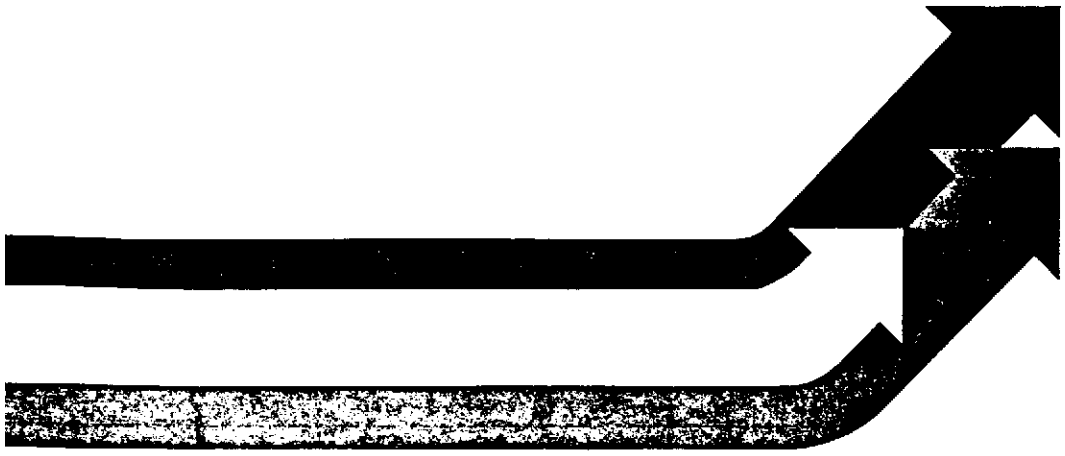
OKI

INSTRUCTION MANUAL

MSM66201

CMOS SINGLE CHIP 8/16 BIT
MICROCONTROLLER



FIRST EDITION
ISSUE DATE : SEP 1991

PREFACE

This manual provides an overview of the MSM66201 and explains its instruction set for programming.

The MSM66201 contains on-chip I/O ports, A/D converters, serial ports, timers, PWM, and other powerful hardware. For information on the functions and control of this hardware, please refer to the *MSM66201 User's Manual*. The explanations presented in this manual assume that the reader understands the contents of the *MSM66201 User's Manual*.

Beyond remembering the hardware configuration of the microcontroller, the first problem encountered by a designer trying to create a program will be how to access the on-chip hardware and memory, and which instructions to use.

The MSM66201 places internal hardware and its control registers in a mapped data memory space. Therefore, control of the MSM66201's on-chip hardware can be considered equivalent to determining how to access its memory space. The fastest way to understand the MSM66201 instruction set is to understand how its memory space is addressed.

Chapter 1, "Overview," explains use and design considerations for the MSM66201's memory configuration and for addressing the hardware mapped in its memory space.

Chapter 2, "Addressing Modes," explains memory space is addressing and its syntax, and provides examples.

Chapter 3, "Details of Instructions," provides a chart of the MSM66201 instructions broken down by function, and it gives detailed explanation of each instruction in alphabetical order. A list of byte counts and cycle counts is also provided.

Chapter 1. Overview

1. Key Features of the MSM66201 Microcontroller 1-1

2. Memory Configuration 1-3
2.1 Program Memory Space 1-3
2.1.1 Structure of Program Memory Space 1-3
(1) Vector Table Space 1-3
(2) VCAL Table Space 1-5
(3) Conventional Coding of Program Space 1-5
2.1.2 Accessing the Program Memory Space 1-7
(1) Access by PC 1-7
(2) Access by Instructions 1-7
2.2 Data Memory Space 1-8
2.2.1 Structure of Data Memory Space 1-8
(1) Concept of Page 1-8
(2) Configuration of Page 0 1-10
(3) Local Registers 1-13
2.2.2 Accessing Data Memory Space 1-15
(1) Accessing the General Data Space 1-15
(2) Accessing the System Stack Space 1-17
2.2.3 Word Boundary 1-17
«Instructions that are Influenced by Word Boundary» 1-20
2.2.4 Data Descriptor (DD) and Stack Flag (SF) 1-21
(1) Data Descriptor (DD) 1-21
2.3 Concept of Segments 1-25
«Description of Absolute Segment» 1-26
«Description of Relocatable Segment» 1-29
«Partial Segment» 1-33

Chapter 2. Addressing Modes

1. RAM Addressing Modes 2-1
1.1 Register Addressing 2-1
(1) Accumulator Addressing 2-1
(2) Pointing Register Addressing 2-1
(3) Control Register Addressing 2-2
(4) Local Register Addressing 2-2
(5) System Stack Pointer Addressing 2-3
1.2 Page Addressing 2-3
(1) Current Page Addressing 2-3
(2) Zero Page Addresssing 2-4
1.3 Pointing Register Indirect Addressing 2-5
(1) Data Pointer Indirect Addressing 2-5
(2) User Stack Pointer Indirect Addressing (with 8-bit Displacement) 2-5
(3) Index Register Indirect Addressing (with 16-bit Displacement) 2-6
1.4 Immediate Addressing 2-7
«Advice About RAM Addressing» 2-7

2. ROM Addressing Modes 2-8
2.1 Direct Addressing 2-8
2.2 Indirect Addressing 2-8
(1) Single Indirect Addressing 2-8
(2) Double Indirect Addressing 2-9

(3) Indirect Addressing with 16-bit base 2-10
3. Bit Addressing Modes **2-12**
4. Logical Bit Address Space **2-13**

Chapter 3. Details of Instructions

1. Classification of Instructions **3-1**
2. Instruction Set **3-5**
3. Summary List of Instructions **3-180**

1. Key Features of the MSM66201 Microcontroller

The MSM66201 is a single chip microcontroller (SCMC) specifically designed for efficient programming and high speed data processing.

SCMCs were originally developed to replace control devices for home equipment, particularly electrical and electronic appliances. The internal memory capacity of early SCMCs ranged from 2 to 4K bytes for ROM and from 128 to 256 bytes for RAM. Internal hardware was limited to ports, timers, counters and LCD drivers. The instruction sets available with these early SCMCs included some novel bit manipulations for control; however, the number of arithmetic instructions and addressing modes as well as the quality of these SCMCs were all inferior to what could be obtained with microprocessors.

With the recent development of DRAM and general memory based on large-scale IC technology, the internal memory capacity of SCMCs has increased tremendously, and the CPU processing speed of SCMCs has jumped from several microseconds to several hundred nanoseconds. Consequently the applications for SCMCs are expanding and gradually invading many areas previously occupied by microprocessors. SCMCs are now used for many devices which require high speed processing of large volume of data such as telephone switchboards, high-performance printers, and automobile engine control units.

Such applications for SCMCs have in turn created new functional requirements beyond the need for increased internal memory capacity. These functional requirements must be satisfied by the architecture of future SCMCs, and they are listed below.

1. Increased parallel processing capability
2. Added internal hardware to perform such functions as A/D and D/A conversions, improved communication control, and various display drivers
3. In addition to the hardware development requirements described in item 2, internal hardware readily applicable to the end products is needed
4. Compatibility with large data memory space
5. High speed program processing/low consumption of electrical power

The MSM66201 satisfies these requirements with the following architecture.

(1) Based on Concepts of Application Specific ICs (ASICs)

Future SCMCs must be ASIC oriented as stated in item 3 above. To meet this requirement, the CPU core of the MSM66201 is designed to be independent of I/O (internal peripheral units). Oki Electric Industry Co. Ltd. intends to develop various I/O units which are readily integrated in the end products with the standard CPUs. The great advantages of developing such a family of I/O units are the flexibility of architecture permitted for the product design, and the reuse of the software developed for the standard CPU in all products. The MSM66201 is the vanguard of SCMCs which are developed based on such a concept.

(2) 64K Byte Program Memory Space

The MSM66201 provides a program memory space of 64K bytes. Any address within the memory space is accessible directly or indirectly by CALL or JUMP instructions. The memory space can also store the program constants.

(3) Hierarchy of 64M Byte Data Memory Space

The MSM66201 provides a data memory space of 64M bytes . The memory space consists of three levels, namely banks, pages, and local registers.

The memory space is divided into 256 pages, each holding 256 bytes. The MSM66201 addressing system is designed to place all of the data required for a single processing action on the same page to achieve quick and efficient addressing. To make the data processing within a page effective, each page is divided in 32 areas of 8 bytes, and local registers may be placed in any area.

(4) Complete Bit Manipulation Instruction

The MSM66201 permits individual manipulation of any bit in the data memory space. Also, it is possible to manipulate directly and indirectly any bit in a byte. Transfer between the carry flag(C) and a bit is also possible.

(5) Ease of Array and Pointer Manipulation

The MSM66201 provides 2 index registers of 16 bits and 8 sets of pointing register of 16 bits. These pointers make it possible to access any address of the data memory space. The contents of the registers are automatically saved when a program interrupt occurs and the context switch-over takes effect. Creating data arrays and manipulating pointers is easy on the MSM66201.

(6) Organized Instructions

In the MSM66201 design, the internal hardware and registers are allocated to the memory space, and numerous addressing modes to access data are available; therefore, it is possible to manage and control all resources of the MSM66201 with a few instructions. The transfer and arithmetic instructions are plain and clear, consequently program coding is easy, and programs produced are easy to understand.

(7) Extremely Quick Switching of Program Context

The MSM66201 design permits extremely quick switching of the program context when a program interrupt handler functions or a task switching is executed in a multitasking program.

(8) Memory Mapped I/O

In the MSM66201, the internal I/O units such as I/O ports and timers are allocated to the data memory space (memory mapped I/O). Consequently no specific input and output instructions are required; these instructions are replaced by the data addressing instructions. This scheme produces not only a simple instruction system but also a flexible design that makes adding and modifying of internal I/O hardware easy. The MSM66201 is really developed to meet the ASIC requirements of the future, and the memory mapped I/O design enhances the ASIC features of the MSM66201.

2. Memory Configuration

The memory space of the MSM66201 is divided into the program and data memory spaces which are described in subsections 2.1 and 2.2 respectively. In subsection 2.3, the concept of data segments that are allocated to the memory space for assembler operation is described.

[NOTE] The program memory space (or program space) is sometimes called ROM (Read Only Memory) space because ROM is usually its main constituent element. Likewise the data memory space (or data space) is sometimes called RAM (Random Access Memory) space because RAM is usually its main constituent element.

2.1 Program Memory Space

2.1.1 Structure of the Program Memory Space

The program space of the MSM66201 accommodates 64K bytes (0000_H to $FFFF_H$) maximum; 40 bytes (0000_H to 0027_H) of which forms the vector table space composed of 20x2-byte segments. Another 16 bytes (0028_H to 0037_H) form the VCAL table space of 8x2-byte segments as shown in Figure 1-1.

(1) Vector Table Space

The vector table space saves the vector addresses when the reset instructions (reset input, BRK and WDT) are executed or interrupts are caused by the peripheral hardware. Each vector address is saved in 2 bytes (16 bits); the lower and upper bytes of the vector address are stored respectively in the lower and upper addresses of the RAM. The vector table is described by the assembler with pseudo instructions. (See the following examples)

Examples of Assembler Coding for Vector Table

```
        CSEG  AT  0000H
;
        DW    START          ----------  0000H
        DW    BRK_RESET      ----------  0002H
        DW    WDT_RESET      ----------  0004H
        DW    NMI_ENTRY      ----------  0006H
        DW    INT0_ENTRY     ----------  0008H
                :
        DW    INT1_ENTRY     ----------  0026H
```

Please note that at a reset or interrupt, the contents of the concerned vector table (vector addresses) are set in the PC; no jump instruction into the vector table space is specified. Also, it is mandatory to allocate addresses 0 through 1 to the vector addresses for reset input, but the remaining addresses (2 through 27_H) need not be specified when no corresponding reset and interrupt instructions are executed, and the unused area may be used to enter programs.

The allocation of the vector table is shown in Table 1-1.

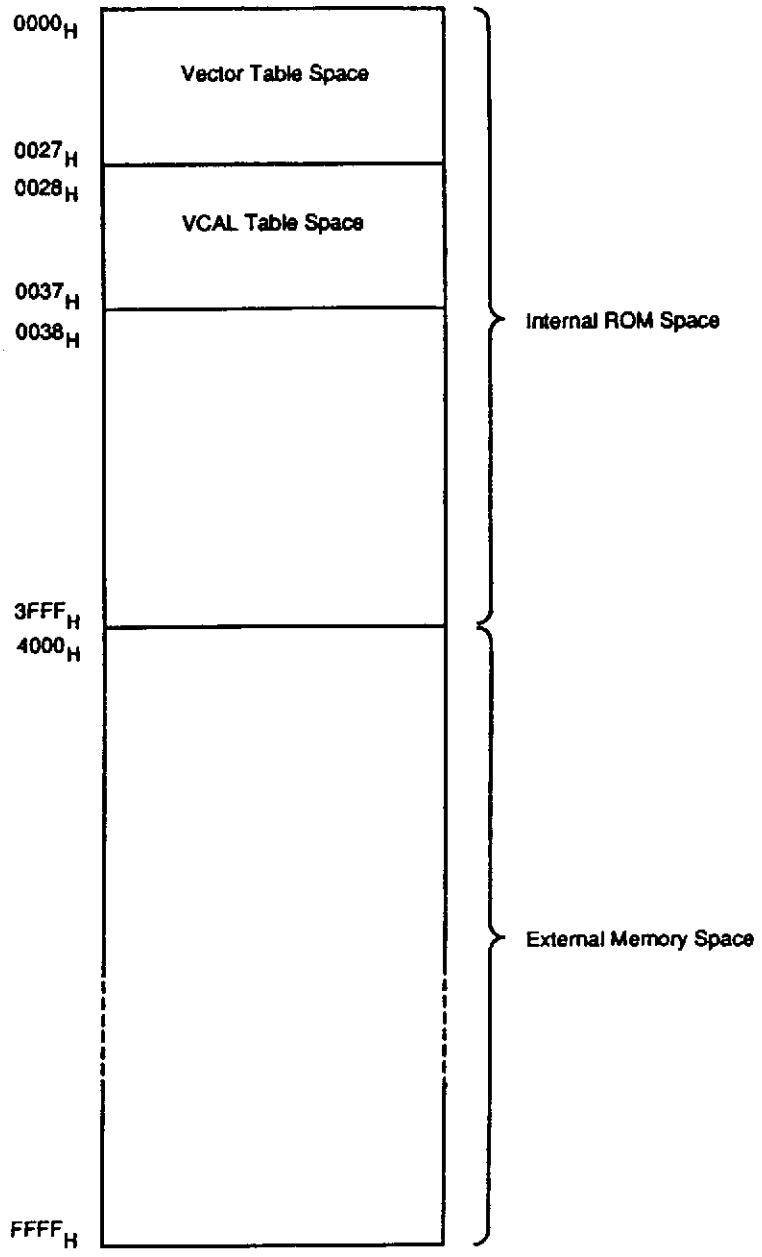


Figure 1-1. Structure of Program Memory Space

Table 1-1. Allocation of Vector Table

Address	Corresponding reset process or interrupt
0000_H	Reset by RESET input
0002_H	Reset by BRK instruction
0004_H	Reset by WDT / Reset by operation code trap function
0006_H	Interrupt by NMI pin input
0008_H	Interrupt by INT0 pin input
$000A_H$	Interrupt by serial port readying for receiving
$000C_H$	Interrupt by serial port readying for transmit
$000E_H$	Interrupt by BRG for serial port receive
0010_H	Interrupt by timer 0 overflow
0012_H	Interrupt by event occurrence of timer 0
0014_H	Interrupt by timer 1 overflow
0016_H	Interrupt by event occurrence of timer 1
0018_H	Interrupt by timer 2 overflow
$001A_H$	Interrupt by event occurrence of timer 2
$001C_H$	Interrupt by timer 3 overflow
$001E_H$	Interrupt by event occurrence of timer 3
0020_H	Interrupt by end of A/D conversion
0022_H	Interrupt by PWM timer overflow
0024_H	Interrupt by BRG for serial port transmit
0026_H	Interrupt by INT1 pin input

[NOTES] 1. The preceding allocation is applicable only for the MSM66201. The MSM66201 is designed as the basic unit of a family of microcontrollers (nX8/300 series) that is based on ASIC concepts, and the MSM66201 is marketed as a standard product. Therefore it is possible to install customer specific internal I/O units which are different from the standard MSM66201 as long as the CPU remains the same. For such cases, the allocation of the vector table may change. Also the number of interrupts may increase beyond the standard memory space (0000_H to 0027_H) capacity; the excess interrupt vector addresses must be allocated behind the VCAL table space(following 0038_H).

