj ≠ 0 then PC←radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127

Description

- This instruction implements a loop process with obj as the counter.
- This instruction decrements the byte-length obj. If the result is not 0, then control will jump to the address indicated by radr. A loop count of up to 256 times can be implemented.
- The jump range possible with the loop instruction is -128 to +127 bytes of the first address of the next instruction.
- Use of local register R4 or R5 can make the instruction more efficient (fewer bytes), by allowing jumps only to lower addresses. The jump range possible with this loop instruction is -128 to -1 bytes of the first address of the next instruction. The assembler chooses the optimal instruction.

Example) Assembler selection

```
LOOP:                              ; R0, R4, and R5 are loop counters
        DJNZ      R4,LOOP          ; 2-byte instruction
        DJNZ      R0,LOOP          ; 3-byte instruction
        DJNZ      R5,NEXT          ; 3-byte instruction
NEXT:
```

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
DJNZ	R4	radr	05	rdiff7					7/10
	R5		05	rdiff7+80					7/10
	X1L		60	EA	rdiff8				7/10
	X2L		61	EA	rdiff8				7/10
	DPL		62	EA	rdiff8				7/10
	USPL		63	EA	rdiff8				7/10

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
DJNZ	*	radr	<byte>	EA	rdiff8				+5/8

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

EI Enable Interrupts

Function

MIE←1

Description

- This instruction enables maskable interrupts.
- This instruction sets MIE (mask interrupt enable flag: PSW bit 8) to 1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
EI			DB						2

EXTND

Byte to Word Sign Extend

Function

$A_{15\text{-}7} \leftarrow A_7$
$DD \leftarrow 1$

Description

- This instruction sign extends the contents of the accumulator low byte (AL) to 16 bits. The extended result is returned to the accumulator (A).
- The actual operation copies bit 7 of A to bits 8-15. At the same time the data descriptor (DD) is set to 1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
		*			1

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
EXTND		FC						2

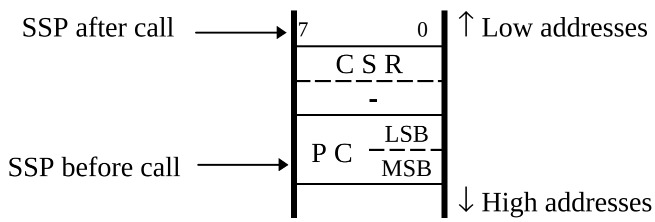
FCAL Fadr 24-Bit Space (16M Bytes: Entire Program Area) Direct Call

Function

(SSP)← PC+5, SSP← SSP-2
(SSP)← CSR, SSP←SSP-2,
CSR←$Fadr_{23\text{-}16}$
PC←$Fadr_{15\text{-}0}$
However, 0:0000H ≤ Fadr ≤ 0FFH:0FFFFH

Description

- This instruction calls any addresss in the entire program space that can be accessed with the nX-8/500S core.
- The first address of the subroutine is coded in Fadr. The state of the stack after execution of an FCAL instruction is shown below. Subroutines called with a FCAL instruction return using an FRT instruction.



- This instruction is executable only under the medium or large memory models.
- If this instruction is executed under the small or compact memory models, then an op-code trap (reset) will occur.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
FCAL	Fadr	07	08	FadrL	FadrM	FadrH		13

FILL A

Word Fill

Function

A←0FFFFH

Description

- This instruction fills the accumulator with 0FFFFH.
- This instruction also sets DD to 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
FILL	A		BC	D7					4

FILL obj

Word Fill

Function

obj←0FFFFH

Description

・This instruction fills the word-length obj with 0FFFFH.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd	+4th	+5th	
FILL	**	<word>	D7					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

FILLB A

Byte Fill

Function

AL←0FFH

Description

- This instruction fills the accumulator low byte (AL) with 0FFH.
- This instruction also sets DD to 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
FILLB	A	BC	D7					4

FILLB obj

Byte Fill

Function

obj←0FFH

Description

・ This instruction fills the byte-length obj with 0FFH.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd	+4th	+5th	
FILLB	**	<byte>	D7					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

FJ Fadr

24-Bit Space (16M Bytes: Entire Program Area) Direct Jump

Function

$CSR \leftarrow Fadr_{23-16}$
$PC \leftarrow Fadr_{15-0}$
However, 0:0000H ≤ Fadr ≤ 0FFH:0FFFFH

Description

- This instruction jumps to any addresss in the entire program space that can be accessed with the nX-8/500S core.
- The jump address is coded in Fadr.
- This instruction is executable only under the medium or large memory models. If it is executed under the small or compact memory models, then an op-code trap (reset) will occur.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
FJ	Fadr	<dumyW>	FA	FadrL	FadrM	FadrH		11

FRT　Return From Far Subroutine

Function

SSP←SSP+2, CSR←(SSP)
SSP←SSP+2, PC←(SSP)

Description

- This instruction returns from a far subroutine.
- This instruction is used to return from an FCAL (24-bit space direct call) or VCAL (vector call) instruction.
- This instruction is executable only under the medium or large memory models.　If this instruction is executed under the small or compact memory models, then an op-code trap (reset) will occur.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
FRT		07	09					9

INC A

Word Increment

Function

A←A+1

Description

- This instruction increments the word-length accumulator by 1.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
INC	A	CC						2

INC obj

Word Increment

Function

obj←obj+1

Description

- This instruction increments the word-length obj by 1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
INC	PRn	40 +n						3

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	prefix	+1st	+2nd	+3rd			(Internal)
INC	**	<word>	C6					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

INCB A

Byte Increment

Function

AL←AL+1

Description

- This instruction increments the accumulator low byte (AL) by 1.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
INCB	A		CC						2

INCB obj

Byte Increment

Function

obj←obj+1

Description

・ This instruction increments the byte-length obj by 1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
INCB	Rn (n=0-3)	C0 +n						3

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	prefix	+1st	+2nd	+3rd			(Internal)
INCB	*	<byte>	C6					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

J Cadr 64K-Byte Space (Within Current Physical Code Segment) Direct Jump

Function

PC←Cadr
However, CSR:0000H ≤ Cadr ≤ CSR:0FFFFH

Description

- This instruction jumps to any addresss in the 64K bytes in the current physical segment.
- The jump address is coded in Cadr. The jump address must exist within the current physical segment.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
J	Cadr	03	CadrL	CadrH				7

J [obj] 64K-Byte Space (Within Current Physical Code Segment) Indirect Jump

Function

PC←obj

Description

- This instruction is a 64K-byte space indirect jump based on the contents of obj (word length).
- This instruction jumps to any addresss in the 64K bytes in the current physical segment.
- obj is the word-length contents of data memory or a register. The jump address must be set in obj prior to executing this instruction. The jump address must exist within the current physical segment.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd			
J	[**]	<word>	C9					+4

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

JBR obj.bit,radr

Bit Test and Jump

Function

if obj.bit=0 then PC←radr

However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127

Description

- This instruction jumps if the bit specified by obj.bit is 0.
- The bit tested is at the bit position specified by bit within the one byte of data specified by obj.
- The jump range possible with this jump instruction is -128 to +127 bytes of the first address of the next instruction.

Example)

```
        JBR     A.5, LABEL              ; bit 5 of AL (accumulator low byte)
        JBR     R0.7, LABEL             ; bit 7 of R0 (local register 0)
        JBR     [DP].1, LABEL           ; bit 1 of data specified by DP (data pointer)
        JBR     BIT_SYM, LABEL          ; bit indicated by BIT_SYM (user-defined bit symbol)
        JBR     sbafix BIT_FIX, LABEL   ; bit indicated by BIT_FIX (user-defined fixed page bit symbol)
        JBR     sbaoff BIT_OFF, LABEL   ; bit indicated by BIT_OFF (user-defined current page bit symbol)
LABEL:
```

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
JBR	sbafix	radr	58 +bit	sbafix6+C0	rdiff8				6/9
	sbaoff		48 +bit	sbaoff6+C0	rdiff8				6/9

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
JBR	∗.bit	radr	<byte>	20 +bit	rdiff8				+4/8

∗	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

JBRS obj.bit,radr

Bit Test and Jump (With Bit Set)

Function

if obj.bit=0 then obj.bit←1,PC←radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127

Description

- This instruction jumps if the bit specified by obj.bit is 0, and sets that bit to 1.
- The bit tested is at the bit position specified by bit within the one byte of data specified by obj.
- The jump range possible with this jump instruction is -128 to +127 bytes of the first address of the next instruction.