2. When an address of 16-bit length is stored in data memory, in a register, or is transferred, the lower byte of the data is stored in the lower address of the memory or register and the upper byte is stored in the upper address. This rule is implicitly applicable in the MSM66201 device unless it is stated otherwise.

(2) VCAL Table Space

The MSM66201 uses a vector call instruction (VCAL) which is one byte long. The addresses of the VCAL table (0028_H through 0037_H) are entered in the operand of the VCAL instruction. When the VCAL instruction is executed, the vector address of the VCAL table specified by the operand is set in the PC after the return address is saved in the system stack.

The VCAL table is entered by a DW pseudo instruction like the vector table. When VCAL instructions are not used, the VCAL table area can be used for entering programs.

(3) Conventional Coding of Program Space

As stated in the articles (1) and (2), the address area from 0000_H to 0037_H is allocated to the tables. Also, as stated in (1), the area excluding from 0000_H to 0001_H may be used for entering

programs, however it is recommended that this be avoided. Enter programs in the addresses higher than 0038_H, because later program changes may require the addition of instructions for interrupts and VCAL.

The following three examples are produced using these rules.

Conventional Coding of Program Space

```
MAIN    SEGMENT  CODE
CSEG    AT   0000H
;
;       ******************
;       *  Vector Table  *
;       ******************

DW      START           ; Reset
DW      BRK_RESET       ; Break instruction

        :

DW      INT1_ENTRY      ; INT1 Interrupt
```

Coding Vector Table

```
;       ******************
;       *  VCAL Table    *
;       ******************

VCAL0:  DW          ERROR_PROCESS
VCAL1:  DW          PORT_INIT_PROCESS
VCAL2:  DW          RAM_INIT_PROCESS

DS      10              ; Reserved for future
```

Coding VCAL Table

```
;       ******************
;       *  Main Routine  *
;       ******************

RSEG    MAIN
;
START:
        :
        :
        :
        :
END
```

Program Body

2.1.2 Accessing Program Memory Space

Accessing program memory space is performed in two ways: by PC, or by instruction.

(1) Access by PC

Accessing the program memory by using the PC is usually performed when a program is being executed. The PC always contains the address of the memory where the next instruction to be executed is stored. The contents of the address indicated by the PC is the instruction. Following the execution of an instruction, the contents of PC is automatically increased an amount equal to the number of bytes of the executed instruction by hardware. When a reset input is given from an external pin or an interrupt occurs, the PC contents is replaced by the value stored in the vector table. These operations are done by the hardware; the programmer need not be concerned with them. However, the PC count can be changed by the programmer if necessary. For instance, it is possible to replace the PC contents with the value stored in the VCAL table by executing a VCAL instruction. Also, the PC contents can be replaced by the addresses specified by the operands of the following instructions upon their executions.

Instructions that Can Replace PC Contents

SJ, J, JC, JBS, JBR, JRNZ, SCAL, CAL

Refer to Chapter 3 for details of these instructions.

(2) Access by Instruction

The MSM66201 provides instructions that allow accessing the program memory space.

Instructions to Access Program Space

LS, LCB, CMPC, CMPCB

With the use of these instructions, it is possible to transfer the contents of the program memory to the accumulator or compare the program memory to the accumulator.

The following addressing modes are available for specifying the locations of memory space when these instructions are used.

- Direct Addressing: specify the address in the space of 64K bytes directly using immediate addressing.
- Simple Indirect Addressing: specify the address indirectly according to the contents of data memory such as a local register (er0 to er3), pointing register (DP, X1, X2, USP), or system stack pointer (SSP).
- Double Indirect Addressing: specify the address indirectly according to the content of the data memory which is addressed indirectly by the contents of a pointing register (DP, X1, X2, USP).
- Indirect addressing with a 16-bit base: specify the address by offsetting from the base address in the 64K byte memory space; the amount of offset is specified by the contents of a pointing register (DP, X1, X2, USP) or the contents of another data memory.

For the details of the instructions and addressing modes, refer to Chapter 3 and Section 2 of Chapter 2 respectively.

2.2 Data Memory Space

2.2.1 Structure of Data Memory

(1) Concept of Page

This article explains the concept of *pages*.

Generally, the data memory space used for a program module or unit process is not so large. In other words, the upper bits of the addresses of the data in the space which are addressed during a unit process hardly change. So, if the data memory space is divided into blocks of a appropriate size for a unit process, and all of the data involved in a program is stored in one block, addressing the data for the process may be achieved by simply offsetting each data from the base address of the block. Such a configuration of data memory space will shorten the access time as well as the machine codes.

Using such concepts, the architecture of the MSM66201 divides each data memory bank into blocks of 256 bytes. This block is called a *page*. Thus, a bank of 64K bytes consists of 256 pages of 256 bytes.

In Figure 1-2, the configuration of the data memory is shown.

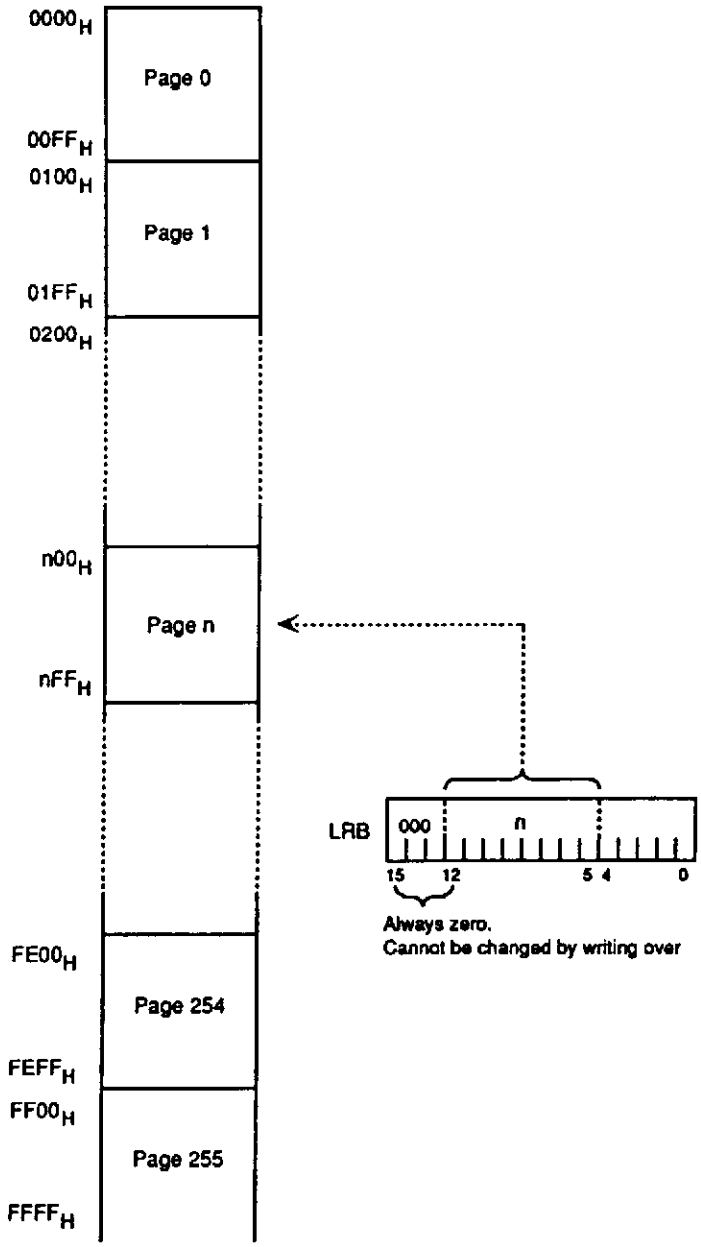


Figure 1-2. Page Configuration of Data Memory

As shown in Figure 1-2, the pages are numbered progressively, starting at the lowest address, as Page 0, Page 1, Page 2, up to Page 255. The page numbers used are specified by bits 12 through 5 of LRB, and the page thus specified is called the current page.

When only the on-chip data space of the MSM66201 is used, the corresponding addresses used are 0000_H through $027F_H$, and the only corresponding pages used are Pages 0, 1, and 2.

Furthermore, only 128 bytes are useable in Page 2. The configuration of the data memory space used for on-chip space only is shown in Figure 1-3.

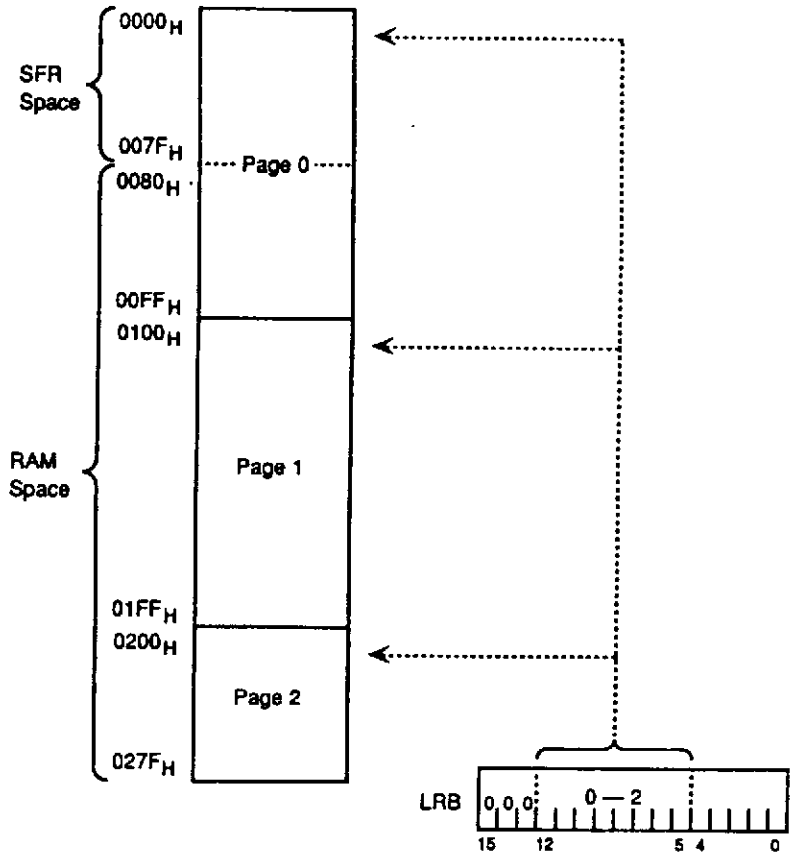


Figure 1-3. Page Configuration of Data Memory (On-Chip Space)

Addressing by offsetting within the current page is called current page addressing; its details are given in Chapter 2. Current page addressing is allowed with most instructions in the MSM66201 instruction set, and our chip designers made their best effort to realize very fast memory access with current page addressing. This fits the concept that the data space for each module should be located within one page, and in that situation current page addressing would be used most frequently.

(2) Configuration of Page 0

All of the pages (256 total) thus defined are basically similar except for Page 0. Initially, the special function register (SFR) and the pointing register (PR) are allocated to Page 0. (See Figure 1-4)

Secondly, to access the contents of Page 0, zero page addressing, which does not depend on LRB, is available. This is in addition to the familiar current page addressing, which is LRB dependent.

These items are further explained in the following paragraphs.

SFR

The internal hardware of the MSM66201 will be accessed at every stage of a program, therefore it is desirable to allocate the hardware to page 0 in a memory-mapped I/O configuration to increase the program efficiency. The configuration of memory-mapped I/O requires fewer instructions to control the hardware units compared to the I/O-mapped I/O configuration, and it also provides the flexibility to modify the internal hardware while the CPU stays unaltered (one of the beneficial features of ASICs).

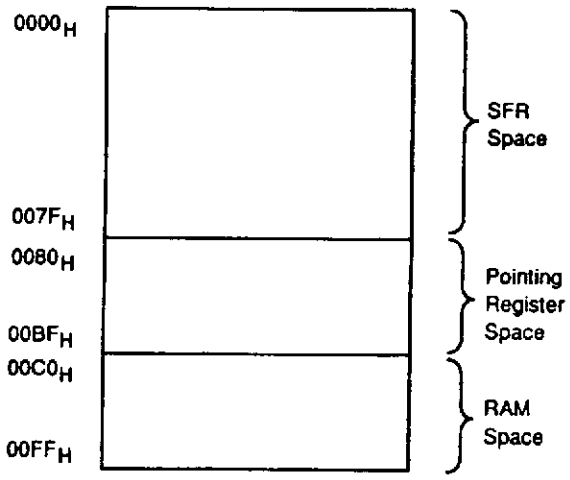


Figure 1-4. Configuration of Page 0

Considering the foregoing factors, the internal hardware of the MSM66201 such as timers, counters, A/D converters, and so on, and the registers which control the memory-mapped I/O hardware, are allocated to the addresses from 0000_H to $007F_H$ of Page 0. The general term for these hardware units is "Special Function Register" or SFR for short; the latter abbreviation is used in this book.

SFR includes, in addition to timers, A/D converters and so on, the general purpose registers which are listed in the following table; these are used for arithmetic and memory management.

General Purpose Registers Included in SFR

SSP	System Stack Pointer
LRB	Local Register Base
PSW	Program Status Word
ACC	Accumulator

Pointing Register (PR)

The pointing registers are allocated to addresses 0080_H through $00BF_H$ of Page 0. They are 16-bit registers and function as pointers when addressing the program or data memory.

The pointing register is composed of the following four registers which form a set.

X1 ..... Index Register 1
X2 ..... Index Register 2
DP ..... Data Pointer
USP ..... User Stack Pointer

Eight pointing registers or eight sets of the foregoing units are allocated to Page 0 and they are named PR0, PR1, PR2 ... PR7. (See Figure 1-5).

For the details of pointing register's function, refer to Section 3 of this chapter and the description of addressing modes in Chapter 3.

As shown in Figure 1-5, selecting the register out of eight units is performed by the lower 3 bits of PSW whose group name is System Control Base (SCB).

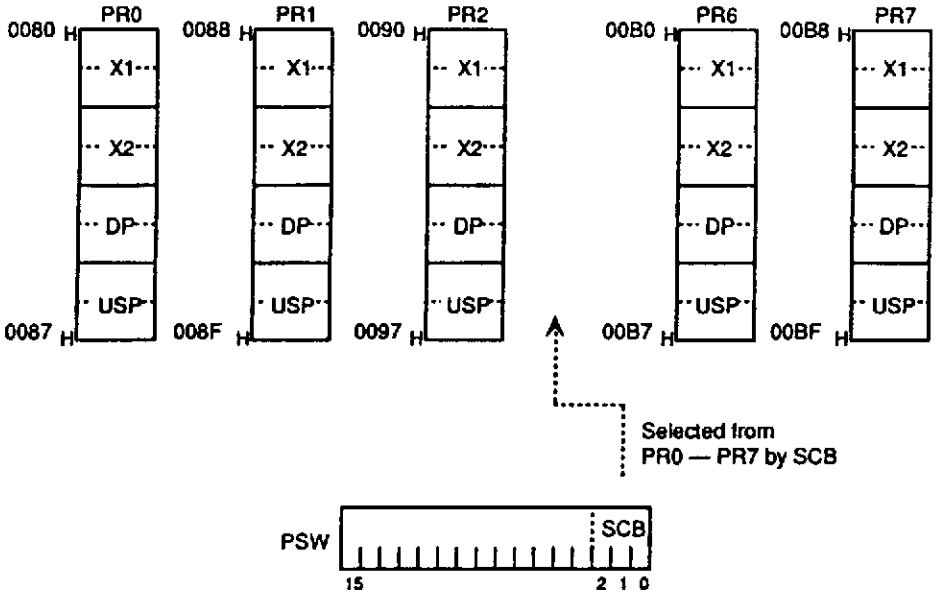


Figure 1-5. Pointing Registers

Zero Page Addressing

As described in (1), the data memory space of the MSM66201 is divided into pages, each of which contains 256 bytes, and the work area of a program module is basically contained within a page. SFR and the pointing registers are contained within Page 0; however, these units are expected to be referred to throughout various stages of program execution, resulting in a changing value of LRB. Such a situation degrades program efficiency. To avoid this, zero page addressing which does not depend on LRB is provided for the data within Page 0. This addressing mode is in addition to universal current page

addressing. The practical application of zero page addressing is described in the *Addressing Modes* in Chapter 2.

As shown in Figure 1-6, addresses 0000_H through $00FF_H$ of Page 0 are RAM. When these addresses are used for space to store global variables like the external variables of a C compiler, zero page addressing can be employed to enhance program efficiency.