Example)

```
    JBRS    A.5, LABEL                  ; bit 5 of AL (accumulator low byte)
    JBRS    R0.7, LABEL                 ; bit 7 of R0 (local register 0)
    JBRS    [DP].1, LABEL               ; bit 1 of data specified by DP (data pointer)
    JBRS    BIT_SYM, LABEL              ; bit indicated by BIT_SYM (user-defined bit symbol)
    JBRS    sbafix BIT_FIX, LABEL       ; bit indicated by BIT_FIX (user-defined fixed page bit symbol)
    JBRS    sbaoff BIT_OFF, LABEL       ; bit indicated by BIT_OFF (user-defined current page bit symbol)
LABEL:
```

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
JBRS	*.bit	radr	<byte>	30 +bit	rdiff8				+4/10

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

JBS obj.bit,radr — Bit Test and Jump

Function

if obj.bit=0 then PC←radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127

Description

- This instruction jumps if the bit specified by obj.bit is 1.
- The bit tested is at the bit position specified by bit within the one byte of data specified by obj.
- The jump range possible with this jump instruction is -128 to +127 bytes of the first address of the next instruction.

Example)

```
    JBS    A.5, LABEL                 ; bit 5 of AL (accumulator low byte)
    JBS    R0.7, LABEL                ; bit 7 of R0 (local register 0)
    JBS    [DP].1, LABEL              ; bit 1 of data specified by DP (data pointer)
    JBS    BIT_SYM, LABEL             ; bit indicated by BIT_SYM (user-defined bit symbol)
    JBS    sbafix BIT_FIX, LABEL      ; bit indicated by BIT_FIX (user-defined fixed page bit symbol)
    JBS    sbaoff BIT_OFF, LABEL      ; bit indicated by BIT_OFF (user-defined current page bit symbol)
LABEL:
```

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
JBS	sbafix	radr	58 +bit	sbafix6+80	rdiff8				6/9
	sbaoff		48 +bit	sbaoff6+80	rdiff8				6/9

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
JBS	∗.bit	radr	<byte>	28 +bit	rdiff8				+4/8

∗	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

JBSR obj.bit,radr

Bit Test and Jump (With Bit Reset)

Function

if obj.bit=1 then obj.bit←0,PC←radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127

Description

- This instruction jumps if the bit specified by obj.bit is 1, and resets that bit to 0.
- The bit tested is at the bit position specified by bit within the one byte of data specified by obj.
- The jump range possible with this jump instruction is -128 to +127 bytes of the first address of the next instruction.

Example)

```
    JBSR    A.5, LABEL              ; bit 5 of AL (accumulator low byte)
    JBSR    R0.7, LABEL             ; bit 7 of R0 (local register 0)
    JBSR    [DP].1, LABEL           ; bit 1 of data specified by DP (data pointer)
    JBSR    BIT_SYM, LABEL          ; bit indicated by BIT_SYM (user-defined bit symbol)
    JBSR    sbafix BIT_FIX, LABEL   ; bit indicated by BIT_FIX (user-defined fixed page bit symbol)
    JBSR    sbaoff BIT_OFF, LABEL   ; bit indicated by BIT_OFF (user-defined current page bit symbol)
LABEL:
```

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
JBSR	∗.bit	radr	<byte>	38 +bit	rdiff8				+4/10

∗	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

JC *cond*,radr
J*cond* radr

Conditional Jump

Function

if cond is true then PC←radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127

Description

- This instruction jumps if the condition specified by cond is true.
- The condition is indicated by the flag state remaining in the PSW (program status word). Therefore, this instruction presumes prior execution of an instruction that leaves its result in the PSW (comparison, etc.). This instruction is then used to evaluate that result.
- The cond can be coded as an operand or as part of the mnemonic string.

Example)

```
        CMP     A,#9
        JC      GT, LABEL           ; cond is operand
        CMP     A,#0FH
        JLE     LABEL               ; cond is part of mnemonic string
         :
LABEL:
```

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax				Instruction Code			
J*cond*	JC *cond*	Meaning	Flag Conditions	1st	2nd	3rd	
JGT	JC GT	unsigned>	(Z=0) ∩ (C=0)	F0	rdiff8		4/6
JGE JNC	JC GE JC NC	unsigned≥	C=0	F5	rdiff8		4/6
JLT JCY	JC LT JC CY	unsigned<	C=1	F2	rdiff8		4/6
JLE	JC LE	unsigned≤	(Z=1) ∪ (C=1)	F7	rdiff8		4/6
JEQ JZ	JC EQ JC ZF	=	Z=1	F1	rdiff8		4/6
JNE JNZ	JC NE JC NZ	≠	Z=0	F6	rdiff8		4/6
JGTS	JC GTS	signed>	((OV $\veebar$ S) ∪ Z)=0	<dumyB>	FE	rdiff8	6/10
JGES	JC GES	signed≥	(OV $\veebar$ S) =0	<dumyB>	FF	rdiff8	6/10
JLTS	JC LTS	signed<	(OV $\veebar$ S) =1	<dumyB>	FC	rdiff8	6/10
JLES	JC LES	signed≤	((OV $\veebar$ S) ∪ Z)=0	<dumyB>	FD	rdiff8	6/10
JPS	JC PS	positive	S=0	F4	rdiff8		4/6
JNS	JC NS	negative	S=1	F3	rdiff8		4/6
JOV	JC OV	overflow	OV=1	9A	28 +1	rdiff8	6/10
JNV	JC NV	no overflow	OV=0	9A	20 +1	rdiff8	6/10

JRNZ DP, radr

Loop

Function

DPL←DPL-1
if DPL ≠ 0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127

Description

- This instruction implements a loop process with the data pointer low byte (DPL) as the counter.
- This instruction decrements the contents of DPL. If the result is not 0, then control will jump to the address indicated by radr. A loop count of up to 256 times can be implemented.
- The jump range possible with the loop instruction is -128 to +127 bytes of the first address of the next instruction.

This instruction is completely identical to "DJNZ DPL,radr". It is provided to support source level compatibility with nX-8/100-400.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
JRNZ	DP	radr	62	EA	rdiff8				6/11

L A,obj Word Load

Function

A←obj
DD←1

Description

- This instruction loads the contents of obj (word length) into the accumulator (A).
- Execution of this instruction sets DD to 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				1

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
L	A	#N16	F8	N16L	N16H				6
		ERn	74 +n						2
		PRn	70 +n						2
		[X1]	80						4
		[DP]	82						4
		[DP−]	81						5
		[DP+]	83						5
		fix	84	fix8					4
		off	85	off8					4
		sfr	86	sir8					4
		dir	87	dirL	dirH				6
		D16[X1]	88	D16L	D16H				6
		n7[USP]	89	n7					6
		n7[DP]	89	80 +n7					6

LB A,obj

Byte Load

Function

AL←obj
DD←0

Description

- This instruction loads the contents of obj (byte length) into the accumulator low byte (AL).
- Execution of this instruction sets DD to 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				0

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
LB	A	#N8	F9	N8					4
		Rn	78 +n						2
		[X1]	90						4
		[DP]	92						4
		[DP−]	91						5
		[DP+]	93						5
		fix	94	fix8					4
		off	95	off8					4
		sfr	96	sir8					4
		dir	97	dirL	dirH				6
		D16[X1]	98	D16L	D16H				6
		n7[USP]	99	n7					6
		n7[DP]	99	80 +n7					6

LC A,[obj]

Word ROM Load (Indirect)

Function

A ← TSR:(obj)

Description

- This instruction loads ROM data (word length) into the accumulator (A).
- The ROM data is word data in the current table segment, with the contents of obj as the address.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
LC	A	[**]	<word>	DA					+9

**	<word> Word Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	BC			2
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

LC A,T16[obj] Word ROM Load (Indirect With 16-Bit Base)

Function

A ← TSR:(T16 + obj)

Description

- This instruction loads ROM data (word length) into the accumulator (A).
- The ROM data is word data in the current table segment, with the contents of obj added to the base address of the data table (T16) as the address.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
LC	A	T16[**]	<word>	E7	T16L	T16H			+13

**	<word> Word Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	BC			2
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

LC A,Tadr

Word ROM Load (Direct)

Function

A ← TSR:Tadr

Description

- This instruction loads ROM data (word length) into the accumulator (A).
- The ROM data is the word data in the current table segment indicated by Tadr.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
LC	A	Tadr	<word>	B7	TadrL	TadrH			15

LCB A,[obj] — Byte ROM Load (Indirect)

Function

AL ← TSR:(obj)

Description

- This instruction loads ROM data (byte length) into the accumulator low byte (AL).
- The ROM data is byte data in the current table segment, with the contents of obj as the address.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
LCB	A	[**]	<word>	DB					+6

**	<word> Word Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	BC			2
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

LCB A,T16[obj] Byte ROM Load (Indirect With 16-Bit Base)

Function

AL ← TSR:(T16 + obj)

Description

- This instruction loads ROM data (byte length) into the accumulator low byte (AL).
- The ROM data is byte data in the current table segment, with the contents of obj added to the base address of the data table (T16) as the address.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
LCB	A	T16[**]	<word>	F7	T16L	T16H			+10

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

LCB A,Tadr

Byte ROM Load (Direct)

Function

AL ← TSR:Tadr

Description

- This instruction loads ROM data (byte length) into the accumulator low byte (AL).
- The ROM data is the byte data in the current table segment indicated by Tadr.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
LCB	A	Tadr	<dumyB>	B7	TadrL	TadrH			12

MAC

Multiply-Addition Calculation

Function

MAC start bit←1
(SB sfr MAC start bit)

Description

- This instruction starts multiply-addition calculations. It can be executed only with target devices in which a multiply-addition calculation circuit exists as an SFR.
- Refer to the appropriate hardware manual for more detailed information of MAC instruction function.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
MAC		B6	*1	08 +bit				4

*1 is the byte address of the MAC start bit.
Bit is the bit position of MAC start bit

MB C,obj.bit — Move Bit

Function

C ← obj.bit

Description

- This instruction moves the contents of the bit specified by bit in obj (byte length) to the carry flag (C).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
MB	C	*.bit	<byte>	10 +bit					+3

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

MB obj.bit,C

Move Bit

Function

obj.bit ← C

Description

- This instruction moves the contents of the carry flag (C) to the bit specified by bit in obj (byte length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	(Internal)
MB	∗.bit	C	<byte>	18 +bit					+3

∗	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

MBR C,obj

Move Bit (Register Indirect Bit Specification)

Function

C ← obj.(AL)

Description

- This instruction moves the contents of the bit at the specified position within the bit block to the carry flag (C).
- The bit block is the block of 256 bits starting from the address obj. A byte addressing specification is coded in obj.
- The bit position is 0-255, specified by the contents of the accumulator low byte (AL).
- The same instruction coded for nX-8/100-400 has a different function. For nX-8/100-400, only the lower 3 bits of AL are valid for the bit position specification. In this case, only the 8 bits of obj can specified as the target bit.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
MBR	C	*	<byte>	BA					+6

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

MBR obj,C

Move Bit (Register Indirect Bit Specification)

Function

obj.(AL) ← C

Description

- This instruction moves the contents of the carry flag to the bit at the specified position within the bit block.
- The bit block is the block of 256 bits starting from the address obj. A byte addressing specification is coded in obj.
- The bit position is 0-255, specified by the contents of the accumulator low byte (AL).
- The same instruction coded for nX-8/100-400 has a different function. For nX-8/100-400, only the lower 3 bits of AL are valid for the bit position specification. In this case, only the 8 bits of obj can specified as the target bit.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
MBR	*	C	<byte>	BB					+5

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

MOV obj1,obj2 Word Move

Function

obj1 ← obj2

Description

- This instruction moves a word of data from obj1 to obj2.
- The address of the source word is coded in obj1.
- The address of the destination word is coded in obj2.
- Difference with nX-8/100-400:
  the instruction "MOV A,obj" does not modify the data descriptor (DD).
  For DD in nX-8/100-400, "MOV A,obj" is handled the same as an L instruction (that is, DD is set to 1). For DD in nX-8/500S, DD does not change. DD switching by the MOV instruction has been eliminated.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

However, all flags will change if PSW is the destination.

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
MOV	ERn	#N16	24 +n	N16L	N16H				6
	PRn		20 +n	N16L	N16H				6
	off		C7	off8	N16L	N16H			8
	sfr		C6	sfr8	N16L	N16H			8
	LRB		C6	02	N16L	N16H			8

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
MOV	A	**	<word>	97					+2
	ERn		<word>	70 +n					+2
	PRn		<word>	74 +n					+2
	[X1]		<word>	88					+4
	[DP]		<word>	8A					+4
	[DP–]		<word>	89					+5
	[DP+]		<word>	8B					+5
	fix		<word>	86	fix8				+4
	off		<word>	87	off8				+4
	sfr		<word>	96	sfr8				+4
	PSW		<word>	96	04				+4
	SSP		<word>	96	00				+4
	LRB		<word>	96	02				+4
	dir		<word>	9B	dirL	dirH			+6
	D16[X1]		<word>	98	D16L	D16H			+6
	D16[X2]		<word>	99	D16L	D16H			+6
	n7[Dp]		<word>	9A	n7				+6
	n7[USP]		<word>	9A	80 +n				+6
	[X1+A]		<word>	F8					+6
	[X1+R0]		<word>	F9					+6
	**	A	<word>	AA					+2
		#N16	<word>	AB	N16L	N16H			+6

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

MOVB obj1,obj2 Byte Move

Function

obj1 ← obj2

Description

- This instruction moves a byte of data from obj1 to obj2.
- The address of the source byte is coded in obj1.
- The address of the destination byte is coded in obj2.
- Difference with nX-8/100-400:
  the instruction "MOVB A,obj" does not modify the data descriptor (DD).
  For DD in nX-8/100-400, "MOVB A,obj" is handled the same as an LB instruction (that is, DD is set to 0). For DD in nX-8/500S, DD does not change. DD switching by the MOVB instruction has been eliminated.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

However, all flags will change if PSW is the destination.

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
MOVB	Rn	#N8	10 +n	NB					4
	off		D7	off8	N8				6
	sfr		D6	sfr8	N8				6

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
MOVB	A	*	<byte>	97					+2
	Rn		<byte>	70 +n					+2
	[X1]		<byte>	88					+4
	[DP]		<byte>	8A					+4
	[DP–]		<byte>	89					+5
	[DP+]		<byte>	8B					+5
	fix		<byte>	86	fix8				+4
	off		<byte>	87	off8				+4
	sfr		<byte>	96	sfr8				+4
	PSWL		<byte>	96	04				+4
	PSWH		<byte>	96	05				+4
	dir		<byte>	9B	dirL	dirH			+6
	D16[X1]		<byte>	98	D16L	D16H			+6
	D16[X2]		<byte>	99	D16L	D16H			+6
	n7[DP]		<byte>	9A	n7				+6
	n7[USP]		<byte>	9A	80 +n7				+6
	[X1+A]		<byte>	F8					+6
	[X1+R0]		<byte>	F9					+6
	*	A	<byte>	AA					+2
		#N8	<byte>	AB	N8				+4

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

MUL obj

Word Multiplication

Function

<A,ER0> ← A×obj

Description

- This instruction multiplies a 16-bit number by a 16-bit number, giving a 32-bit product.
- The multiplicand is the contents of the accumulator (A). The multiplier is the word data indicated by obj. For the results of the multiplication, the product is stored in the A and ER0 pair.
- Refer to the appropriate hardware manual for with or without of multiplier circuit.
- Difference with nX-8/100-400:

  Word addressing can be coded in the multiplier. This has to be a fixed register for nX-8/100-400.

  The registers that store the high and low words of the product are different.

  nX-8/500S : <A,ER0> ← A × obj

  nX-8/100-400 : <ER1,A> ← A × ER0

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Z: The zero flag will be 1 if the product is 0, and will be 0 otherwise.

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
MUL	**	(Without Multiplier)	<word>	A9					+21
MUL	**	(With Multiplier)	<word>	A9					+3

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

MULB obj

Byte Multiplication

Function

A ← AL × obj

Description

- This instruction multiplies an 8-bit number by an 8-bit number, giving a 16-bit product.
- The multiplicand is the contents of the accumulator low byte (AL). The multiplier is the byte data indicated by obj. For the results of the multiplication, the product is stored in the accumulator (A).
- Refer to the appropriate hardware manual for with or without of multiplier circuit.
- Difference with nX-8/100-400:
  Byte addressing can be coded in the multiplier. This has to be a fixed register for nX-8/100-400.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Z: The zero flag will be 1 if the product is 0, and will be 0 otherwise.