(3) Local Registers

As described in the articles (1) through (2), the data memory space is stratified in 256 pages. The upper eleven bits, from 12 to 5, of LRB are used to identify the banks and pages, and the remaining lower five bits of the 13-bit register are used to set the base for local registers.

As described in (1), the data memory space is divided into 256-byte pages. Each page provides register space in 8-byte units. The architecture of the MSM66201 permits the allocation of registers to any convenient addresses within the current page rather than specific fixed addresses. Such registers are called *local registers* and their base addresses are specified by the lowest five bits of LRB. (See Figure 1-6)

The local registers are used to store the data when transfer or arithmetic instructions are executed. The unit of the register can be either byte or word. For byte unit operations, each register forms 8 bits x 8 register sets, and the data is entered starting at the lowest, as r0, r1, r2 . . . r7. The corresponding configuration for word operations is 16 bits x 4 register sets with data entry starting at the lowest, as er0, er1, er2, er3.

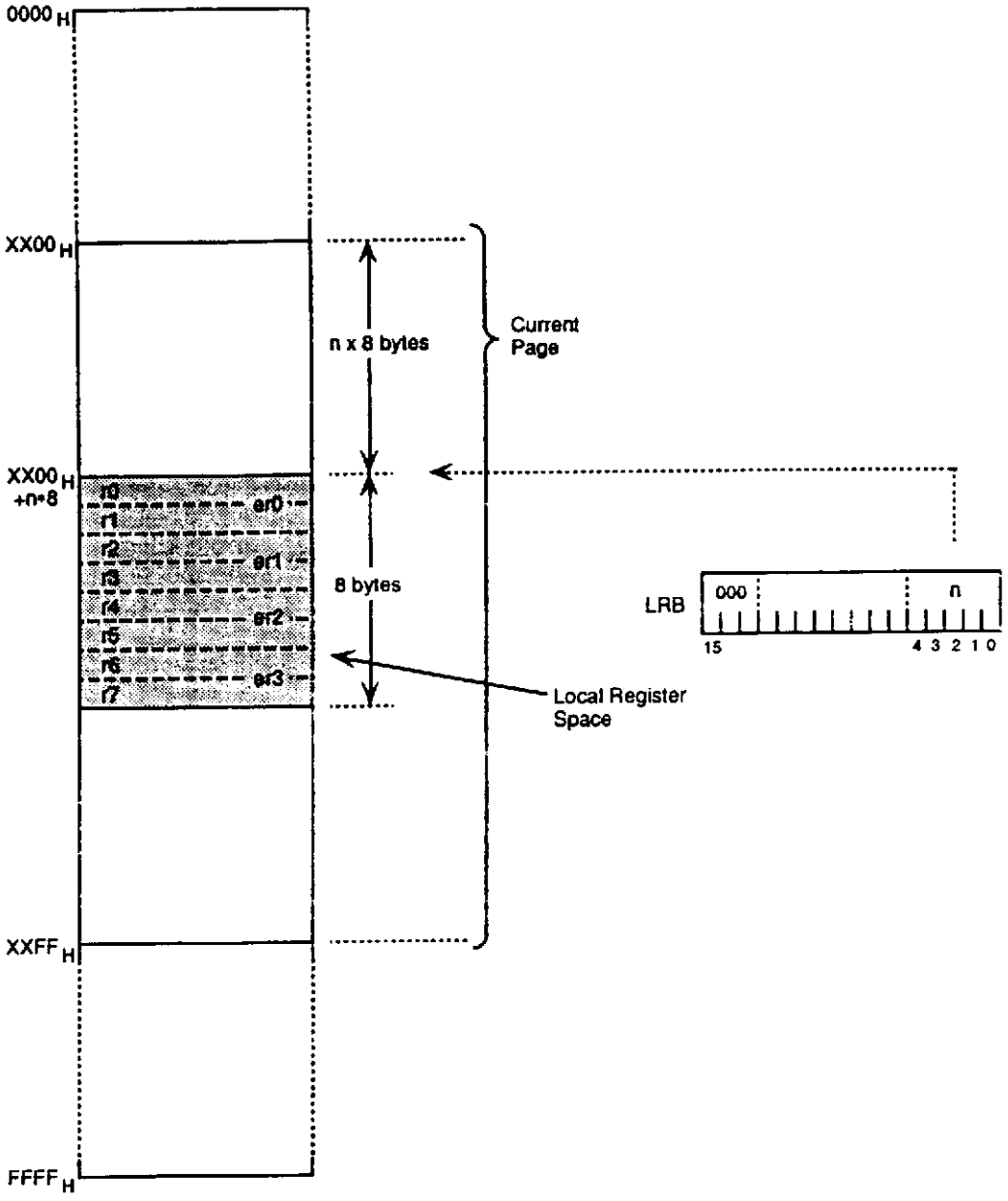


Figure 1-6. Local Register Space

2.2.2 Accessing Data Memory Space

There are three kinds of data memory accessing:

(1) Addressing the general data space

(2) Addressing the system stack space

Detailed descriptions follow.

(1) Accessing the General Data Space

The general data space is all data space other than the spaces which are allocated to the system and user stacks. (Note that the general space includes SFRs, which include ACC, PSW, LRB, etc.)

All addressing of the general data space is performed by the instructions. More than half of the instructions used by the MSM66201 are for addressing the general data space. They are listed below. The details of these instructions are described in Chapter 3, *Details of Instructions*.

Instructions for Addressing the General Data Space

• Data transfer instructions	(Load)	L, LB	(Store)	ST, STB
	(Move)	MOV, MOVB	(Clear)	CLR, CLRB
	(Exchange)	XCHG, XCHGB, XNBL		
• Rotate/Shift instructions	(Rotate)	ROL, ROLB, ROR, RORB		
	(Shift)	SLL, SLLB, SRL, SRLB, SRA, SRAB		
• Increment/Decrement instructions		INC, INCB, DEC, DECB		
• Arithmetic operation instructions	(Multiply)	MUL, MULB	(Divide)	DIV, DIVB
	(Add)	ADD, ADDB, ADC, ADCB	(Subtract)	SUB, SUBB, SBC, SBCB
• Logic operation instructions	(Logical AND)	AND, ANDB	(Logical OR)	OR, ORB
	(Exclusive OR)	XOR, XORB		
• Compare instructions	CMP, CMPB			
• Bit manipulation instructions	(Set bit)	SB, SBR	(Reset bit)	RB, RBR
	(Test bit)	TBR	(Move bit)	MBR

Almost all of the addressing instructions may operate in word length (16-bit length) or byte length (8-bit length) modes. Each instruction accommodates these operations with two distinctive codes like MOV and MOVB, or ADD and ADDB. In Chapter 3, the details of the instructions are given.

In some instructions primarily for the accumulator, the same machine codes, depending on the flag called *data descriptor* (DD), perform both byte and word operations. This flag is allocated in PSW, and constitute the most notable features of the MSM66201 control system with their ability to provide multiple instructions. The functions of these two flags are detailed in 2.2.4.

This section provides additional information about addressing modes for the general data memory. Although the previously described instructions perform the addressing, the programmer must enter the mnemonic codes (MOV, ADD, CMP, etc.) in the assembler source program. These mnemonic codes define the operations, and the objects of the operations are the addresses of the data space specified by the operands following the mnemonic codes.

For example, transfer instruction MOV is entered as

```
MOV operand_1, operand_2
```

This instruction orders the transfer of the contents of operand_2 into operand_1 using word units. Operand_1 and operand_2 specify the addresses that are the objects of the operation. There are various ways to specify or note the addresses of the objects. Addressing notation defines how to access the data space, or in other words, defines the addressing mode.

Two of the addressing modes for the general data space have been described in 2.2.1. They are listed below, and further detailed in Chapter 2.

1. Current page addressing
2. Zero page addressing

In zero page addressing, the address or symbol which is the object of an operation is entered as it is. In current page addressing, a descriptive word *off* for addressing is placed preceding the address or symbol which is the object. For example, to transfer in word length units the data space which is labelled WORK_1 in the current page to the address $A0_H$ of Page 0, the following code is used.

```
MOV 0A0H, off WORK_1
```

Current page and zero page addressing modes are the basic modes; however, the following additional addressing modes are available to enhance program efficiency as listed below.

3. Register addressing
4. Pointing register indirect addressing

The register addressing is specifically used to access the general purpose registers within Page 0. The registers include SSP, LRB,PSW, ACC, X1, X2, DP and USP and the local registers in the current page (r0, r1, r2, r3, r4, r5, r6, r7 or er0, er1, er2, er3). The name of the specific register is entered in the statement except for ACC which changes to A. For example, the following entry will transfer er0 to ACC.

```
MOV A, er0
```

Pointing register indirect addressing is used for programs that require accessing arrays allocated in the data space with the use of pointers, or for accessing data that can be stored indirectly with pointers to improve overall program efficiency. There are three variations to this addressing mode.

The first variation is indirect addressing with a base using X1 and X2. Array-type data are conveniently handled with this access scheme in which the amount of offset from the base is processed by X1 and X2. For example, the following entry will transfer the 10th array element, SEG_DATA[10], in the array SEG_DATA to the accumulator after X1 is set to 10.

```
MOV A, SEG_DATA[X1]
```

The second variation is indirect addressing by the data pointer (DP) that is used as a pointer for general indirect addressing. For example, the following entry will move the data of a byte length pointed to by DP to local register r0.

```
MOVB r0, [DP]
```

The third variation is indirect addressing by USP. The effective address is generated by USP and the signed (±) displacement.

```
MOVB A, +2[USP]
```

The access modes for the general data memory space have been outlined in the preceding sections; however, the functional details of the instructions and the thorough descriptions of the addressing modes are not given. Refer to Chapter III for functional details of each instruction. Refer to Chapter II for details on addressing.

There is one more addressing mode for the general data space which is termed *bit addressing*. Bit addressing, as well as the bit segment, will be described in 2.3, "Concept of Segments".

(2) Accessing the System Stack Space

The system stack is the stack space that is indirectly accessed by system stack pointer (SSP). Accessing the system stack space is performed in the following cases.

1. SCAL, CAL and VCAL instructions are executed
2. Interrupts occur
3. PUSHS instruction is executed
4. RT and RTI instructions are executed
5. POPS instruction is executed

The system stack space is generally called *stack space*, and it is used to save the return address when call instructions are executed (case 1). It is also used to save the return address and the contents of ACC, LRB and PSW when interrupts occur (case 2). The system stack space can be used to stack PR, ACC and LRB directly using the PUSHS instruction (case 3). The RT instruction returns the return address to PC which is saved in the system stack space by CALL instructions (case 4). RTI instruction is used to return from interrupts and to return PSW, LRB, ACC and the return address from the system stack. The POPS instruction is used to pop the data in the system stack which has been put in by PUSHS instruction (case 5).

2.2.3 Word Boundary

In 2.2.2, three categories of accessing the data space are established:

1. Accessing the general data space
2. Accessing the system stack

In category 1, it has been said that accessing the general data space can be entered in word unit or in byte unit. When the word unit is used, the existence of the word boundary must be observed.

The word boundary is the borderline between word units, and it follows behind every odd-numbered address. As shown in Figure 1-7, there is no word boundary between addresses 0 and 1, hence a word that consists of addresses 0 and 1 is permissible. On the other hand, there is a word boundary behind address 3 because it is an odd-numbered address. Therefore, a word that consists of addresses 3 and 4 crosses the word boundary, and such a word is not usable for the accessing operation. In short, each word must start at an even-numbered address.

Using the transfer instruction, MOV as an example, the operation of the CPU core is explained. When the contents of addresses in the data memory space are transferred to the accumulator by zero page addressing, the following entry will transfer the contents of the addresses $C0_H$ and $C1_H$ to the accumulator.

```
MOV A, 0C0H
```

The corresponding machine code is given as:

B5 C0 99

The C0 in the second byte corresponds to the address of the second operand. When these codes are executed by CPU, the contents of the two bytes which starts at address $C0_H$ will be transferred.

If the $C0_H$ of the example address is replaced with $C1_H$, the entry is changed as shown.

```
MOV A, 0C1H
```

As mentioned previously, the words used in the access operation must start at an even-numbered address and be 2 bytes wide. However the $0C1_H$ of the last entry is an odd-numbered address which creates a logical inconsistency, and consequently the assembler issues a warning. The assembler, however proceeds to change the address code as shown.

B5 C1 99

When the CPU executes these codes, the process is identical to the original one where the two bytes of data in addresses $0C0_H$ and $0C1_H$ are transferred to the accumulator. Thus the combination of $0C1_H$ and $0C2_H$ is ineffective, because the CPU forces the LSB (lowest bit) of the effective address to be 0 in the accessing process.

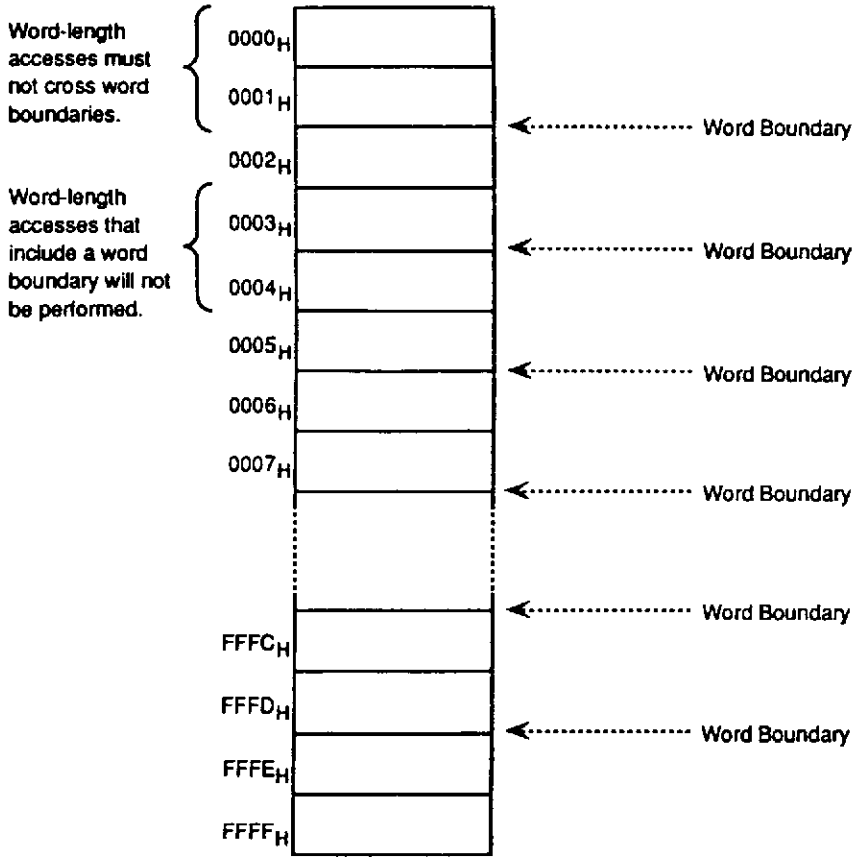


Figure 1-7. Word Boundary

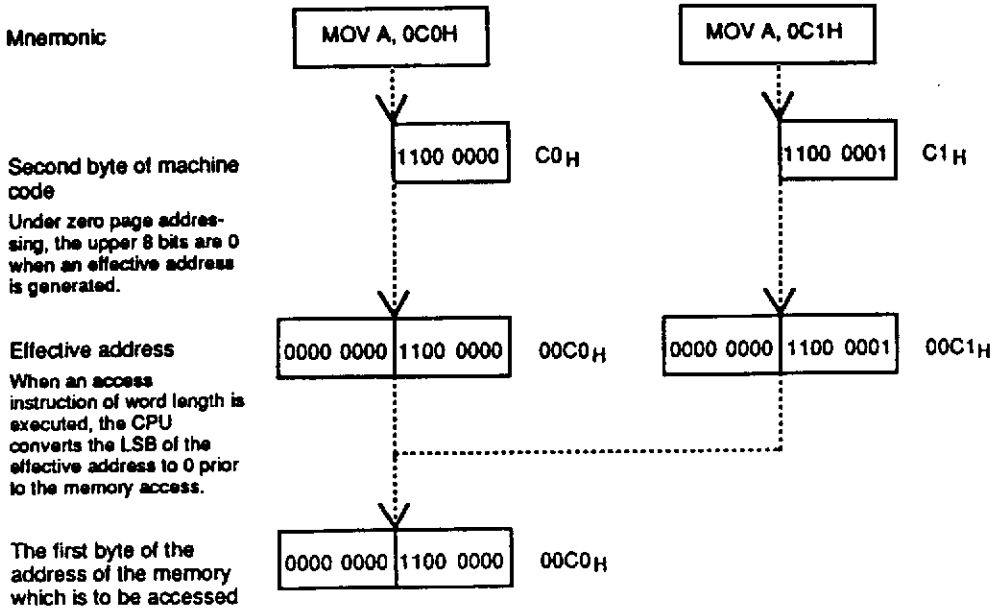


Figure 1-8. Address Generation Process in Accessing Word Long Data Space

Instructions that are Influenced by Word Boundary

The instructions that are influenced by word boundaries are specified here. Word boundaries exists for almost all the instructions that access the general data space in word length units, and for all instructions used to access the system stack space. These instructions are listed in Table 1-3-1 and Table 1-3-2.