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
MULB	*	(Without Multiplier)	<byte>	A9					+12
MULB	*	(With Multiplier)	<byte>	A9					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

NEG A

Word Negate Sign

Function

A ← -A

Description

- This instruction takes the 2's complement of the contents of the accumulator (A), and returns the results in A.
- Execution of this instruction sets DD to 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
NEG	A	CF						3

NEGB A

Byte Negate Sign

Function

A ← -AL

Description

- This instruction takes the 2's complement of the contents of the accumulator low byte (AL), and returns the results in AL.
- Execution of this instruction sets DD to 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
NEG	A	CF						3

NOP

No Operation

Function

NO OPERATION

Description

- This instruction just consumes a fixed number of cycles and moves the program counter to the next instruction.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
NOP			00						2

OR A,obj

Word Logical OR

Function

A ← A ∪ obj

Description

- This instruction takes the word logical OR of the contents of obj (word length) and the accumulator (A), and stores the result in the accumulator.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
OR	A	off	CD	off8					4
		#N16	CE	N16L	N16H				6

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
OR	A	**	<word>	C5					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

OR obj1,obj2

Word Logical OR

Function

obj1 ← obj1 ∪ obj2

Description

- This instruction takes the word logical OR of the contents of obj1 (word length) and obj2 (word length), and stores the result in obj1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
OR	**	fix	<word>	C0	fix8				+5
		off	<word>	C1	off8				+5
		sfr	<word>	C2	sfr8				+5
		#N16	<word>	C3	N16L	N16H			+6
		A	<word>	C4					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

ORB A,obj

Byte Logical OR

Function

AL ← AL ∪ obj

Description

- This instruction takes the word logical OR of the contents of obj (byte length) and the accumulator low byte (AL), and stores the result in the accumulator.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ORB	A	off	CD	off8					4
		#N8	CE	N8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
ORB	A	*	<byte>	C5					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

ORB obj1,obj2 Byte Logical OR

Function

obj1 ← obj1 ∪ obj2

Description

- This instruction takes the word logical OR of the contents of obj1 (byte length) and obj2 (byte length), and stores the result in obj1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
ORB	*	fix	<byte>	C0	fix8				+5
		off	<byte>	C1	off8				+5
		sfr	<byte>	C2	sfr8				+5
		#N8	<byte>	C3	N8				+4
		A	<byte>	C4					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

POPS register_list

Pop Off System Stack

Function

Register group←System stack

SSP ← SSP+n (n:number of popped registers×2)

Description

- This instruction pops data off the system stack to the group of registers specified by the register_list.
- The register_list can be one of the following:

(1)Extended local register list
(2)Pointing register list
(3)Control register list
(4)ER
(5)PR
(6)CR

A list of register names is coded for (1), (2), or (3).
An extended local register list must be one or more of ER0, ER1, ER2, and ER3. A pointing register list must be one or more of X1, X2, DP, and USP. A control register list must be one or more of A, LRP, and PSW.
When two or more registers are specified in one of these three ways, they should be delimited by commas. The registers can be coded in any order in the operand, but the order in which they are popped and written is fixed.

For (4), (5), and (6), the symbols indicate register sets.
POPS ER" is equivalent to "POPS ER0,ER1,ER2,ER3." "POSPS PR" is equivalent to "POPS X1,X2,DP,USP." "POPS CR" is equivalent to "POPS A,LRB,PSW." The popping sequence for local registers is ER0→ER1→ER2→ER3. For pointer registers, it is X1→X2→DP→USP. For control registers, it is PSW→LRB→A.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	*

Flags affecting instruction execution

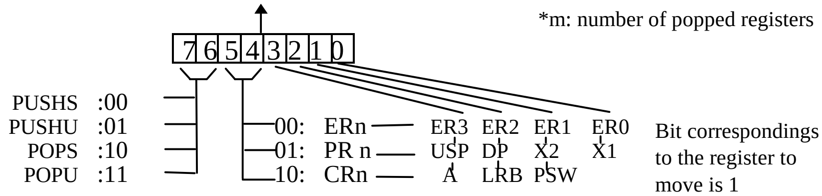
DD

C, Z, S, OV, HC, DD are all changed only when the PSW is popped. They are unchanged in all other cases.

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
POPS	register_list	06	register mask value					5+4m*

*m: number of popped registers



PUSHS register_list

Push On System Stack

Function

Register group←System stack
SSP←SSP+n (n:number of popped registers×2)

Description

- This instruction pops data off the system stack to the group of registers specified by the register_list.
- The register_list can be one of the following:

(1)Extended local register list
(2)Pointing register list
(3)Control register list
(4)ER
(5)PR
(6)CR

A list of register names is coded for (1), (2), or (3).
An extended local register list must be one or more of ER0, ER1, ER2, and ER3. A pointing register list must be one or more of X1, X2, DP, and USP. A control register list must be one or more of A, LRP, and PSW.
When two or more registers are specified in one of these three ways, they should be delimited by commas. The registers can be coded in any order in the operand, but the order in which they are popped and written is fixed.

For (4), (5), and (6), the symbols indicate register sets.
"PUSHS ER" is equivalent to "PUSHS ER0,ER1,ER2,ER3." "POSPS PR" is equivalent to " PUSHS X1,X2,DP,USP." " PUSHS CR" is equivalent to " PUSHS A,LRB,PSW." The popping sequence for local registers is ER3→ER2→ER1→ER0. For pointer registers, it is USP→DP→X2→X1. For control registers, it is A→LRB→PSW.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

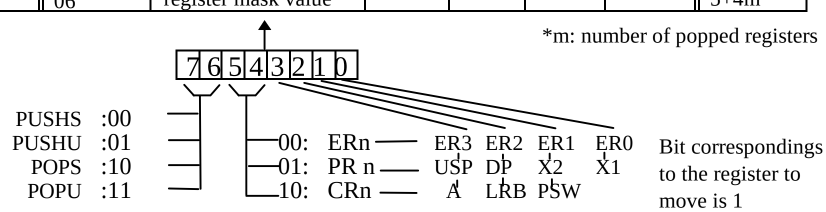
DD

C, Z, S, OV, HC, DD are all changed only when the PSW is popped. They are unchanged in all other cases.

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
PUSHS	A	07						3
PUSHS	register_list	06	register mask value					5+4m*

*m: number of popped registers



RB obj.bit Reset Bit (Bit Position Direct Specification)

Function

if obj.bit = 0 then Z←1 else Z←0
obj.bit←0

Description

- This instruction resets to 0 the contents of the bit specified by bit in obj (byte length).
- Byte addressing is coded in obj.
- Before resetting the specified bit, this instruction examines its contents and sets the zero flag (Z). If the specified bit is 0 before instruction execution, then Z will be set to 1; if the bit is 1, then Z will be reset to 0.
- For bits in particular areas, this instruction can be executed more effectively with sbafix/sbaoff addressing. Please see the chapter that explains addressing for details.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
RB	sbafix	58 +bit	sbafix6+40					4
	sbaoff	48 +bit	sbaoff6+40					4

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	prefix	+1st	+2nd	+3rd			(Internal)
RB	*.bit	<byte>	00 +bit					+3

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

RBR obj

Reset Bit (Register Indirect Bit Specification)

Function

if obj.(AL) = 0 then Z←1 else Z←0
obj.(AL) ←0

Description

- This instruction resets to 0 the contents of the bit at the specified position within the bit block.
- The bit block is the block of 256 bits starting from the address obj. A byte addressing specification is coded in obj.
- The bit position is 0-255, specified by the contents of the accumulator low byte (AL).
- Before resetting the specified bit, this instruction examines its contents and sets the zero flag (Z). If the specified bit is 0 before instruction execution, then Z will be set to 1; if the bit is 1, then Z will be reset to 0.
- The same instruction coded for nX-8/100-400 has a different function. For nX-8/100-400, only the lower 3 bits of AL are valid for the bit position specification. In this case, only the 8 bits of obj can specified as the target bit.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd	+4th	+5th	
RBR	*	<byte>	B9					+5

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

RC

Reset Carry

Function

C ← 0

Description

- This instruction resets the carry flag (C) to 0.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
0					

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
RC		CA						2

RDD

Reset DD

Function

DD ← 0

Description

- This instruction resets the data descriptor (DD) to 0 (byte).
- DD is the flag that specifies how calculations with the accumulator are to be performed.
- Following this instruction, accumulator calculations will be byte length.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
					0

Flags affecting instruction execution

DD

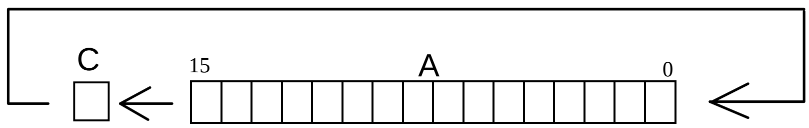
Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
RDD			D8						2

ROL A,width
ROL A

Word Left Rotate (With Carry)

Function



Description

- This instruction rotates the accumulator (A) up to 4 bits to the left, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- "ROL A" is equivalent to "ROL A,1."
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
1

Codes

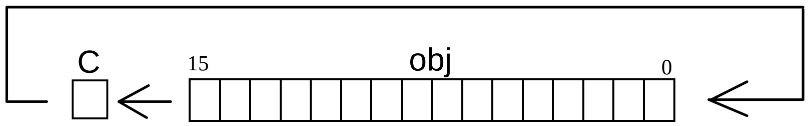
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
ROL	A		AF						2
		width	BC	AC +width					4+n*

* n=number of bits to rotate

ROL obj,width
ROL obj

Word Left Rotate (With Carry)

Function



Description

- This instruction rotates obj (word length) up to 4 bits to the left, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- "ROL obj" is equivalent to "ROL obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
ROL	**	width	<word>	AC +width					+2+n *

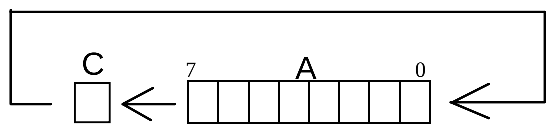
* n=number of bits to rotate

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	64 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

ROLB A,width
ROLB A

Byte Left Rotate (With Carry)

Function



Description

- This instruction rotates the accumulator low byte (AL) up to 4 bits to the left, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- "ROLB A" is equivalent to "ROLB A,1."
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
0

Codes

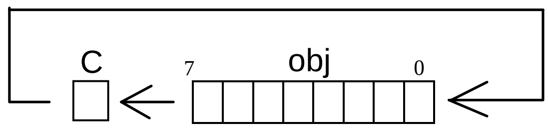
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
ROLB	A		AF						2
		width	BC	AC +width					4+n*

* n=number of bits to rotate

ROLB obj,width
ROLB obj

Byte Left Rotate (With Carry)

Function



Description

- This instruction rotates obj (byte length) up to 4 bits to the left, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- "ROLB obj" is equivalent to "ROLB obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

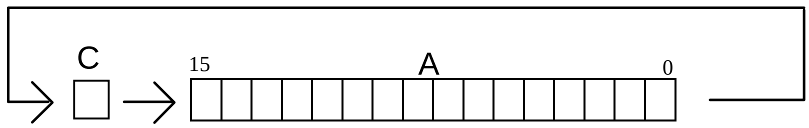
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
ROLB	*	width	<word>	AC+width					+2+n *

* n=number of bits to rotate

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

ROR A,width
ROR A
Word Right Rotate (With Carry)

Function



Description

- This instruction rotates the accumulator (A) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- "ROR A" is equivalent to "ROR A,1."
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
1

Codes

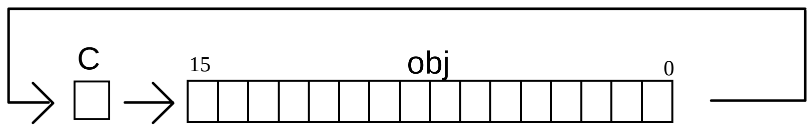
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
ROR	A		BF						2
		width	BC	BC +width					4+n *

* n=number of bits to rotate

ROR obj,width
ROR obj

Word Right Rotate (With Carry)

Function



Description

- This instruction rotates obj (word length) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- ROR obj" is equivalent to "ROR obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
ROR	**	width	<word>	BC +width					+2+n *

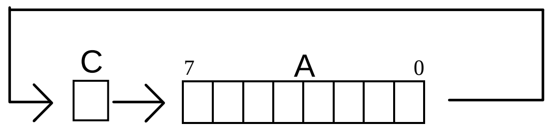
* n=number of bits to rotate

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	64 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

RORB A,width
RORB A

Byte Right Rotate (With Carry)

Function



Description

- This instruction rotates the accumulator low byte (AL) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- "RORB A" is equivalent to "RORB A,1."
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
0

Codes

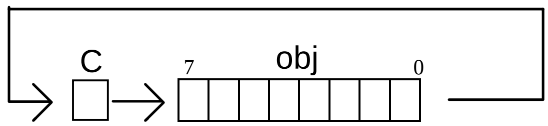
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
RORB	A		BF						2
		width	BC	BC +width					4+n *

* n=number of bits to rotate

RORB obj,width
RORB obj

Byte Right Rotate (With Carry)

Function



Description

- This instruction rotates obj (byte length) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to rotate with a value 1 to 4. One instruction can rotate a maximum of 4 bits.
- "RORB obj" is equivalent to "RORB obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
RORB	*	width	<byte>	BC+width					+2+n *

* n=number of bits to rotate

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

RT

Return From Subroutine

Function

SSP ← SSP+2
PC ← (SSP)

Description

- This instruction returns from a subroutine called by an SCAL, CAL, or ACAL instruction, or by a VCAL instruction under the small or compact memory models.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
RT		01						6

RTI

Return From Interrupt

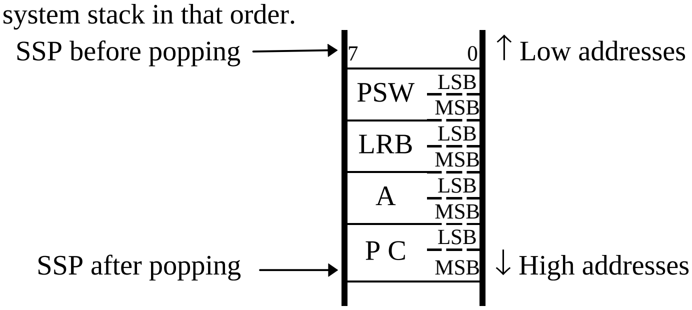
Function

1) Small/compact memory models
   SSP ← SSP+2, PSW ← (SSP)
   SSP ← SSP+2, LRB ← (SSP)
   SSP ← SSP+2, A ← (SSP)
   SSP ← SSP+2, PC ← (SSP)
2) Medium/large memory models
   SSP ← SSP+2, PSW ← (SSP)
   SSP ← SSP+2, LRB ← (SSP)
   SSP ← SSP+2, A ← (SSP)
   SSP ← SSP+2, CSR ← (SSP)
   SSP ← SSP+2, PC ← (SSP)

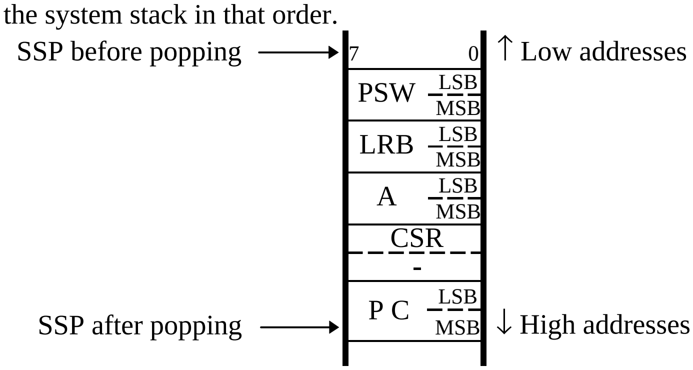
Description

- This instruction returns from an interrupt routine.

1)Under the small/compact memory models, the PSW, LRB, A, and PC are popped from the system stack in that order.



2)Under the small/compact memory models, the PSW, LRB, A, PC, and CSR are popped from the system stack in that order.



Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	*

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
RTI		02						12/14

Near/Far

SB obj.bit

Set Bit (Bit Position Direct Specification)

Function

if obj.bit = 0 then Z←1 else Z←0
obj.bit←1

Description

- This instruction sets to 1 the contents of the bit specified by bit in obj (byte length).
- Byte addressing is coded in obj.
- Before setting the specified bit, this instruction examines its contents and sets the zero flag (Z). If the specified bit is 0 before instruction execution, then Z will be set to 1; if the bit is 1, then Z will be reset to 0.
- For bits in particular areas, this instruction can be executed more effectively with sbafix/sbaoff addressing. Please see the chapter that explains addressing for details.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
SB	sbafix	58 +bit	sbafix6					4
	sbaoff	48 +bit	sbaoff6					4

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	prefix	+1st	+2nd	+3rd			(Internal)
SB	*.bit	<byte>	08 +bit					+3

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	BC			2
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SBC A,obj — Word Subtraction With Carry

Function

A←A-obj-C

Description

- This instruction performs word subtraction, subtracting the contents of obj (word length) and the carry flag from the accumulator (A).
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SBC	A	#N16	BC	E3	N16L	N16H			8

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
SBC	A	**	<word>	E5					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

SBC obj1,obj2

Word Subtraction With Carry

Function

obj1 ← obj1-obj2-C

Description

- This instruction performs word subtraction, subtracting the contents of obj2 (word length) and the carry flag from obj1 (word length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
SBC	**	fix	<word>	E0	fix8				+5
		off	<word>	E1	off8				+5
		sfr	<word>	E2	sfr8				+5
		#N16	<word>	E3	N16L	N16H			+6
		A	<word>	E4					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

SBCB　A,obj　　Byte Subtraction With Carry

Function

AL ← AL-obj-C

Description

- This instruction performs byte subtraction, subtracting the contents of obj (byte length) and the carry flag from the accumulator low byte (AL).
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SBCB	A	#N8	BC	E3	N8				6

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
SBCB	A	*	<byte>	E5					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68　+n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80　+n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SBCB obj1,obj2

Byte Subtraction With Carry

Function

obj1 ← obj1-obj2-C

Description

- This instruction performs byte subtraction, subtracting the contents of obj2 (byte length) and the carry flag from obj1 (byte length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
SBCB	*	fix	<byte>	E0	fix8				+5
		off	<byte>	E1	off8				+5
		sfr	<byte>	E2	sfr8				+5
		#N8	<byte>	E3	N8				+4
		A	<byte>	E4					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SBR obj

Set Bit (Register Indirect Bit Specification)

Function

if obj.(AL) = 0 then Z←1 else Z←0
obj.(AL) ← 1

Description

- This instruction sets to 1 the contents of the bit at the specified position within the bit block.
- The bit block is the block of 256 bits starting from the address obj. A byte addressing specification is coded in obj.
- The bit position is 0-255, specified by the contents of the accumulator low byte (AL).
- Before setting the specified bit, this instruction examines its contents and sets the zero flag (Z). If the specified bit is 0 before instruction execution, then Z will be set to 1; if the bit is 1, then Z will be reset to 0.