Table 1-3-1. Instructions Influenced by Word Boundary

• Data transfer instructions	L, ST, MOV, CLR, XCHG
• Rotate/Shift instructions	ROL, ROR, SLL, SRL, SRA
• Increment/Decrement instructions	INC, DEC
• Arithmetic operation instructions	ADD, ADC, SUB, SBC
• Logic operation instructions	AND, OR, XOR
• Compare instructions	CMP

Table 1-3-2. Instructions Influenced by Word Boundary

• Call instructions	SCAL, CAL, VCAL
• Return instructions	RT, RTI
• Push/Pop for system stack	PUSHS, POPS
• Save into system stack at interrupt	

The instructions unaffected by the word boundary are the instructions for accessing the user stack PUSHU, POPU), and ROM addressing.

Thus, it is possible to push the data in the user stack without regard to their order; the data may be in byte or word length. Also, it is possible to refer to the ROM table starting at an odd-numbered address.

When a programmer codes a source program, the addresses referred to are seldom entered in numbers such as shown below:

```
MOV A, 0C0H
```

The addresses displayed in the operands are usually symbols which are defined in the data segment as labels or defined by DATA pseudo instructions. The data segment is the terminology used for assembler, and its concept is described in detail in 2.3. For the time being, it may be considered to be synonymous to the data memory space.

When the data space of a word unit is entered in a program, the data segments become meaningful. The data segments that contain the work space accessed by word-unit instructions must meet the requirement that the starting bytes (addresses) be even-numbered. In other words, labels for data segments used for the operand of an access instruction that uses word units must be assigned to even-numbered addresses.

When an odd-numbered address is entered for the operand of an access instruction that operates with words in the general data space, the assembler issues an warning message for errors only when the operand is evaluated as an absolute value.

When the operand is evaluated as a relocatable value, the symbol for the operand has the attribute of a relocatable data segment, and the real value of the relocatable value or symbol can not be known until the memory allocation is completed by the linker. This means that the assembler itself is unable to check the value. The currently available linker (RL66K) lacks the capability to perform such a checking function. (Linker improvement is is planned.)

The starting byte of the data segment that is accessed by the instructions which operate on words must be able to instruct the assembler to allocate an even-numbered address. Therefore, the relocatable assembler (RAS66K) interprets the pseudo instruction segment to define segments with the added requirements for word attributes (no crossing of word boundaries). The segment with word attributes is assigned so that the starting address is always an even number.

This checking of word boundaries is performed only for cases in which the addressing entry in the operand indicates directly the accessing object as is done in page addressing. Addressing the general data space can also be done by other methods such as indirect addressing by pointing register, where the operand is entered with an indirect address. For such a case, the assembler is unable to check the word boundary because the assembler needs to know the contents of the register used for indirect addressing prior to checking the word boundary. For the assembler to know the contents of a register it is necessary to check the entire program flow, and this is far beyond the assembler's function. Therefore, managing the word boundary in indirect addressing is entirely the responsibility of the programmer.

2.2.4 Data Descriptor (DD)

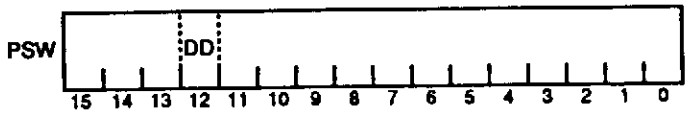
MSM66201 provides a special flag called data descriptor (DD)

DD is associated with manipulation and transfer instructions for the accumulator of the MSM66201. DD affects the length of data accessed.

DD flag is described in the following articles.

(i) DD allocation

DD is allocated to bit 12 of the program status word (PSW).



(ii) Function of DD

Prior to the description of the function of DD, the mnemonic used for the MSM66201 will be explained. The instructions for the MSM66201 are generally classified in two groups: the instructions that operate on almost all of the resources in word length units, and the remaining instructions that operate in byte units. The mnemonic expressions for the latter group of instructions are distinguished from the former with suffix B as shown in the following examples.

Word long MOVE instruction : MOV
Byte long MOVE instruction : MOVB

Word long ADD instruction : ADD
Byte long ADD instruction : ADDB

Word long logical AND : AND
Byte long logical AND : ANDB

The corresponding machine codes for the word based and byte based instructions are generally different. The exception to this is the instructions which operate on the accumulator where the same machine code is used for both instructions of word and byte unit in conjunction with the use of DD. With DD=1, the instructions operate in word length, and with DD=0, they operate in byte length. Such instructions influenced by DD are listed in (iii).

(iii) Instructions influenced by DD

ST / STB	A , obj	store in accumulator instruction
SWAP / SWAPB		swap instruction for accumulator
ROL / ROLB	A	left rotation instruction for accumulator
ROR / RORB	A	right rotation instruction for accumulator
SRA / SRAB	A	right arithmetic shift instruction for accumulator
SLL / SLLB	A	left logical shift instruction for accumulator
SRL / SRLB	A	right logical shift instruction for accumulator

The term *obj* above represents the addressing entry (like r0, er0, [DP], 100H and off 0C0H) which is legally used as the operand of the instruction. Note that all instructions are for operations that use the accumulator.

The machine code for each pair of the instructions listed above is the same. Basically, for an instruction, the mnemonic and the machine code form a one to one correspondence. The assembler of the MSM66201, however, allows two mnemonic terms for a machine code.

For example, the machine code 88 causes the contents of the accumulator to be stored in local register 0. With DD=1, the instruction operates in word units; the contents of the accumulator are extended to 16 bits and transferred to the local register er0. With DD=0, the operation is in byte units, and the lower 8 bits (A_L) are transferred to the local register r0. The mnemonic statements for these instructions follow.

```
ST   A, er0      word length operation

STB  A, r0       byte length operation
```

Providing two mnemonic terms for one machine code has two advantages. The first advantage is that the programmer can transmit his intention to the assembler: "the operation should be in word or byte length." The second advantage is that when the programmer or a third person reviews the source program, the unit of the operation is easy to see.

Even if the assembler understands the programmer's intention, the assembler must have some capability to verify the unit of operation. For instance, the programmer intends to use byte units in the store instruction and enters in the source program as shown. However, if DD=1 at the time of

```
STB  A, r0
```

program execution, the CPU will execute the instruction in word unit contrary to the programmer's intention. It is desirable for the assembler to provide some security means to verify the consistency between the programmer's intention and the DD value. However, the assembler cannot obtain the DD value because DD is specified by the program as explained later in detail.

As a supplementary checking means, the assembler provides the USING DATA pseudo instruction by which the programmer indicates to the assembler the value of DD to be used. To implement this process, the programmer enters

```
"USING DATA WORD"
```

in the source program which is equivalent to the message "Following this statement, DD=1 in this program" by the programmer. Similarly, the entry of

```
"USING DATA BYTE"
```

is equal to the message "Following this statement, DD=0 in this program". When DD is given by the USING DATA instruction, the assembler verifies the DD dependent instructions for consistency between the data unit of the instruction and DD; an error warning is issued if inconsistencies are found.

If no such verification of DD consistency is required, the statement "USING DATA ANY" is entered in the program. The USING DATA statement provides some degree of verification of consistency regarding the data length. However, it does not guarantee that the program is executed with the DD value specified by the USING DATA pseudo instruction. Please note that no error messages from the assembler does not guarantee consistency. Final responsibility rests with the programmer.

(iv) Changing the DD's value

The DD's value can be reentered in three ways. Initially, the DD's value is reset directly by PSW access instructions when they are executed. Alternatively, DD is reentered automatically by the specific instructions that influence DD. DD can also be initialized by reset processing.

1. Directly Accessing PSW

Since DD is allocated to bit 12 of the PSW, it can be changed directly by PSW access instructions, the following are some of the practical examples.

ANDB PSWH, 11101111B ..... DD is reset, remaining bits unaffected

ORB PSWH, 00010000B ..... DD is set, remaining bits unaffected

XORB PSWH, 00010000B ..... DD is inverted, remaining bits unaffected

SB PSWH. 4 (or SB DD) ..... DD is set

RB PSWH. 4 (or RB DD) ... DD is reset

2. Specific Instructions Which Affect DD

When the following instructions are executed, DD is set automatically.

Commands that Set DD

L	A, obj	load instruction of word length for accumulator
MOV	A, obj	move instruction of word length for accumulator
CLR	A	clear instruction of word length for accumulator
POPS	A	pop (from system stack) instruction of word length for accumulator
EXTND	A	extend code instruction for accumulator

When the following instructions are executed, DD is reset automatically.

Commands that Reset DD

LB	A, obj	load instruction of byte length for accumulator
MOVB	A, obj	move instruction of byte length for accumulator
CLRB	A	clear instruction of byte length for accumulator

As shown in the above lists, the instructions that affect DD all transport the data from somewhere to the accumulator except for EXTND. When the concerned data is word long, DD is set, and DD is reset when the data is byte long. The instructions that are influenced by DD have been described in (iii). Each instruction in the group somehow operates on the accumulator. To operate on the accumulator, it is necessary to transfer the data from some place to the accumulator prior to the operation. For data which is transferred in word length, the accumulator will operate in word length. Similarly for data which is transferred in byte lengths, the accumulator will also operate in byte lengths. Thus, the matching of DD and the unit of the data is maintained.

3. Initialization by Reset Processing

DD is initialized or set to 0 by the reset processing. The reset processing is executed for the following

1. A proper reset pulse is applied to the $\overline{\text{RESET}}$ pin.

2. A BRK instruction is executed.

3. Overflow exists in WDT (watch dog timer).

«Caution: Setting DD at Program Interupts»

The interrupt procedure does not set or reset DD. DD is unrelated to the processes occuring following an interupt, hence the value of DD cannot be preddicted. When the interrupt procedure includes instructions that are affected by DD, it becomes necessary to initialize the flag. But the contents of the flag (namely

PSW) need not be saved, because they are saved automatically at the time of interrupt and returned by RTI (return from interrupt) instruction.

2.3 Concept of Segments

The concept of segments is used for managing the data of the assembler and the linker. As mentioned previously, the MSM66201 provides two memory spaces, the program and data memory spaces. When the programmer codes the assembler source program, he must tell the assembler in which memory space the entry is allocated. Then, the assembler employs segments that correspond to the physical memory space. Segments may be considered as logical spaces. The code segment of the assembler corresponds to the program space, while the data and bit segments correspond to the data space. For example, the program statement "Now, the following entry is to be saved in the program memory" becomes the assembler statement "Now the following entry is to be saved in the code segment". Thus the code segment is almost synonymous to the program memory.

The situation for the data memory is slightly different. The data memory is usually accessed using word or byte units, and occasionally a bit unit is used such as for setting flags. Such flag may be identified as "nth bit of address q" or "bit address n" which is more efficient in managing data (in the view of some programmers).

In the bit segment, bit 0 of byte 0 in the data memory is simply called the bit 0 address and the following bits are numerically sequenced in bit units as shown in Figure 1-9.

In the data segment, the data space is addressed in byte units, so the bit segment may seem to be created only to define the flags in an extreme sense.

The absolute segment is used for the entry of the data space of fixed address such as the vector table and VCAL table; it is also used when a certain data space is allocated to specific addresses for program convenience. The relocatable segment is used for the entry of the codes and the space that do not require specific addresses. The use of relocatable segment is recommended for programs in which many programmers are involved simultaneously, or for programs intended to be a submodule of a future program.

									Byte location
D	7	6	5	4	3	2	1	0	0
A	F	E	D	C	B	A	9	8	1
T	17	16	15	14	13	12	11	10	2
A	1F	1E	1D	1C	1B	1A	19	18	3
	27	26	25	24	23	22	21	20	4
M E M O									
R	7FFF7	7FFF6	7FFF5	7FFF4	7FFF3	7FFF2	7FFF1	7FFF0	FFFE
Y	7FFFF	7FFFE	7FFFD	7FFFC	7FFFB	7FFFA	7FFF9	7FFF8	FFFF
Bit location	7	6	5	4	3	2	1	0	

Figure 1-9. Bit Addresses

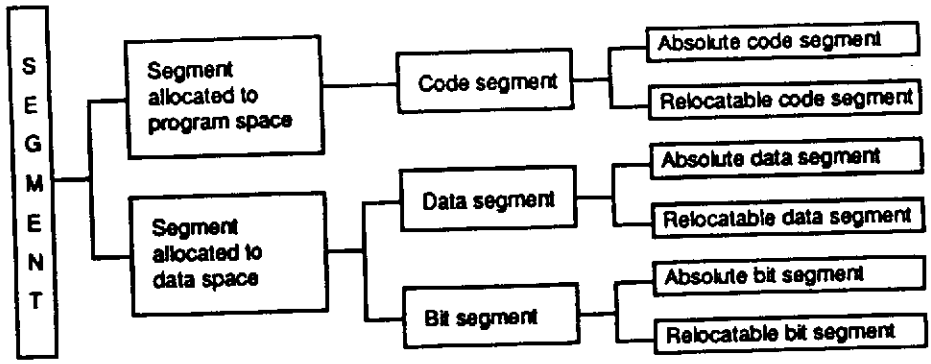


Figure 1-10 Segment Classification

The concept and outline of segments have been described so far. The following section describes the entry of segments to a real source program.

«Description of Absolute Segments»

For the assembler, three pseudo instructions are available to specify the segment type. Prior to the segment entry, these pseudo instructions: (CESG, DSEG, and BSEG) are entered to declare the segment type.

CSEG	This pseudo instruction declares that the following entries are to be located in the absolute code segment.
DSEG	This pseudo instruction declares that the following entries are to be located in the absolute data segment.
BSEG	This pseudo instruction declares that the following entries are to be located in the absolute bit segment

$$\left\{\begin{matrix}\text{CSEG}\\\text{DSEG}\\\text{BSEG}\end{matrix}\right\}\ \text{[AT starting address]}$$

The absolute code segment is used to enter the vector table, the VCAL table, and the ROM table, which is accessed by ROM reference instructions. The real statements entered will be labels, pseudo instructions which specify the data of word length (DW) and the data of byte length (DB), and the storage defining pseudo instruction (DS).

Refer to 2.1.1 for actual entry of statements.

When the relocatable assembler is used, the program main body (the instruction mnemonics of the MSM66201) is almost always entered in the relocatable code segment.

The actual entry of absolute data segment is explained here. The absolute data segment is entered to assign the data space to a specific addresses--for example, to assign the data space for global use to Page 0, or to allocate the stack space to certain fixed addresses. The entry of the absolute data segment is usually performed by the label and storage defining pseudo instruction (DS).

In Figure 1-11, the example entry assigns data of word length to WK_0, WK_1, WK_2, WK_3, and data of byte length to BUF_0, BUF_1, BUF_2, BUF_3 starting at the address $00C0_H$. When only the label values of the preceding example are desired to be defined, the DATA pseudo instruction may be used. The corresponding entry is shown in Figure 1-12. The latter entry differs from the former entry in that data space allocation is not possible for the latter. When the relocatable assembler is used with the latter entry, another relocatable segment might be allocated to the space behind $00C0_H$. This may be a concern,

while no such concern exists for the absolute assembler. DATA pseudo instruction is used for suballocation of the general data space or renaming of the label. An example of suballocation is shown in Figure 1-13. By placing the entry in Figure 1-13 after the entry in Figure 1-11, a data space of word length is suballocated to the low and high byte spaces.

```
DSEG    AT      00C0H
;
WK_0:   DS      2       ; Space for word long data
WK_1:   DS      2
WK_2:   DS      2
WK_3:   DS      2
;
BUF_0:  DS      1       ; Space for byte long data
BUF_1:  DS      1
BUF_2:  DS      1
BUF_3:  DS      1
        :
```

Figure 1-11. Absolute Data Segment

```
WK_0:   DATA    00C0H
WK_1:   DATA    WK_0+2
WK_2:   DATA    WK_0+4
WK_3:   DATA    WK_0+6
;
BUF_0:  DATA    0C08H
BUF_1:  DATA    BUF_0+1
BUF_2:  DATA    BUF_0+2
BUF_3:  DATA    BUF_0+3
        :
```

Figure 1-12. Specifying by DATA Pseudo Instruction

```
WK_0L   DATA    WK_0    ; Low byte of WK_0
WK_0H   DATA    WK_0+1  ; High byte of WK_0
WK_1L   DATA    WK_1    ; Low byte of WK_1
WK_1H   DATA    WK_1+1  ; High byte of WK_1
WK_2L   DATA    WK_2    ; Low byte of WK_2
WK_2H   DATA    WK_2+1  ; High byte of WK_2
WK_3L   DATA    WK_3    ; Low byte of WK_3
WK_3H   DATA    WK_3+1  ; High byte of WK_4
        :
```

Figure 1-13. Specifying by DATA Pseudo Instruction

As the last item described in this discussion on absolute segments, an example of a real program entry of bit segments is introduced.