The same instruction coded for nX-8/100-400 has a different function. For nX-8/100-400, only the lower 3 bits of AL are valid for the bit position specification. In this case, only the 8 bits of obj can specified as the target bit.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	prefix	+1st	+2nd	+3rd	+4th	+5th	
SBR	*	<byte>	B8					+5

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SC Set Carry

Function

C ← 1

Description

- This instruction sets the carry flag (C) to 1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
1					

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
SC		CB						2

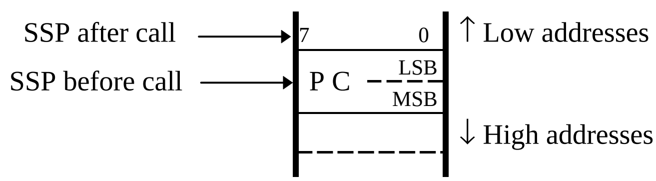
SCAL　Cadr　64K-Byte Space (Within Current Physical Code Segment) Direct Call

Function

(SSP)←PC+3
SSP←SSP-2,
PC←Cadr
However, CSR:0000H ≤ Cadr ≤ CSR:0FFFFH

Description

- This instruction is supported to provide compatibility with nX-8/100-400.　It is actually identical to the CAL instruction.
- This instruction calls any addresss in the 64K bytes in the current physical segment.
- The first address of the subroutine is coded in Cadr.　The subroutine must exist within the current physical segment.
- The state of the stack after execution of an SCAL instruction is shown below.　Subroutines called with an SCAL instruction return using an RT instruction.



Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
SCAL	Cadr	FE	CadrL	CadrH				9

SDD

Set DD

Function

DD ← 1

Description

- This instruction sets the data descriptor (DD) to 1 (word).
- DD is the flag that specifies how calculations with the accumulator are to be performed.
- Following this instruction, accumulator calculations will be word length.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
					1

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SDD			D9						2

SJ radr

Short Jump

Function

PC←radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction is a relative jump to an address found by adding a signed 8-bit displacement to a base, the first address of the next instruction.
- The jump address is coded in radr. The jump address must exist within the current physical segment.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

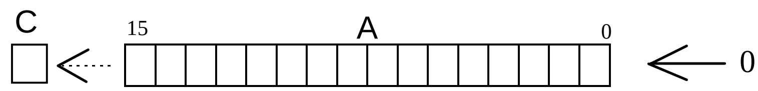
DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SJ	radr		04	rdiff8					6

SLL A,width
SLL A
Word Left Shift (With Carry)

Function



Description

- This instruction shifts the accumulator (A) up to 4 bits to the left.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SLL A" is equivalent to "SLL A,1."
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
1

C : If any of the bits carried out of bit 15 of A from the shift operation is 1, then C will be set to 1. If all carry-out bits are 0, then C will be reset to 0.

Codes

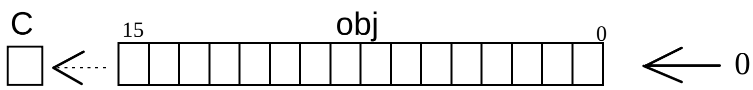
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
SLL	A		8F						2
		width	BC	8C +width					4+n*

* n=number of bits to shift

SLL obj,width
SLL obj

Word Left Shift (With Carry)

Function



Description

- This instruction shifts obj (word length) up to 4 bits to the left.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SLL obj" is equivalent to "SLL obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

C : If any of the bits carried out of bit 15 of obj from the shift operation is 1, then C will be set to 1. If all carry-out bits are 0, then C will be reset to 0.

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
SLL	**	width	<word>	8C +width					+2+n *

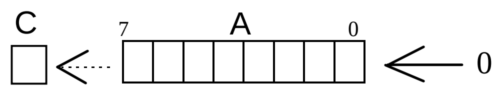
* n=number of bits to shift

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

SSLB A,width
SSLB A

Byte Left Shift (With Carry)

Function



Description

- This instruction shifts the accumulator low byte (AL) up to 4 bits to the left.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SLLB A" is equivalent to "SLLB A,1."
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
0

C : If any of the bits carried out of bit 7 of obj from the shift operation is 1, then C will be set to 1. If all carry-out bits are 0, then C will be reset to 0.

Codes

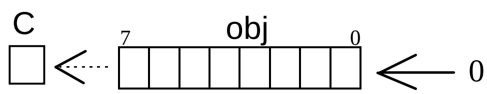
Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SLLB	A		8F						2
		width	BC	8C +width					4+n*

* n=number of bits to shift

SLLB obj,width
SLLB obj

Byte Left Shift (With Carry)

Function



Description

- This instruction shifts obj (byte length) up to 4 bits to the left.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SLLB obj" is equivalent to "SLLB obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

C : If any of the bits carried out of bit 7 of obj from the shift operation is 1, then C will be set to 1. If all carry-out bits are 0, then C will be reset to 0.

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	(Internal)
SLLB	*	width	<byte>	8C+width					+2+n *

* n=number of bits to shift

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SQR A

Word Square

Function

<A,ER0> ← A × A

Description

- This instruction squares the contents of the 16-bit accumulator (A), giving a 32-bit result.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
SQR	A	BC	A9					23

SQRB　A

Byte Square

Function

A ← AL × AL

Description

- This instruction squares the contents of the 16-bit accumulator low byte (AL), giving a 16-bit result.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

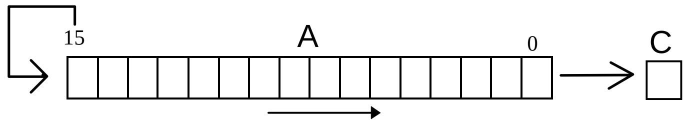
DD
0

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
SQRB	A		BC	A9					14

SRA A,width
SRA A
Word Arithmetic Right Shift (With Carry)

Function



Description

- This instruction shifts the accumulator (A) up to 4 bits to the right, including the carry flag.
- Each time one bit is shifted in an arithmetic shift, the carry-out from A_0 is entered into C and A_{15} itself is entered into A_{15}.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SRA A" is equivalent to "SRA A,1."
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
1

C : The last value carried out of A_0 will be entered in C.

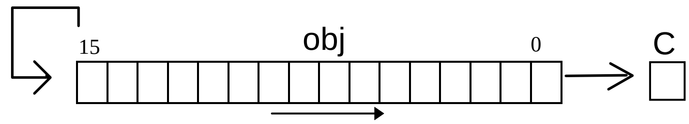
Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
SRA	A	width	BC	EC +width					4+n*

* n=number of bits to shift

SRA　obj,width
SRA　obj　　Word Arithmetic Right Shift (With Carry)

Function



Description

- This instruction shifts obj (word length) up to 4 bits to the right, including the carry flag.
- Each time one bit is shifted in an arithmetic shift, the carry-out from obj_0 is entered into C and obj_{15} itself is entered into obj_{15}.
- The width specifies the number of bits to shift with a value 1 to 4.　One instruction can shift a maximum of 4 bits.
- "SRA obj" is equivalent to "SRA obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

C : The last value carried out of A_0 will be entered in C.

Codes

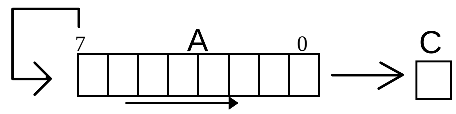
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
SRA	**	width	<word>	EC +width					+2+n*

* n=number of bits to shift

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64　+n			2
PRn	60　+n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80　+n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

SRAB A,width
SRAB A
Byte Arithmetic Right Shift (With Carry)

Function



Description

- This instruction shifts the accumulator low byte (AL) up to 4 bits to the right, including the carry flag.
- Each time one bit is shifted in an arithmetic shift, the carry-out from A_0 is entered into C and A_7 itself is entered into A_7.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SRAB A" is equivalent to "SRAB A,1."

- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
0

C : The last value carried out of A_0 will be entered in C.

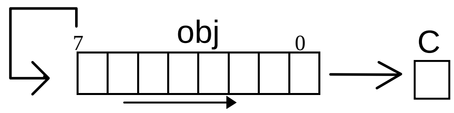
Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SRAB	A	width	BC	EC +width					4+n*

* n=number of bits to shift

SRAB　obj,width
SRAB　obj　　Byte Arithmetic Right Shift (With Carry)

Function



Description

- This instruction shifts obj (byte length) up to 4 bits to the right, including the carry flag.
- Each time one bit is shifted in an arithmetic shift, the carry-out from obj_0 is entered into C and obj_7 itself is entered into obj_7.
- The width specifies the number of bits to shift with a value 1 to 4.　One instruction can shift a maximum of 4 bits.
- "SRAB obj" is equivalent to "SRAB obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

C : The last value carried out of A_0 will be entered in C.

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
SRAB	*	width	<byte>	EC+width					+2+n *

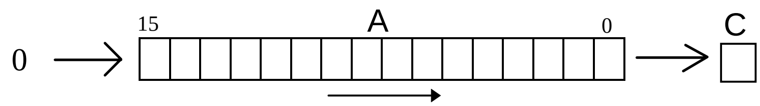
* n=number of bits to shift

*	<byte> Byte Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	-			-
Rn	68　+n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80　+n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SRL　A,width
SRL　A

Word Right Shift (With Carry)

Function



Description

- This instruction shifts the accumulator (A) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to shift with a value 1 to 4.　One instruction can shift a maximum of 4 bits.
- "SRL A" is equivalent to "SRL A,1."
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
1

C : The last value carried out of A_0 will be entered in C.

Codes

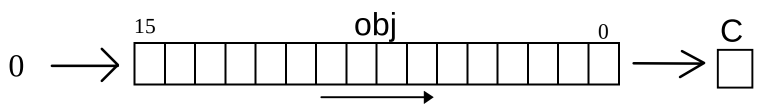
Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SRL	A		9F						2
		width	BC	9C +width					4+n*

* n=number of bits to shift

SRL obj,width
SRL obj

Word Right Shift (With Carry)

Function



Description

- This instruction shifts obj (word length) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SRL obj" is equivalent to "SRL obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

C : The last value carried out of A_0 will be entered in C.

Codes

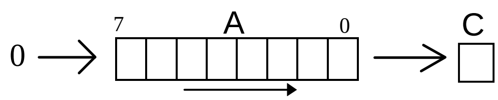
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
SRL	**	width	<word>	9C +width					+2+n*

* n=number of bits to shift

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

SRLB A,width
SRLB A Byte Right Shift (With Carry)

Function



Description

- This instruction shifts the accumulator low byte (AL) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SRLB A" is equivalent to "SRLB A,1."
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD
0

C : The last value carried out of A_0 will be entered in C.

Codes

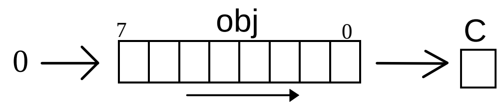
Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
SRLB	A		9F						2
		width	BC	9C +width					4+n *

* n=number of bits to shift

SRLB obj,width
SRLB obj

Byte Right Shift (With Carry)

Function



Description

- This instruction shifts obj (byte length) up to 4 bits to the right, including the carry flag.
- The width specifies the number of bits to shift with a value 1 to 4. One instruction can shift a maximum of 4 bits.
- "SRLB obj" is equivalent to "SRLB obj,1."

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*					

Flags affecting instruction execution

DD

C : The last value carried out of A_0 will be entered in C.

* n=number of bits to shift

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
SRLB	*	width	<byte>	9C+width					+2+n *

* n=number of bits to shift

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

ST A,obj

Word Store

Function

obj ← A

Description

- This instruction stores the contents of the accumulator (A) into obj (word length).
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
ST	A	ERn	38 +n						2
		PRn	3C +n						2
		[X1]	30						4
		[DP]	32						4
		[DP–]	31						5
		[DP+]	33						5
		fix	34	fix8					4
		off	35	off8					4
		sfr	36	sir8					4
		dir	37	dirL	dirH				6
		D16[X1]	C8	D16L	D16H				6
		D16[X2]	BC	99	D16L	D16H			8
		n7[USP]	C9	n7					6
		n7[DP]	C9	80 +n7					6

STB A,obj

Byte Store

Function

obj ← AL

Description

- This instruction stores the contents of the accumulator low byte (AL) into obj (word length).
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
STB	A	Rn	38 +n						2
		[X1]	30						4
		[DP]	32						4
		[DP–]	31						5
		[DP+]	33						5
		fix	34	fix8					4
		off	35	off8					4
		sfr	36	sir8					4
		dir	37	dirL	dirH				6
		D16[X1]	C8	D16L	D16H				6
		D16[X1]	BC	99	D16L	D16H			8
		n7[USP]	C9	n7					6
		n7[DP]	C9	80 +n7					6

SUB A,obj

Word Subtraction

Function

A ← A-obj

Description

- This instruction performs word subtraction, subtracting the contents of obj (word length) from the accumulator (A).
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	
SUB	A	ERn	08+n						3
		PRn	0C+n						3
		#N16	8E	N16L	N16H				6
		fix	8C	fix8					4
		off	8D	off8					4

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
SUB	A	**	<word>	85					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

SUB obj1,obj2

Word Subtraction

Function

obj1 ← obj1+obj2

Description

- This instruction performs word subtraction, subtracting the contents of obj2 (word length) from obj1 (word length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
SUB	**	fix	<word>	80	fix8				+5
		off	<word>	81	off8				+5
		sfr	<word>	82	sfr8				+5
		#N16	<word>	83	N16L	N16H			+6
		A	<word>	84					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

SUBB A,obj

Byte Subtraction

Function

AL ← AL-obj

Description

- This instruction performs byte subtraction, subtracting the contents of obj (byte length) from the accumulator low byte (AL).
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
SUBB	A	Rn	08+n						3
		#N8	8E	N8					4
		fix	8C	fix8					4
		off	8D	off8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
SUBB	A	*	<byte>	85					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SUBB obj1,obj2

Byte Subtraction

Function

obj1 ← obj1–obj2

Description

- This instruction performs byte subtraction, subtracting the contents of obj2 (byte length) from obj1 (byte length).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
*	*	*	*	*	

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
SUBB	*	fix	<byte>	80	fix8				+5
		off	<byte>	81	off8				+5
		sfr	<byte>	82	sfr8				+5
		#N8	<byte>	83	N8				+4
		A	<byte>	84					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

SWAP

High/Low Byte Swap

Function

AH ←→ AL

Description

- This instruction swaps the accumulator's high byte (AH) and low byte (AL).
- Differences with nX-8/100-400:

□ nX-8/500S : DD does not affect instruction execution.

```
LB      A,#12H
SWAP            ; AH←→AL
```

□ nX-8/100-400 : Instruction execution is limited to when DD is 1.

```
LB      A,#12H
SWAP            ; will operate as SWAPB
```

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	(Internal)
SWAP		DF						2

TBR obj

Test Bit (Register Indirect Bit Specification)

Function

if obj.(AL)=0 then Z←1 else Z←0

Description

- This instruction tests the contents of the bit at the specified position within the bit block and sets the zero flag. If the specified bit is 0, then Z will be set to 1; if the bit is 1, then Z will be reset to 0.
- The bit block is the block of 256 bits starting from the address obj. A byte addressing specification is coded in obj.
- The bit position is 0-255, specified by the contents of the accumulator low byte (AL).
- The same instruction coded for nX-8/100-400 has a different function. For nX-8/100-400, only the lower 3 bits of AL are valid for the bit position specification. In this case, only the 8 bits of obj can specified as the target bit.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*				

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles
mnemonic	operand	prefix	+1st	+2nd	+3rd	+4th	+5th	(Internal)
TBR	*	<byte>	CA					+5

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

TJNZ A, radr

Word Test and Jump (Jump If Non-Zero)

Function

if A≠0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of the accumulator (A) are non-zero.
- The jump address is coded in radr. It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction). radr must exist within the current physical segment.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
TJNZ	A	radr	BC	A6	rdiff8				7/11

TJNZ obj, radr

Word Test and Jump (Jump If Non-Zero)

Function

if obj≠0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of obj (word length) are non-zero.
- The jump address is coded in radr. It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction). radr must exist within the current physical segment.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd	+4th	+5th	
TJNZ	**	radr	<word>	A6	rdiff8				+4/8