The entry of a bit segment is performed by the label and bit long space defining pseudo instruction (DBIT), just as the entry of the data segment is done. The exact address arrangement of the bit segment is as shown in Figure 1-9. It is rather unwieldy to enter the address behind the AT portion of the BSEG pseudo instruction or the operand of the ORG pseudo instruction that controls the location counter directly with the bit addresses. For instance, bit 2 of byte address 1 is bit address 10. It is coded as 0001H.2, hence the following relationship exists.

address bit = address byte. n = address byte x 8 + n

An example of the entry of bit segments is described here. The following entry specifies the value of the four flags FLG_0, FLG_1, FLG_2, and FLG_3 starting at bit 0 of byte address $D0_H$.

```
            BSEG AT   00D0H.0
;
FLG_0:      DBIT      1
FLG_1:      DBIT      1
FLG_2:      DBIT      1
FLG_3:      DBIT      1
```

Figure 1-14. Entry of Absolute Bit Segment

In Figure 1-14, the first line statement "DSEG AT 00D0H.0" may be substituted by the statement "BSEG AT 0104".

The absolute bit segment may overlap partially or fully the absolute data segment. For example, the word long data space WK_0 is defined as the data segment; this is shown in Figure 1-13. For convenience, it became necessary for the lower four bits starting at WK_0 to be accessed using bit units, and the four bits are identified in the bit unit as WK_00, WK_01, WK_02 and WK_03. The new entry is easily made as shown in Figure 1-17 where the bit segment starts at WK_0.0. The assembler does not issue a warning for the overlapping condition.

There are two ways to specify the symbol which has the attribute of a bit segment. One way is to specify the label within the bit segment as shown in Figure 1-15, and the other way is to specify the label directly using a BIT pseudo instruction. The BIT pseudo instruction imparts values and the bit segment attribute to the symbol as shown in Figure 1-16.

```
            BSEG AT   WK_0.0
;
WK_00:      DBIT      1
WK_01:      DBIT      1
WK_02:      DBIT      1
WK_03:      DBIT      1
```

Figure 1-15. Bit Segment is Overlaid on Data Segment of Figure 1-12

```
WK_00:    BIT    WK_0.0
WK_01:    BIT    WK_0.1
WK_02:    BIT    WK_0.2
WK_03:    BIT    WK0.3
```

Figure 1-16. Equivalent Result (to Figure 1-15) Obtained by BIT Pseudo Instruction

[NOTE] The description of the coding formats and the functions of the assembler pseudo instructions given here is not complete because the main objective is to explain the concept of segments. For the details of the coding formats and the functions of the assembler pseudo instructions, refer to the assembler manual.

-Description of Relocatable Segments-

The entry of relocatable segments is described here, and it is more complex than the entry of the absolute segment. For the absolute segment, the entry is started by simply stating the pseudo instruction such as CSEG, DSEG or BSEG. For the relocatable segment, the name and type of segment are declared by the SEGMENT pseudo instruction, and even the type of relocation is declared if necessary. The entry format of the SEGMENT pseudo instruction is shown below.

```
segment name    SEGMENT    segment type    [relocation type]
```

The square bracket, [ ], for the relocation type implies that this entry in the statement is optional. This segment defining statement can be placed anywhere in the program as long as it precedes the RSEG pseudo instruction. However, it is usually placed at the head of program with other symbol defining statements.

The segment type specifies the segment to be defined: either code, data, or bit segment. The relocation type means the restricting factors when the linker allocates the segment to the memory addresses. The segment and relocation types are shown below.

Segment Type

CODE	defined segment is code segment
DATA	defined segment is data segment
BIT	defined segment is bit segment

Relocation Type

UNIT	match segment head with unit boundary
WORD	match segment head with word boundary
OCT	match segment head with 8-byte boundary
PAGE	match segment head with page head
NUMBERn	match segment head with boundary of n units where n=1, 2, 4, 8, . . . 2048 (power of 2)
INPAGE	match segment head within page

In the following description, the unit means the smallest unit of segment used in the accessing operation involved; hence the unit for the code and data segments is a byte, while it is a bit for the bit segment.

The UNIT in the relocation type above means "allocate any place". Without declaration of the relocation type listed in the above table, the assembler assumes that the segment possesses the UNIT attributes and processes the segment accordingly. Therefore, UNIT is not usually used unless its explicit understanding is required. As it is easily understood from the preceding discussions, the attributes of WORD and OCT concern the word boundary and the local register accordingly.

PAGE and INPAGE obviously take into account the page structure of the data space. NUMBER n permits the specification and allocation of boundaries which can not be defined by WORD, OCT and PAGE. Number, in this case, means so many units, and the unit is byte for code and data segments and bit for the bit segment. The attributes of UNIT, WORD, OCT and PAGE are inclusive in ascending order. For example, specifying OCT (its attributes) necessarily specifies all of the attributes of UNIT and WORD, and specifying PAGE satisfies all of the attributes of UNIT, WORD and OCT.

To enter the relocatable segment statement, the RSEG pseudo instruction whose format is shown below is used. This starts the entry of the segment indicated in the format.

RSEG	segment name

Following the entry of RSEG pseudo instruction, each segment is entered in the same manner as for the absolute segment, so no further description is given. It should be noted that overlapping the data and bit segments is not allowed for relocatable segments, while a similar overlapping for absolute segments is permissible.

For reference, an example entry is given in the following pages for an actual case where the absolute and relocatable segments coexists. Following this example, another important concept for the relocatable segment, "partial segment", will be explained.

(Example of Segment Entry)

```
            MAIN        SEGMENT   CODE        ; Main Routine
            MAIN_BUF    SEGMENT   DATA WORD   ; Buffer Area for Main Routine
            MAIN_FLG    SEGMENT   BIT         ; Flag Area for Main Routine
;
            STACK_SIZE  EQU       256         ; Stack Size
;
;           ***** Definition of Stack Space *****
            DSEG    AT    0280H-STACK_SIZE    ; Reserve a stack space with a size of
            DS      STACK_SIZE-1              ; STACK_SIZE starting at address 0280H
STACK_BTM   DS      1
;
;           ***** Vector Table *****
;           CSEG    AT    00C0H.0
            DW      START                     ; Reset vector
            DS      38                        ; Reserved for future
;
;           ***** VCAL Table *****
            CSEG    AT    0028H
            DS      16                        ; Reserved for future
;
;           ***** Flag Definition *****
            BSEG    AT    00C0H.0
FLG0:       DBIT    1
FLG1:       DBIT    1
FLG2:       DBIT    1
FLG3:       DBIT    1
            DBIT    4                         ; Reserved for future
;
;           ***** ROM Table *****
            CSEG    AT    3C00H
TBL_1:      DW      56DCH, 0013H, 9FA4H, 0A37H
TBL_2       DB      03H, 05H, 5FH, 87H, 0C4H, 0A1H
;
;           ***** WORK *****
            DSEG    AT    00C1H
WK_0:       DS      2
WK_1:       DS      2
WK_2:       DS      2
WK_3:       DS      2
;
```

(Continued on next page)

```
;           **********************
;           *  MAIN ROUTINE      *
;           **********************
            RSEG    MAIN

START:
            Main Routine Program Section

            RSEG    MAIN_BUF
            Main Routine Data Section

            RSEG    MAIN_FLG
            Main Routine Flag Section

                    :

            END
```

«Partial Segments»

Sometimes, when creating program code, relocatable segments are referred to as partial segments. Here is an example. Imagine the situation in which three programmers, A, B, and C, are in the process of designing a program as a team. Assume that a module in the program calls six subroutines which are named SUB1, SUB2, SUB3, SUB4, SUB5 and SUB6, respectively. Programmer A is assigned to SUB1 and SUB2, Programmer B to SUB3 and SUB 4, and Programmer C is assigned to SUB5 and SUB6. Each programmer works on his assigned subroutines independently using three different files. The completed files are assembled independently and the resulting relocatable objects are linked by the linker. The six subroutines, SUB1 through SUB6, are lumped in the code segment named SUBROOT that will be allocated to a continuous area of the memory space. Then, to implement the preceding plan, the programmers enter their statements in the source programs in the following manner.

1. Define the relocatable code segment named SUBROOT by SEGMENT pseudo instruction.
2. Each programmer enters RSEG SUBROOT prior to entering SUB1 through SUB6.

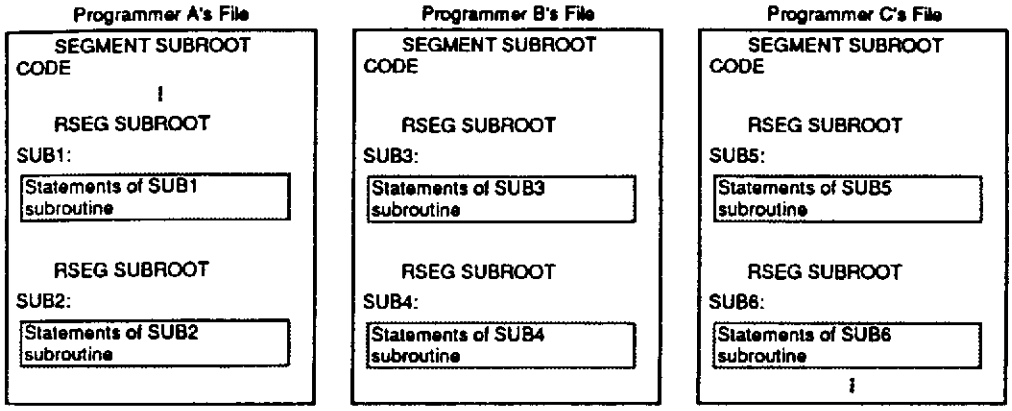


Figure 1-17. Example of Partial Segment Coding

The files, some of which are in shown in Figure 1-17, are assembled by the assembler RAS66K, and each of the resulting three object files contain two of the relocatable segments named SUBROOT. When the three object files are linked, the linker will find six relocatable segments of the identical name; however, the linker will consider them as one entity of SUBROOT, and allocate the six segments to a continuous memory space. Each of the six segments prior to the last allocation by the linker is called a partial segment. Each partial segment:

- is a relocatable segment
- has an identical segment name
- has identical segment attributes
- has identical relocation attributes
- implies the existence of plural segments

Among relocation attributes, the UNIT attribute should agree with all other attributes.

The linker regards the partial segments as the subdivisions of one larger segment whenever they are found, and allocates them to a continuous space.

Chapter 2. Addressing Modes

1. RAM Addressing Modes 2-1
- 1.1 Register Addressing 2-1
  - (1) Accumulator Addressing 2-1
  - (2) Pointing Register Addressing 2-1
  - (3) Control Register Addressing 2-2
  - (4) Local Register Addressing 2-2
  - (5) System Stack Pointer Addressing 2-3
- 1.2 Page Addressing 2-3
  - (1) Current Page Addressing 2-3
  - (2) Zero Page Addresssing 2-4
- 1.3 Pointing Register Indirect Addressing 2-5
  - (1) Data Pointer Indirect Addressing 2-5
  - (2) User Stack Pointer Indirect Addressing (with 8-bit Displacement) 2-5
  - (3) Index Register Indirect Addressing (with 16-bit Displacement) 2-6
- 1.4 Immediate Addressing 2-7
- ▪Advice About RAM Addressing▪ 2-7

2. ROM Addressing Modes 2-8
- 2.1 Direct Addressing 2-8
- 2.2 Indirect Addressing 2-8
  - (1) Single Indirect Addressing 2-8
  - (2) Double Indirect Addressing 2-9
  - (3) Indirect Addressing with 16-bit base 2-10

3. Bit Addressing Modes 2-12

4. Logical Bit Address Space 2-13

The MSM66201 provides two independent memory spaces, namely the program and data memory spaces. The MSM66201 accommodates the accessing of these memory spaces through a number of different addressing modes. The program memory space usually consists of ROM and is referred to as ROM space in this chapter. The data memory space generally consists of RAM and is referred to as RAM space. The RAM space includes registers, counters, ports and other pieces belonging to SFR, as well as the pointing and local registers. Addressing the RAM and ROM is referred to as RAM addressing and ROM addressing respectively.

1. RAM Addressing Modes

Except for a few cases, RAM addressing modes are generally used for the operands of the instructions. There are five major classes of RAM addressing modes, which are listed below:

1. Register addressing
2. Page addressing
3. Pointing register indirect addressing
4. Stack addressing
5. Immediate addressing

1.1 Register addressing

Register addressing, for accessing the contents of the registers, includes the following:

(1) Accumulator addressing

[Symbol]	A

The above is the addressing mode for the accumulator. It is applicable equally to word long and byte long data.

Example:

```
L     A, #0        LB    A, #0
MOV   A, er0       MOVB  A, #0
```

(2) Pointing register addressing

[Symbol]	DP	(data pointer)
	USP	(user stack pointer)
	X1, X2	(index register)

This addressing mode is for pointing registers, handling the registers individually. This addressing mode can be used in word length instruction only.

[NOTE] There are eight units of pointing registers (PR0 through PR7), and the units to be accessed by These modes are specified by the system control base (SCB).

Example:
```
L     A, DP          MOV   USP, A
MOV   X1, #2000H
ROL   X2
```

(3) Control register addressing

[Symbol]	LRB	(local register base)
	PSW	(program status word)
	PSWH	(program status word upper byte)
	PSWL	(program status word lower byte)

Above, the addressing modes for the local register base and program status word are shown. LRB and PSW are applicable only to word long data, while PSWH and PSWL are applicable only to byte long data.

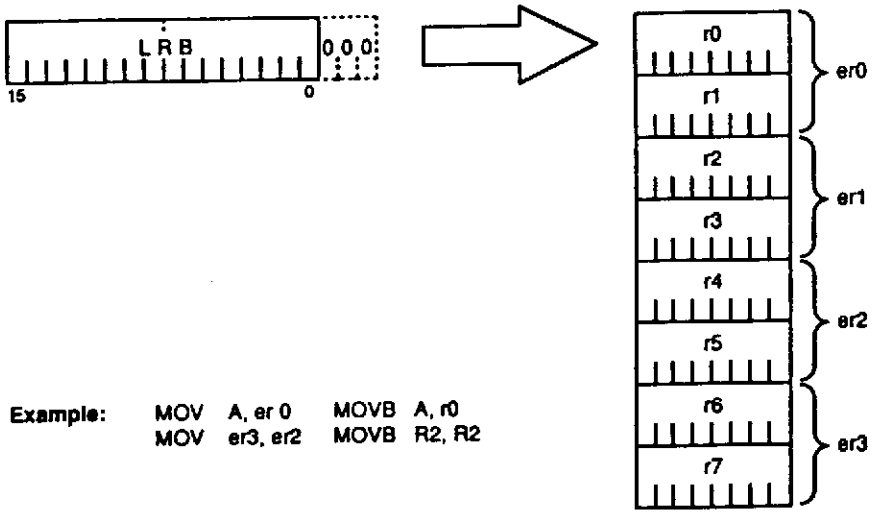
Example:
```
L     A, LRB         MOV   PSW, #00H
MOVB  PSWH, A        ROLB  PSWL
```

(4) Local register addressing

[Symbol]	r0—r7	(local register addressing)
	er0—er3	(extended local register addressing)

Above, the addressing modes for local registers are shown. The base address of the set of the local registers is generated by adding three bits of 0 to the contents of LRB (local register base).

Furthermore, r0—r7, and er0—er3, may respectively be stated as R0—R7, and ER0—ER3.



Example:
```
MOV   A, er 0        MOVB  A, r0
MOV   er3, er2       MOVB  R2, R2
```

(5) System stack pointer addressing

[Symbol]	SSP

Above, the addressing mode for the system stack pointer is shown. It is applicable only to word long data.

Example: MOV A, SSP MOV SSP, #1000H

1.2 Page addressing

In this addressing mode, an address is specified by offsetting from the starting address of the page. There are two modes of page addressing: current page addressing and zero page addressing modes. In current page addressing, the offset is given from the starting address of the page indicated by LRB (local register). In zero page addressing, the offset within Page 0 is not affected by LRB.

(1) Current page addressing

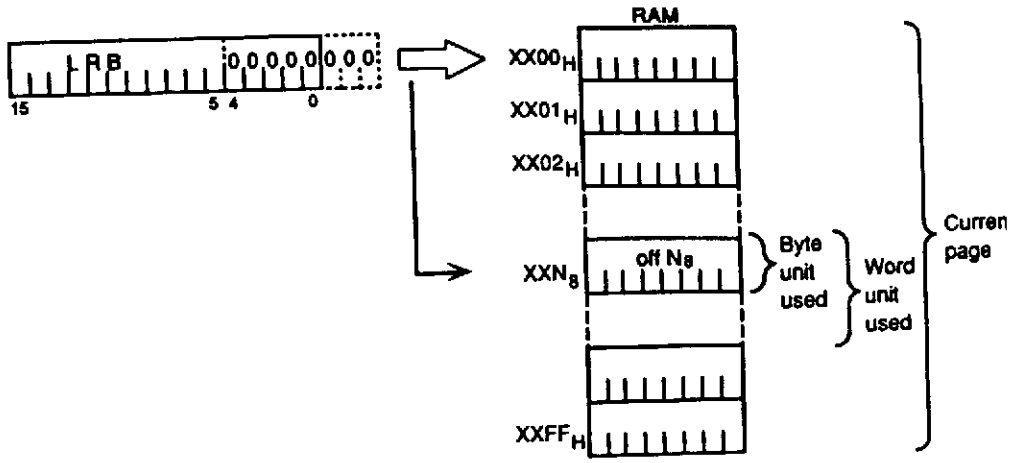
[Symbol]	off*1 N_8*2

As shown in the diagram, bits 12 through 5 of LRB indicate the page address in RAM space. Bits 4 through 0 of LRB with an additional 3 bits form the address (8 bits) within the page. All 8 bits are 0 for the base address of the page, and the offset (0 ~ 255) is given from the base address.

This addressing mode is applicable to word long data and byte long data.

[NOTE] 1 "off" is the agreed code to specify current page addressing. It may be written as "OFF". However, the latter expression resembles 0FFH for the immediate value. To avoid possible confusion, the use of "off" is recommended.

2 N_8 is the amount of the offset within the page. However, at the coding of the source program, the absolute, non-offset address itself or a symbol such as the label is entered. The programmer should not be concerned with offsetting.



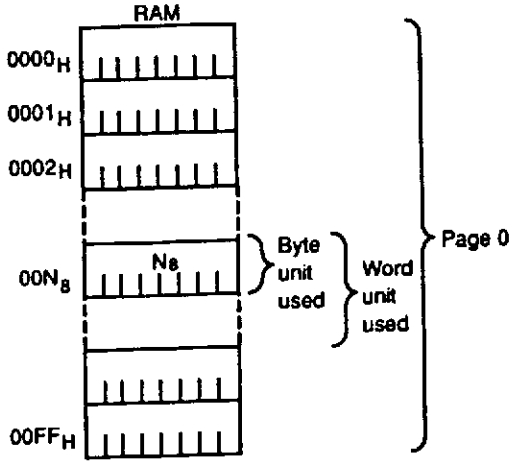
Example:

```
L     A, off 2000H      MOV   A, off WORK_01
LB    A, off 2000H      MOVB  A, off WORK_01
```

(2) Zero Page addressing

[Symbol]	N8

Above is the zero page addressing mode, where the offset is given within Page 0 (addresses 0 through $0FF_H$) in the RAM space. This addressing mode is applicable to word long and byte long data.



[NOTE] SFR is located in Page 0. To simplify the entry of addresses to SFR, the assembler defines the data address symbols, some of which are easily mistaken as one of the special assembler symbols used to describe the addressing modes. Care should be observed. For instance, A is a special assembler symbol used to describe addressing modes such as ACC for zero page addressing, and off ACC for current page addressing.

More examples of confusing symbols are listed below.

Example 1:	MOV	PSW, A	register addressing
	MOV	APSW, A	zero page addressing
	MOV	off APSW, A	current page addressing
Example 2:	MOV	LRB, A	register addressing
	MOV	ALRB, A	zero page addressing
	MOV	off ALRB, A	current page addressing
Example 3:	MOV	SSP, A	register addressing
	MOV	ASSP, A	zero page addressing
	MOV	off ASSP, A	current page addressing
Example 4:	MOVB	PSW, A	register addressing
	MOVB	APSW, A	zero page addressing
	MOVB	off APSW, A	current page addressing

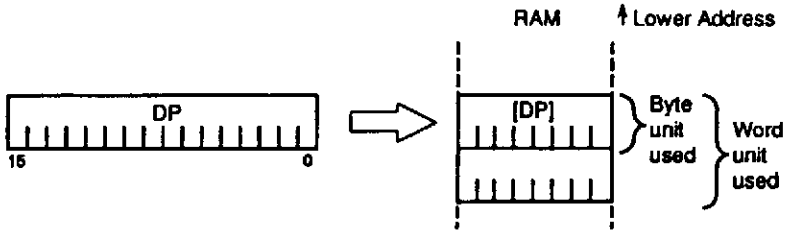
1.3 Pointing register indirect addressing

This addressing mode is performed by the pointing registers using indirect addressing, and it is used for both word and byte length instructions.

(1) Data pointer indirect addressing

[Symbol]	[DP]

The RAM space whose address is specified by the contents of the data pointer is accessed by this mode.



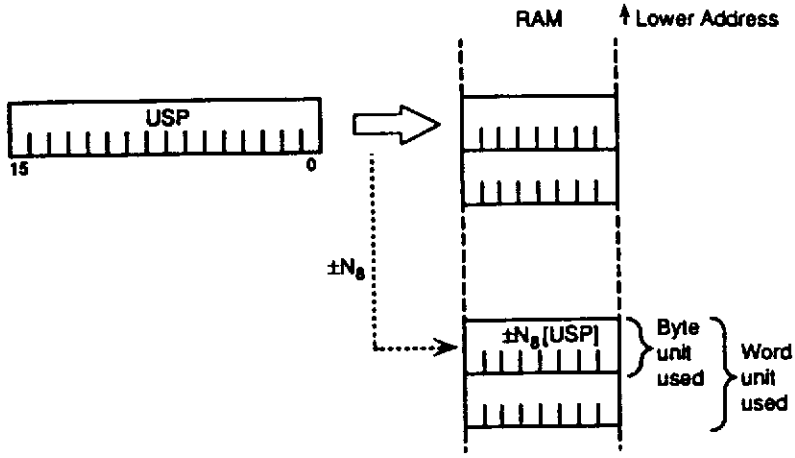
Example:

```
L     A, [DP]
LB    A, [DP]
```

(2) User stack pointer indirect addressing (with 8-bit displacement)

[Symbol]	±N8[USP]

The RAM space whose address is specified by the sum of the user stack pointer contents and $\pm N_8$ (displacement) is accessed. $\pm N_8$ is in the range from +127 to -128 or the equivalent range specified by symbols.



[NOTE] In the machine code, $\pm N_8$ is the displacement code sign which is given by the uppermost bit; however, the integer between +127 and -128 itself is used at program entry. For instance, the code for the displacement of -1 is FFH and the entry seems to be 0FFH[USP]; however, it is incorrect for the assembler, because its value exceeds +127. The correct entry in this case is -1 [USP].

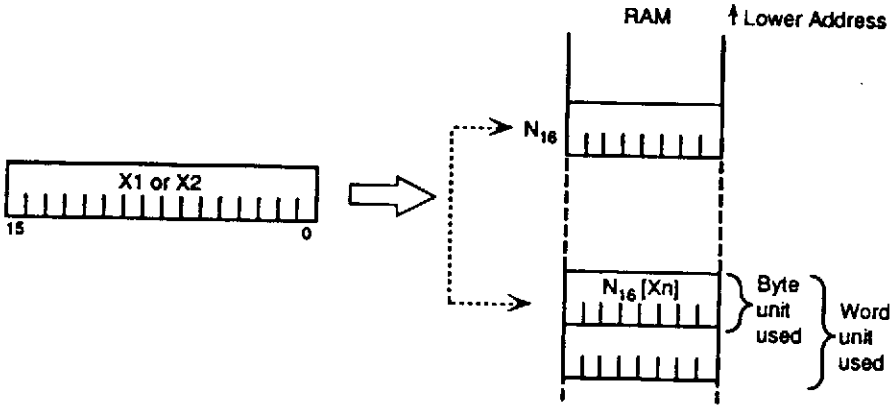
```
Example:    L     A, 70H[USP]        LB    A, -120[USP]
            ADD   A, DISP1[USP]
```

(3) Index register indirect addressing (with 16-bit base)

[Symbol]	N_{16}[X1], N_{16}[X2]

The RAM space whose address is the sum of the base address N_{16} (16 bits) and the contents of the index register is accessed.

The addition of N_{16} and X1, or N_{16} and X2 is performed in 16 bit length without sign, and the overflow is ignored. The maximum space addressable is 64K bytes; namely, the entire space of the *current bank*.



Example: L A, 5000H[X1] LB A, TBL_BASE[X2]

1.4 Immediate addressing

[Symbol]	#N_8, #N_{16}

The number or symbol in the operand itself specifies the address. The operand statement is #N_{16} for word long data and #N_8 for byte long data.

Example: MOV er0, #1234H MOVB r0, #12H

«Advice About RAM Addressing»

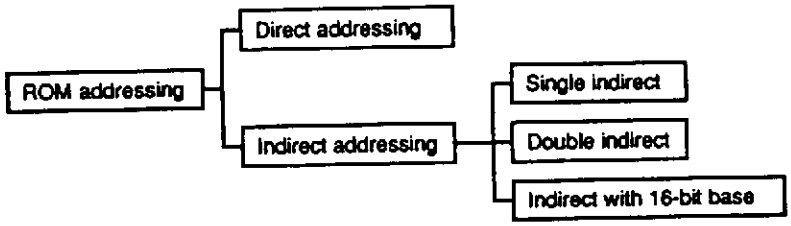
In RAM addressing modes, there is not a 64K-space direct addressing mode. The design of the CPU is based on the idea of processing a job completely within a page of 256 bytes. The data area required for the job is contained in the page as much as possible so that direct page addressing (off N8) can be used.

On the other hand, some data area need to be accessed by many processes. For example, a global variable area may need to be accessed by multiple subroutines. Special function registers (SFR) are another good example. This is why SFR is allocated to Page 0, and usually accessed by zero page addressing. Also, allocating a global variable area to Page 0 enables the use of zero page addressing regardless of the value of the local register (LRB).

When direct accesses are desired beyond the page limit of 256 bytes, the index-register indirect addressing with a 16-bit base (N_{16}[X1], N_{16}[X2]) is used. X1 and X2 are 0 while N_{16} is the address of the object in this case. However, the preceding addressing is not efficient because of the large numbers of bytes required for the instruction and the number of cycles it uses. Current page addressing used with careful data arrangement or register addressing with the local registers strategically placed in the areas should be employed as much as possible for better program efficiency.

2. ROM Addressing Modes

ROM addressing is used for accessing program memory space which consists of on-chip ROM and external program memory. ROM addressing is classified into two types, the direct and the indirect type. The indirect addressing type is further classified into three sub-types: single indirect addressing, double indirect addressing, and indirect addressing with a 16-bit base.



The ROM addressing modes are specified only in the operands of the ROM table reference instructions such as LC, LCB, CMPC, CMPCB.

The addressing modes are described below.

2.1 Direct addressing

[Symbol]	N^{*}_{16}

The reference address is directly specified with 16-bit immediate addressing N^{*}_{16}.

Example:

```
LC      A, 2000H
LC      A, DATA_TABLE
LCB     A, 3800H
```

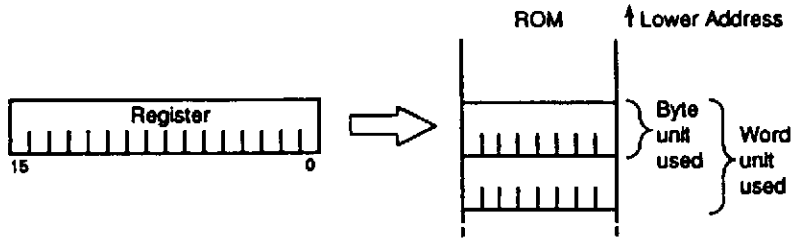
2.2 Indirect addressing

Indirect addressing includes: single indirect addressing, in which the register and the contents of the RAM specifies the object address; double indirect addressing, in which the contents of the RAM addressed by the pointing register specifies the object address; and indirect addressing with a 16-bit base.

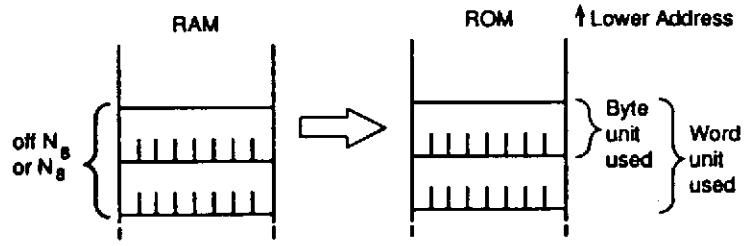
(1) Single indirect addressing

[Symbol]		
	1.	local register indirect ..... [er0], [er1], [er2], [er3]
	2.	pointing register indirect ..... [DP], [X1], [X2], [USP]
	3.	SSP indirect ..... [SSP]
	4.	LRB indirect ..... [LRB]
	5.	RAM indirect ..... [off N_8], [N_8]

For cases 1—4, the contents of each 16-bit register (er0—er3,DP, X1, X2, USP, SSP, LRB) is the address of the ROM to be accessed.



For case 5, the contents (16 bits) of the RAM which is specified by current or zero page addressing in word length is the address of the ROM to be accessed.



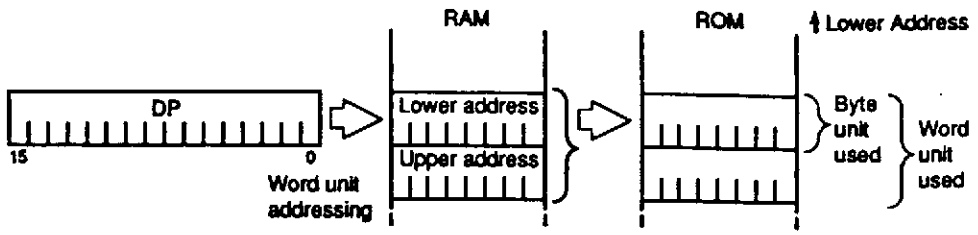
Example:

```
LC    A, [er0]          LC    A, [DP]
LC    A, [SSP]          LCB   A, [LRB]
LCB   A, [off 20H]      LC    A, [off DATA_TBL]
LCB   A, [ACC]
```

(2) Double indirect addressing

The section below describes double indirect addressing, in which the RAM addressed indirectly by the pointing register specifies the ROM indirectly.