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

TJNZB A, radr

Byte Test and Jump (Jump If Non-Zero)

Function

if AL≠0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of the accumulator low byte (AL) are non-zero.
- The jump address is coded in radr. It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction). radr must exist within the current physical segment.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
TJNZB	A	radr	BC	A6	rdiff8				7/11

TJNZB　obj, radr　　Byte Test and Jump (Jump If Non-Zero)

Function

if obj≠0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of obj (byte length) are non-zero.
- The jump address is coded in radr.　It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction).　radr must exist within the current physical segment.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
TJNZB	*	radr	<byte>	A6	rdiff8				+4/8

*	<byte> Byte Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	-			-
Rn	68　+n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80　+n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

TJZ　A, radr

Word Test and Jump (Jump If Zero)

Function

if A=0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of the accumulator (A) are zero.
- The jump address is coded in radr.　It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction).　radr must exist within the current physical segment.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
TJZ	A	radr	BC	A7	rdiff8				7/11

TJZ obj, radr

Word Test and Jump (Jump If Zero)

Function

if obj=0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of obj (word length) are zero.
- The jump address is coded in radr. It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction). radr must exist within the current physical segment.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
TJZ	**	radr	<word>	A7	rdiff8				+4/8

**	<word> Word Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

TJZB A, radr

Byte Test and Jump (Jump If Zero)

Function

if AL=0 then PC← radr
However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of the accumulator low byte (AL) are zero.
- The jump address is coded in radr. It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction). radr must exist within the current physical segment.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
TJZB	A	radr	BC	A7	rdiff8				7/11

TJZB obj, radr

Byte Test and Jump (Jump If Zero)

Function

if obj=0 then PC← radr

However, the next instruction's first address-128 ≤ radr ≤ the next instruction's first address+127 and CSR:0000H ≤ radr ≤ CSR:0FFFFH

Description

- This instruction branches to the specified jump address if the contents of obj (byte length) are zero.
- The jump address is coded in radr. It is restricted to the relative jump range defined by a signed 8-bit displacement added to a base (the first address of the next instruction). radr must exist within the current physical segment.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
TJZB	*	radr	<byte>	A7	rdiff8				+4/8

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

VCAL Vadr

Vector Call

Function

1) Small/compact memory models

(SSP)←PC+1 ; move next PC
SSP←SSP-2
PC←(Vadr) ; store Vadr to PC

2) Medium/large memory models

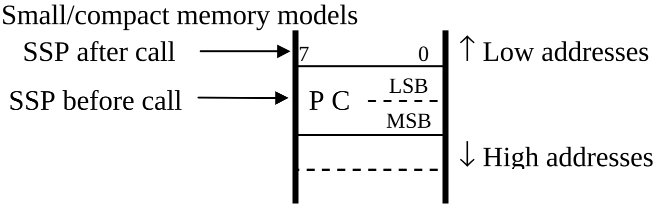
(SSP)← PC+1 ; move next PC
SSP←SSP-2
(SSP)← CSR ; move CSR
SSP←SSP-2
CSR←0 ; VCAL subroutine must be in physical segment 0
PC←(Vadr) ; store Vadr to PC

However, 0:4AH ≤ Cadr ≤ 0:68H, and even address for both 1) and 2).

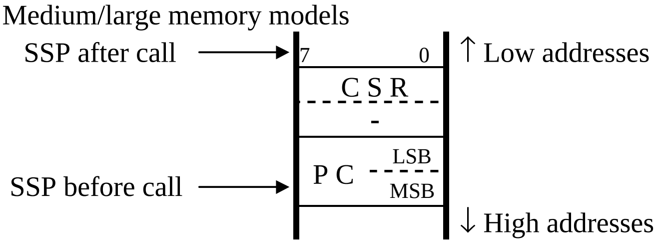
Description

- This instruction calls the subroutine whose jump address is the data word in the VCAL table area specified by Vadr.
- A vector address is coded in Vadr. Any address in the 64K bytes of physical segment 0 can be specified as a vector.
- The called subroutine must exist in physical segment 0.
- The state of the stack after execution of a VCAL instruction is shown below.
- Subroutines called with a VCAL instruction return using an RT instruction in the small/compact memory models, or an FRT instruction in the medium/large memory models.

1) Small/compact memory models



2) Medium/large memory models



Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD

Codes

Instruction Syntax		Instruction Code						Cycles (Internal)
mnemonic	operand	1st	2nd	3rd	4th	5th	6th	
VCAL	Vadr	E0 +Vno						10

XCHG A,obj

Word Exchange

Function

A←→obj

Description

- This instruction exchanges the contents of the accumulator (A) with the contents of obj (word length).
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
XCHG	A	**	<word>	C8					+3

**	<word> Word Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

XCHGB A,obj

Byte Exchange

Function

AL←→obj

Description

- This instruction exchanges the contents of the accumulator low byte (AL) with the contents of obj (byte length).
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
XCHGB	A	*	<byte>	C8					+3

*	<byte> Byte Prefix Instruction Code 1st	2nd	3rd	Cycles (Internal)
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

XOR　A,obj　　Word Logical Exclusive OR

Function

$A \leftarrow A \veebar obj$

Description

- This instruction takes the word logical exclusive OR of the contents of obj (word length) and the accumulator (A), and stores the result in the accumulator.
- Execution of this instruction is limited to when DD is 1 (word).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD
1

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
XOR	A	off	DD	off8					4
		#N16	DE	N16L	N16H				6

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
XOR	A	**	<word>	D5					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64　+n			2
PRn	60　+n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80　+n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

XOR obj1,obj2

Word Logical Exclusive OR

Function

$obj1 \leftarrow obj1 \veebar obj2$

Description

- This instruction takes the word logical exclusive OR of the contents of obj1 (word length) and obj2 (word length), and stores the result in obj1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles (Internal)
mnemonic	operand		prefix	+1st	+2nd	+3rd			
XOR	**	fix	<word>	D0	fix8				+5
		off	<word>	D1	off8				+5
		sfr	<word>	D2	sfr8				+5
		#N16	<word>	D3	N16L	N16H			+6
		A	<word>	D4					+2

**	<word> Word Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
ERn	64 +n			2
PRn	60 +n			2
[X1]	A0			4
[DP]	A2			4
[DP–]	A1			5
[DP+]	A3			5
fix	A4	fix8		4
off	A5	off8		4
sfr	A6	sfr8		4
SSP	A6	00		4
LRB	A6	02		4
dir	A7	dirL	dirH	6
D16[X1]	A8	D16L	D16H	6
D16[X2]	A9	D16L	D16H	6
n7[DP]	8B	n7		6
n7[USP]	8B	80 +n7		6
[X1+A]	AA			6
[X1+R0]	AB			6

XORB A,obj

Byte Logical Exclusive OR

Function

$AL \leftarrow AL \veebar obj$

Description

- This instruction takes the word logical exclusive OR of the contents of obj (byte length) and the accumulator low byte (AL), and stores the result in the accumulator.
- Execution of this instruction is limited to when DD is 0 (byte).

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD
0

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		1st	2nd	3rd	4th	5th	6th	(Internal)
XORB	A	off	DD	off8					4
		#N8	DE	N8					4

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
XORB	A	*	<byte>	D5					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2

XORB　obj1,obj2　　Byte Logical Exclusive OR

Function

obj1 ← obj1 $\veebar$ obj2

Description

- This instruction takes the word logical exclusive OR of the contents of obj1 (byte length) and obj2 (byte length), and stores the result in obj1.

Flags

Flags affected by instruction execution

C	Z	S	OV	HC	DD
	*	*			

Flags affecting instruction execution

DD

Codes

Instruction Syntax			Instruction Code						Cycles
mnemonic	operand		prefix	+1st	+2nd	+3rd			(Internal)
XORB	*	fix	<byte>	D0	fix8				+5
		off	<byte>	D1	off8				+5
		sfr	<byte>	D2	sfr8				+5
		#N8	<byte>	D3	N8				+4
		A	<byte>	D4					+2

*	<byte> Byte Prefix Instruction Code			Cycles (Internal)
	1st	2nd	3rd	
A	-			-
Rn	68 +n			2
[X1]	B0			4
[DP]	B2			4
[DP–]	B1			5
[DP+]	B3			5
fix	B4	fix8		4
off	B5	off8		4
sfr	B6	sfr8		4
dir	B7	dirL	dirH	6
D16[X1]	B8	D16L	D16H	6
D16[X2]	B9	D16L	D16H	6
n7[DP]	9B	n7		6
n7[USP]	9B	80 +n7		6
[X1+A]	BA			6
[X1+R0]	BB			6
PSWL	8A			2
PSWH	9A			2