[Symbol]		
	1. DP double indirect	[[DP]]
	2. USP double indirect (with signed 8-bit displacement)	[±N_8[USP]]
	3. Index register double indirect (with 16-bit base)	[N_{16}[X1]],[N_{16}[X2]]



Example:

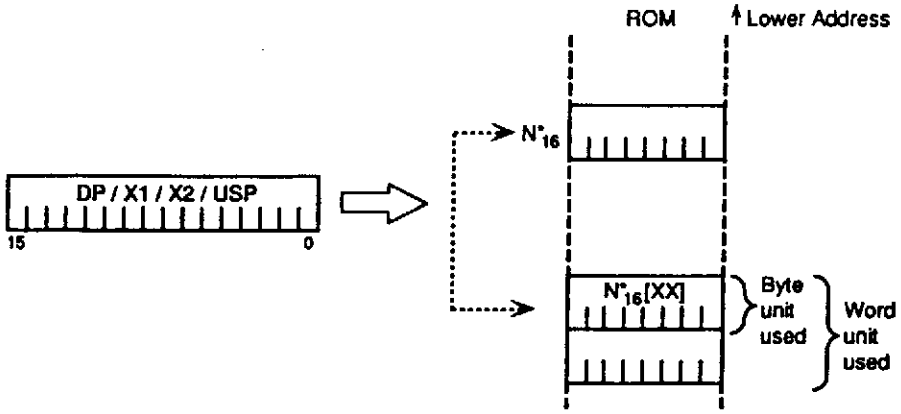
```
LC    A, [[DP]]
LC    A, [-10[USP]]
LCB   A, [1000H[X1]]
LC    A, [OFFSET_1[X2]]
```

(3) Indirect addressing with 16-bit base

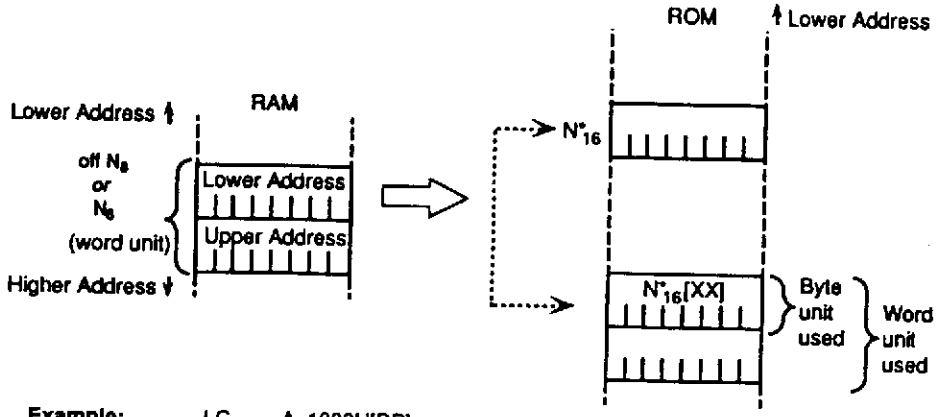
[Symbol]		
	1. pointing register indirect	N°_{16}[DP], N°_{16}[X1] N°_{16}[X2], N°_{16}[USP]
	2. RAM indirect	N°_{16}[off N_8], N°_{16}[N_8]

The sum of the immediate address (N°_{16}, the base), and the contents of the pointing register or the RAM (word long) is the address of the object.

Pointing Register Indirect

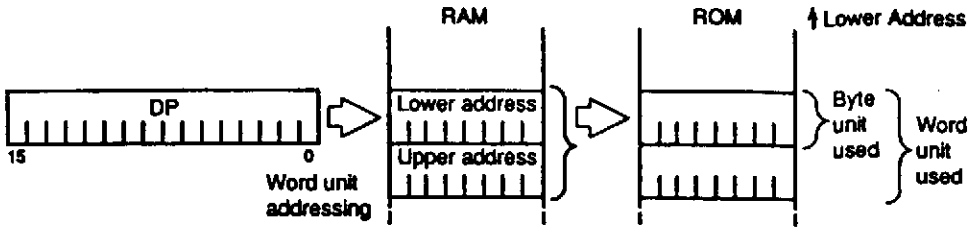


RAM Indirect



Example:

```
LC    A, 1000H[DP]
LC    A, EXT_ROM_BASE[X1]
LC    A, OFFSET_1[off DISP]
```



Example:

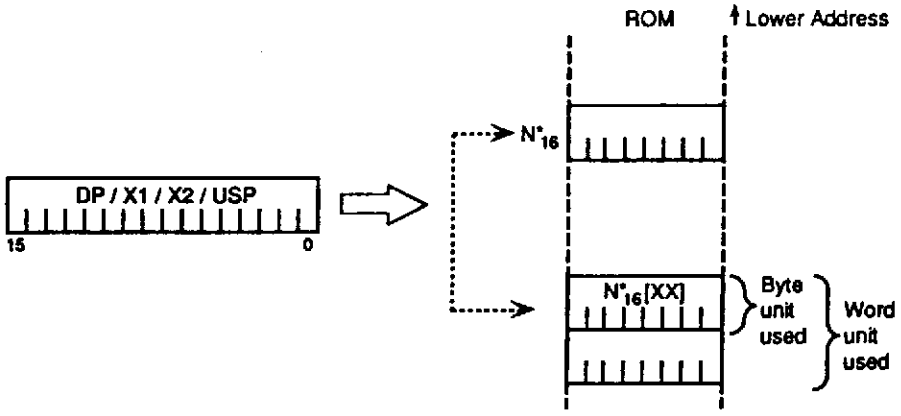
```
LC    A, [[DP]]
LC    A, [-10[USP]]
LCB   A, [1000H[X1]]
LC    A, [OFFSET_1[X2]]
```

(3) Indirect addressing with 16-bit base

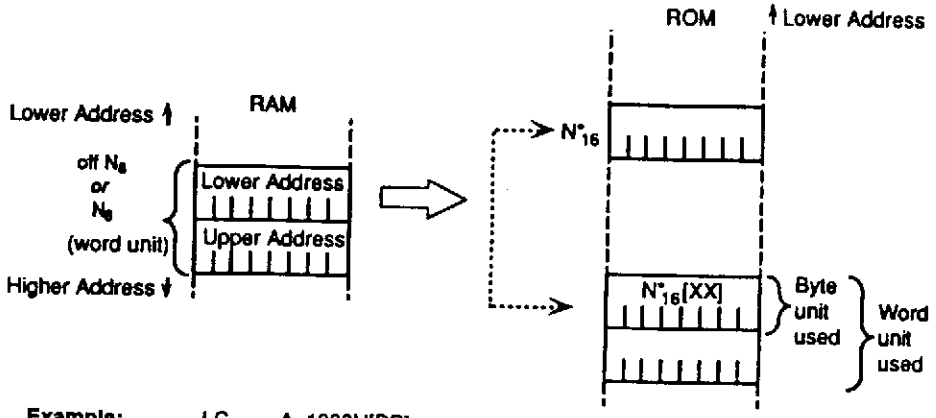
[Symbol]		
	1. pointing register indirect	N^{*}_{16}[DP], N^{*}_{16}[X1] N^{*}_{16}[X2], N^{*}_{16}[USP]
	2. RAM indirect	N^{*}_{16}[off N_8], N^{*}_{16}[N_8]

The sum of the immediate address (N^{*}_{16}, the base), and the contents of the pointing register or the RAM (word long) is the address of the object.

Pointing Register Indirect



RAM Indirect



Example:

```
LC    A, 1000H[DP]
LC    A, EXT_ROM_BASE[X1]
LC    A, OFFSET_1[off DISP]
```

3. Bit Addressing Modes

The addressing instructions of the MSM66201 includes those which permit accessing a specific bit desired. Examples of these are SBR and SB. The addressing of the desired bit is realized by combining the locations of byte and bit. The byte location is specified by RAM addressing. The bit location is specified either by the bits from 0 to 2 of the accumulator indirectly or by the operand directly. Note that usable RAM addresses are byte long in length.

(1) Specifying a bit using the accumulator indirectly

This mode appears in the SBR, TBR, and MBR instructions. Only the following RAM addressing modes are available to specify the byte location.

r0—r7, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Example:

```
SBR     r0              RBR     PSWH
TBR     off FLAGS       MBR     C, [DP]
```

(2) Specifying the bit directly

This mode appears in the SB, RB, and JRB instructions. Addressing is accomplished by specifying the object bit with the bit location (from 0 to 7) and byte location specified by the RAM addressing described in (1).

In the statement (shown below), the RAM addressing entry and the bit location entry are connected with a dot (.).

[Symbol] RAM addressing code.Bit location

Example:

```
SB      r0.2
RB      PSWH.BIT_POINT
JBR     off FLAGS.0, TIMER_LOOP
```

4. Logical Bit Address Space

Bit addressing for the MSM66201 is performed by specifying the bit address with the byte location specified by RAM addressing, and the bit location within the byte thus specified.

There is no physical bit address space for the MSM66201; however, its use for the assembler statement is acknowledged. Because the bit addressing statement with a combination of byte and bit locations is permitted for assemblers of the MSM80C51 series, such bit address statements are also recognized by the MSM66201 assembler. Strictly speaking, the bit address space is a logical concept, and its realization at the assembler level is accomplished by combining byte and bit locations.

The logical bit addresses of the MSM66201 start at address 0 (that is, location 0 of the data memory) and they progress in ascending order: address 1, address 2, and so on.

Logical Bit Address

									Byte location
DATA MEMORY	7	6	5	4	3	2	1	0	0
	F	E	D	C	B	A	9	8	1
	17	16	15	14	13	12	11	10	2
	1F	1E	1D	1C	1B	1A	19	18	3
	27	26	25	24	23	22	21	20	4
	7FFF7	7FFF6	7FFF5	7FFF4	7FFF3	7FFF2	7FFF1	7FFF0	FFFE
	7FFFF	7FFFE	7FFFD	7FFFC	7FFFB	7FFFA	7FFF9	7FFF8	FFFF
Bit location	7	6	5	4	3	2	1	0	

Chapter 3. Details of Instructions

1. Classification of Instructions 3-1
2. Instruction Set 3-5
3. Summary List of Instructions 3-180

The instructions of the MSM66201 may be classified in 16 groups as shown in the following table, "Classification of Instructions".

In the sets of instructions, the details of all instructions will be explained in alphabetical order.

Tables 3-1 through 3-35 at the end of this chapter are the summary list of the functions for all of the instructions.

Classification of Instructions

Group	Mnemonic	Instruction name
(1) Data Transfer Instructions	L	16-bit Load
	LB	8-bit Load
	ST	16-bit Store
	STB	8-bit Store
	MOV	16-bit Move
	MOVB	8-bit Move
	CLR	16-bit Clear
	CLRB	8-bit Clear
	SWAP	16-bit Swap
	SWAPB	8-bit Swap
	XCHG	16-bit Exchange
	XCHGB	8-bit Exchange
	XNBL	8-bit Nibble Exchange
(2) Stack Operation Instructions	PUSHS	16-bit Push (System Stack)
	POPS	16-bit Pop (System Stack)
(3) Rotate/Shift Instructions	ROL	16-bit Left Rotate
	ROLB	8-bit Left Rotate
	ROR	16-bit Right Rotate
	RORB	8-bit Right Rotate
	SLL	16-bit Left Shift
	SLLB	8-bit Left Shift
	SRL	16-bit Right Shift
	SRLB	8-bit Right Shift
	SRA	16-bit Arithmetic Right Shift
	SRAB	8-bit Arithmetic Right Shift
(4) Increment/Decrement Instructions	INC	16-bit Increment
	INCB	8-bit Increment
	DEC	16-bit Decrement
	DECB	8-bit Decrement
(5) ROM Table Reference Instructions	LC	16-bit ROM Reference
	LCB	8-bit ROM Reference
	CMPC	16-bit ROM Compare
	CMPCB	8-bit ROM Compare

Group	Mnemonic	Instruction name
(6) Arithmetic Operation Instructions	MUL	16-bit Multiply
	MULB	8-bit Multiply
	DIV	16-bit Divide
	DIVB	8-bit Divide
	ADD	16-bit Add
	ADDB	8-bit Add
	ADC	16-bit Add with Carry
	ADCB	8-bit Add with Carry
	SUB	16-bit Subtract
	SUBB	8-bit Subtract
	SBC	16-bit Subtract with Carry
	SBCB	8-bit Subtract with Carry
(7) Logical Operation Instructions	AND	16-bit Logical AND
	ANDB	8-bit Logical AND
	OR	16-bit Logical OR
	ORB	8- bit Logical OR
	XOR	16-bit Exclusive OR
	XORB	8-bit Exclusive OR
(8) Compare Instructions	CMP	16-bit Compare
	CMPB	8-bit Compare
(9) Decimal Adjust Instructions	DAA	8-bit Decimal Adjust (Add)
	DAS	8-bit Decimal Adjust (Subtract)
(10) Code Extend Instruction	EXTND	Coded Byte to Word Extend
(11) Bit Operation Instructions	SBR	Set Bit (Register Indirect Bit Addressing)
	RBR	Reset Bit (Register Indirect Bit Addressing)
	TBR	Test Bit (Register Indirect Bit Addressing)
	MBR	Transfer Bit (Register Indirect Bit Addressing)
	SB	Set Bit (Direct Bit Addressing)
	RB	Reset Bit (Direct Bit Addressing)
	MB	Transfer Bit (Direct Bit Addressing)

Group	Mnemonic	Instruction name
(12) Jump/Call Group Instructions	SJ	Short Jump
	J	16-bit (64K byte) Space Jump
	JC	Conditional Jump
	JBR	Bit Test and Jump
	JBS	Bit Test and Jump
	JRNZ	Loop
	SCAL	Short Call
	CAL	16-bit (64K byte) Space Call
	VCAL	Vector Call
	RT	Return from Normal Subroutine
	RTI	Return from Interrupt Routine
(13) Other Instructions	SC	Set Carry
	RC	Reset Carry
	BRK	Break (System Reset)
	NOP	No Operation

ADC A, obj

16-bit Add with Carry

obj

#N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A ← A + obj + C (word long)

Description

This instruction adds together the contents of the accumulator, the addressing object, and the carry (C) in a word long operation.

This instruction is influenced by the data descriptor (DD).
For the instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}	96	N_L	N_H				6	—
erN	18+N						3	7
DP	92	92					4	—
X1	90	92					4	—
X2	91	92					4	—
USP	A1	92					4	—
SSP	A0	92					4	—
LRB	A4	92					4	—
off N_8	97	N_8					4	9
N_8	B5	N_8	92				6	—
[DP]	B2	92					6	10
±N_8[USP]	B3	N_8	92				7	11
N_{16}[X1]	B0	N_L	N_H	92			8	12
N_{16}[X2]	B1	N_L	N_H	92			8	12

ADC obj, off N_8

16-bit Add with Carry

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + off N_8 + C (word long)

Description

This instruction adds together the contents of the addressing object, the data memory specified by current page addressing (off N_8), and the carry (C) in a word long operation.

The resulting sum (16 bits) is stored in the addressing object, and the carry over from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT ← obj	INT ← obj	EXT ← obj	EXT ← obj	← off N_8
							INT	EXT	INT	EXT	
A											
#N_{16}											
erN	44+N	93	N_8				7	—	—	20	
DP	92	93	N_8				7	16	—	—	
X1	90	93	N_8				7	16	—	—	
X2	91	93	N_8				7	16	—	—	
USP	A1	93	N_8				7	16	—	—	
SSP	A0	93	N_8				7	16	—	—	
LRB	A4	93	N_8				7	16	—	—	
off N'_8	B4	N'_8	93	N_8			9	—	—	23	
N'_8	B5	N'_8	93	N_8			9	18	—	—	
[DP]	B2	93	N_8				9	18	13	22	
$\pm N_8$[USP]	B3	N'_8	93				10	19	14	23	
N_{16}[X1]	B0	N_L	N_H	93	N_8		11	20	15	24	
N_{16}[X2]	B1	N_L	N_H	93	N_8		11	20	15	24	

ADC obj, #N_{16}

16-bit Add with Carry

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N'$_{16}$[X1], N'$_{16}$[X2]

Function

obj ← obj + #N_{16} + C (word long)

Description

This instruction adds together the contents of the addressing object (word long), the 16-bit immediate value (#N_{16}), and the carry (C).

The resulting sum (16 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	90	N_L	N_H			8	17
DP	92	90	N_L	N_H			8	—
X1	90	90	N_L	N_H			8	—
X2	91	90	N_L	N_H			8	—
USP	A1	90	N_L	N_H			8	—
SSP	A0	90	N_L	N_H			8	—
LRB	A4	90	N_L	N_H			8	—
off N_8	B4	N_8	90	N_L	N_H		10	20
N_8	B5	N_8	90	N_L	N_H		10	—
[DP]	B2	90	N_L	N_H			10	19
±N_8[USP]	B3	N_8	90	N_L	N_H		11	20
N'$_{16}$[X1]	B0	N'$_L$	N'$_H$	90	N_L	N_H	12	21
N'$_{16}$[X2]	B1	N'$_L$	N'$_H$	90	N_L	N_H	12	21

ADCB A, obj

8-bit Add with Carry

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

$A_L \leftarrow A_L + obj + C$ (byte long)

Description

This instruction adds together the contents of the lower byte of the accumulator (A_L), the addressing object (byte long), and the carry (C). The resulting sum (8 bits) is stored in the lower byte of the accumulator (A_L), and the carry from the highest (most significant) bit is stored in the carry (C).

This instruction is affected by the data descriptor (DD).
For the instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8	96	N_8					4	—
rN	18+N						3	5
PSWH	A2	92					4	—
PSWL	A3	92					4	—
off N_8	97	N_8					4	7
N_8	C5	N_8	92				6	—
[DP]	C2	92					6	—
±N_8[USP]	C3	N_8	92				7	9
N_{16}[X1]	C0	N_L	N_H	92			8	10
N_{16}[X2]	C1	N_L	N_H	92			8	10

ADCB obj, A

8-bit Add with Carry

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + A_L + C (byte long)

Description

This instruction adds together the contents of the lower byte of the accumulator (A_L), the addressing object (byte long), and the carry (C).

The resulting sum (8 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	91					5	9
PSWH	A2	91					5	—
PSWL	A3	91					5	—
off N_8	C4	N_8	91				7	12
N_8	C5	N_8	91				7	—
[DP]	C2	91					7	11
$\pm N_8$[USP]	C3	N_8	91				8	12
N_{16}[X1]	C0	N_L	N_H	91			9	13
N_{16}[X2]	C1	N_L	N_H	91			9	13

ADCB obj, off N_8

8-bit Add with Carry

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + off N_8 + C (byte long)

Description

This instruction adds together the contents of the addressing object, the data memory specified by current page addressing (off N_8), and the carry (C).

The resulting sum (8 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT ← obj	INT ← obj	EXT ← obj	EXT ← obj	INT ← off N_8	EXT ← off N_8	INT ← off N_8	EXT ← off N_8
A														
#N_8														
rN	20+N	93	N_8				7	—	—	14				
PSWH	A2	93	N_8				7	12	—	—				
PSWL	A3	93	N_8				7	12	—	—				
off N'_8	C4	N'_8	93	N_8			9	—	—	17				
N'_8	C5	N'_8	93	N_8			9	14	—	—				
[DP]	C2	93	N_8				9	14	11	16				
$\pm N'_8$[USP]	C3	N'_8	93	N_8			10	15	12	17				
N_{16}[X1]	C0	N_L	N_H	93	N_8		11	16	13	18				
N_{16}[X2]	C1	N_L	N_H	93	N_8		11	16	13	18				

ADCB obj, #N_8

8-bit Add with Carry

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + #N_8 + C (byte long)

Description

This instruction adds together the contents (byte long) of the addressing object, the 8-bit immediate value (#N_8), and the carry (C).

The resulting sum (8 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	90	N_8				6	11
PSWH	A2	90	N_8				6	—
PSWL	A3	90	N_8				6	—
off N'_8	C4	N'_8	90	N_8			8	14
N'_8	C5	N'_8	90	N_8			8	—
[DP]	C2	90	N_8				8	13
$\pm N'_8$[USP]	C3	N'_8	90	N_8			9	14
N_{16}[X1]	C0	N_L	N_H	90	N_8		10	15
N_{16}[X2]	C1	N_L	N_H	90	N_8		10	15

ADD A, obj

16-bit Add (user stack)

obj

#N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A←A+ obj (word long)

Description

This instruction adds the contents of the accumulator to the addressing object in a word long operation. The resulting sum (16 bits) is stored in the accumulator, and the carry from the highest (most significant) bit is stored in the carry (C).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}	86	N_L	N_H				6	—
erN	08+N						3	7
DP	92	82					4	—
X1	90	82					4	—
X2	91	82					4	—
USP	A1	82					4	—
SSP	A0	82					4	—
LRB	A4	82					4	—
off N_8	87	N_8					4	9
N_8	B5	N_8	82				6	—
[DP]	B2	82					6	10
±N_8[USP]	B3	N_8	82				7	11
N_{16}[X1]	B0	N_L	N_H	82			8	12
N_{16}[X2]	B1	N_L	N_H	82			8	12

ADD obj, off N_8

16-bit Add

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + off N_8 (word long)

Description

This instruction adds the contents of the addressing object to the contents (off N_8) of the data memory specified by current page addressing.

The resulting sum (16 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT ← obj	INT ← obj	EXT ← obj	EXT ← obj	INT ← off N_8	EXT ← off N_8	INT ← off N_8	EXT ← off N_8
A														
#N_{16}														
erN	44+N	83	N_8				7	—	—	20				
DP	92	83	N_8				7	16	—	—				
X1	90	83	N_8				7	16	—	—				
X2	91	83	N_8				7	16	—	—				
USP	A1	83	N_8				7	16	—	—				
SSP	A0	83	N_8				7	16	—	—				
LRB	A4	83	N_8				7	16	—	—				
off N'_8	B4	N'_8	83	N_8			9	—	—	23				
N'_8	B5	N'_8	83	N_8			9	18	—	—				
[DP]	B2	83	N_8				9	18	13	22				
$\pm N'_8$[USP]	B3	N'_8	83	N_8			10	19	14	23				
N_{16}[X1]	B0	N_L	N_H	83	N_8		11	20	15	24				
N_{16}[X2]	B1	N_L	N_H	83	N_8		11	20	15	24				

ADD obj, #N_{16}

16-bit Add

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N'$_{16}$[X1], N'$_{16}$[X2]

Function

obj ← obj + #N_{16} (word long)

Description

This instruction adds the contents of the addressing object (word long) to the 16-bit immediate value (#N_{16}).

The resulting sum (16 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	80	N_L	N_H			8	17
DP	92	80	N_L	N_H			8	—
X1	90	80	N_L	N_H			8	—
X2	91	80	N_L	N_H			8	—
USP	A1	80	N_L	N_H			8	—
SSP	A0	80	N_L	N_H			8	—
LRB	A4	80	N_L	N_H			8	—
off N_8	B4	N_8	80	N_L	N_H		10	20
N_8	B5	N_8	80	N_L	N_H		10	—
[DP]	B2	80	N_L	N_H			10	19
±N_8[USP]	B3	N_8	80	N_L	N_H		11	20
N'$_{16}$[X1]	B0	N'$_L$	N'$_H$	80	N_L	N_H	12	21
N'$_{16}$[X2]	B1	N'$_L$	N'$_H$	80	N_L	N_H	12	21

ADDB A, obj

8-bit Add

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

$A_L \leftarrow A_L$ + obj (byte long)

Description

This instruction adds the contents of the lower byte of the accumulator (A_L) to the addressing object (byte long). The resulting sum (8 bits) is stored in the lower byte of the accumulator (A_L), and the carry over from the highest (most significant) bit is stored in the carry (C).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8	86	N_8					4	—
rN	08 + N						3	5
PSWH	A2	82					4	—
PSWL	A3	82					4	—
off N_8	87	N_8					4	7
N_8	C5	N_8	82				6	—
[DP]	C2	82					6	—
±N_8[USP]	C3	N_8	82				7	9
N_{16}[X1]	C0	N_L	N_H	82			8	10
N_{16}[X2]	C1	N_L	N_H	82			8	10

ADDB obj, A

8-bit Add

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + A_L (byte long)

Description

This instruction adds the contents of the lower byte of the accumulator (A_L) to the addressing object (byte long).

The resulting sum (8 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_8								
rN	20+N	81					5	9
PSWH	A2	81					5	—
PSWL	A3	81					5	—
off N_8	C4	N_8	81				7	12
N_8	C5	N_8	81				7	—
[DP]	C2	81					7	11
$\pm N_8$[USP]	C3	N_8	81				8	12
N_{16}[X1]	C0	N_L	N_H	81			9	13
N_{16}[X2]	C1	N_L	N_H	81			9	13

ADDB obj, off N_8

8-bit Add

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + off N_8 (byte long)

Description

This instruction adds the contents of the addressing object to the data memory (off N_8) specified by current page addressing.

The resulting sum (8 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT (← obj)	INT (← off N_8)	EXT (← obj)	EXT (← off N_8)
A										
#N_8										
rN	20+N	83	N_8				7	—	—	14
PSWH	A2	83	N_8				7	12	—	—
PSWL	A3	83	N_8				7	12	—	—
off N'_8	C4	N'_8	83	N_8			9	—	—	17
N'_8	C5	N'_8	83	N_8			9	14	—	—
[DP]	C2	83	N_8				9	14	11	16
$\pm N'_8$[USP]	C3	N'_8	83	N_8			10	15	12	17
N_{16}[X1]	C0	N_L	N_H	83	N_8		11	16	13	18
N_{16}[X2]	C1	N_L	N_H	83	N_8		11	16	13	18

ADDB obj, #N_8

8-bit Add

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + #N_8 (byte long)

Description

This instruction adds the contents (byte long) of the addressing object to the 8-bit immediate value (#N_8).

The resulting sum (8 bits) is stored in the addressing object, and the carry from the highest (most significant) bit is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	80	N_8				6	11
PSWH	A2	80	N_8				6	—
PSWL	A3	80	N_8				6	—
off N'_8	C4	N'_8	80	N_8			8	14
N'_8	C5	N'_8	80	N_8			8	—
[DP]	C2	80	N_8				8	13
$\pm N'_8$[USP]	C3	N'_8	80	N_8			9	14
N_{16}[X1]	C0	N_L	N_H	80	N_8		10	15
N_{16}[X2]	C1	N_L	N_H	80	N_8		10	15

AND A, obj

16-bit logical AND

obj

#N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A ← A ∧ obj (word long)

Description

This instruction performs the word long logical AND operation on the contents of the accumulator and the addressing object. The result (16 bits) is stored in the accumulator.

This instruction is affected by the data descriptor (DD).
For the instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}	D6	N_L	N_H				6	—
erN	58+N						3	7
DP	92	D2					4	—
X1	90	D2					4	—
X2	91	D2					4	—
USP	A1	D2					4	—
SSP	A0	D2					4	—
LRB	A4	D2					4	—
off N_8	D7	N_8					4	9
N_8	B5	N_8	D2				6	—
[DP]	B2	D2					6	10
±N_8[USP]	B3	N_8	D2				7	11
N_{16}[X1]	B0	N_L	N_H	D2			8	12
N_{16}[X2]	B1	N_L	N_H	D2			8	12

AND obj, A

16-bit logical AND

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∧ A (word long)

Description

This instruction performs the word long logical AND operation on the contents of the addressing object and the contents of the accumulator.

The result (16 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	D1					5	13
DP	92	D1					5	—
X1	90	D1					5	—
X2	91	D1					5	—
USP	A1	D1					5	—
SSP	A0	D1					5	—
LRB	A4	D1					5	—
off N_8	B4	N_8	D1				7	16
N_8	B5	N_8	D1				7	—
[DP]	B2	D1					7	15
$\pm N_8$[USP]	B3	N_8	D1				8	16
N_{16}[X1]	B0	N_L	N_H	D1			9	17
N_{16}[X2]	B1	N_L	N_H	D1			9	17

AND obj, off N_8

16-bit logical AND

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∧off N_8 (word long)

Description

This instruction performs the word long logical AND operation on the contents of the addressing object and the data memory (off N_8) specified by current page addressing.

The result (16 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES				← obj
	1	2	3	4	5	6	INT	EXT	INT	EXT	← off N_8
A											
#N_{16}											
erN	44+N	D3	N_8				7	—	—	20	
DP	92	D3	N_8				7	16	—	—	
X1	90	D3	N_8				7	16	—	—	
X2	91	D3	N_8				7	16	—	—	
USP	A1	D3	N_8				7	16	—	—	
SSP	A0	D3	N_8				7	16	—	—	
LRB	A4	D3	N_8				7	16	—	—	
off N'_8	B4	N'_8	D3	N_8			9	—	—	23	
N'_8	B5	N'_8	D3	N_8			9	18	—	—	
[DP]	B2	D3	N_8				9	18	13	22	
$\pm N'_8$[USP]	B3	N'_8	D3	N_8			10	19	14	23	
N_{16}[X1]	B0	N_L	N_H	D3	N_8		11	20	15	24	
N_{16}[X2]	B1	N_L	N_H	D3	N_8		11	20	15	24	

AND obj, #N_{16}

16-bit logical AND

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N'_{16}[X1], N'_{16}[X2]

Function

obj ← obj ∧ #N_{16} (word long)

Description

This instruction performs the logical AND operation on the contents of the addressing object (word long) and the 16-bit immediate value (#N_{16}).

The result (16 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	D0	N_L	N_H			8	17
DP	92	D0	N_L	N_H			8	—
X1	90	D0	N_L	N_H			8	—
X2	91	D0	N_L	N_H			8	—
USP	A1	D0	N_L	N_H			8	—
SSP	A0	D0	N_L	N_H			8	—
LRB	A4	D0	N_L	N_H			8	—
off N_8	B4	N_8	D0	N_L	N_H		10	20
N_8	B5	N_8	D0	N_L	N_H		10	—
[DP]	B2	D0	N_L	N_H			10	19
$\pm N_8$[USP]	B3	N_8	D0	N_L	N_H		11	20
N'_{16}[X1]	B0	N'_L	N'_H	D0	N_L	N_H	12	21
N'_{16}[X2]	B1	N'_L	N'_H	D0	N_L	N_H	12	21

\NDB A, obj

8-bit logical AND

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

$A_L \leftarrow A_L \wedge$ obj (byte long)

Description

This instruction performs the logical AND operation on the contents of the lower byte of the accumulator (A_L) and the addressing object (byte long). The result (8 bits) is stored in the lower byte of the accumulator (A_L).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8	D6	N_8					4	—
rN	58 + N						3	5
PSWH	A2	D2					4	—
PSWL	A3	D2					4	—
off N_8	D7	N_8					4	7
N_8	C5	N_8	D2				6	—
[DP]	C2	D2					6	—
±N_8[USP]	C3	N_8	D2				7	9
N_{16}[X1]	C0	N_L	N_H	D2			8	10
N_{16}[X2]	C1	N_L	N_H	D2			8	10

ANDB obj, A

8-bit logical AND

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∧ A_L (byte long)

Description

This instruction performs the logical AND operation on the contents of the lower byte of the accumulator (A_L) and the contents (byte long) of the addressing object.

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	D1					5	9
PSWH	A2	D1					5	—
PSWL	A3	D1					5	—
off N_8	C4	N_8	D1				7	12
N_8	C5	N_8	D1				7	—
[DP]	C2	D1					7	11
$\pm N_8$[USP]	C3	N_8	D1				8	12
N_{16}[X1]	C0	N_L	N_H	D1			9	13
N_{16}[X2]	C1	N_L	N_H	D1			9	13

ANDB obj, off N_8

8-bit logical AND

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∧ off N_8 (byte long)

Description

This instruction performs the logical AND operation on the contents (byte long) of the addressing object and the data memory (off N_8) specified by current page addressing.

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT ← obj	INT ← off N_8	EXT ← obj	EXT ← off N_8
A										
#N_8										
rN	20+N	D3	N_8				7	—	—	14
PSWH	A2	D3	N_8				7	12	—	—
PSWL	A3	D3	N_8				7	12	—	—
off N'_8	C4	N'_8	D3	N_8			9	—	—	17
N'_8	C5	N'_8	D3	N_8			9	14	—	—
[DP]	C2	D3	N_8				9	14	11	16
$\pm N'_8$[USP]	C3	N'_8	D3	N_8			10	15	12	17
N_{16}[X1]	C0	N_L	N_H	D3	N_8		11	16	13	18
N_{16}[X2]	C1	N_L	N_H	D3	N_8		11	16	13	18

ANDB obj, #N_8

8-bit logical AND

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∧ N_8 (byte long)

Description

This instruction performs the logical AND operation on the contents (byte long) of the addressing object and the 8-bit long immediate value (#N_8).

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	D0	N_8				6	11
PSWH	A2	D0	N_8				6	—
PSWL	A3	D0	N_8				6	—
off N'_8	C4	N'_8	D0	N_8			8	14
N'_8	C5	N'_8	D0	N_8			8	—
[DP]	C2	D0	N_8				8	13
$\pm N'_8$[USP]	C3	N'_8	D0	N_8			9	14
N_{16}[X1]	C0	N_L	N_H	D0	N_8		10	15
N_{16}[X2]	C1	N_L	N_H	D0	N_8		10	15

BRK

Break (system reset)

obj

Function

SYSTEM RESET
PC ← (Vector Table 0002_H)

Description

The CPU, upon decoding this instruction, executes the system reset process whose specific details are given in the user's manual of the MSM66301.

Following the completion of the system reset, the contents of addresses 2 and 3 of the vector-table are transferred to the PC.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES
	BYTE						
	1	2	3	4	5	6	
	FF						13

CAL address

16-bit Space (64K bytes) Direct Call

obj

Function

(SSP) ← PC + 3
SSP ← SSP - 2
SF ← 0
PC ← address (16 bits)

Description

This is the direct call instruction for accessing the 64K byte program space. The address (in the instruction) is evaluated as a 16-bit address value (0000_H through $FFFF_H$) and becomes the address to be accessed by the CAL instruction.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	32	adr_L	adr_H				9	13

[NOTE] adr_L and adr_H indicate the lower 8 and upper 8 bits respectively.

CAL obj

16-bit Space (64K byte) Indirect Call

obj

[erN], [DP], [X1], [X2], [USP], [SSP], [LRB], [off N_8], [N_8], [[DP]], [$\pm N_8$[USP]], [N_{16}[X1]], [N_{16}[X2]]

Function

(SSP) ← PC + n
SSP ← SSP - 2
SF ← 0
PC ← obj (16 bits)

[NOTE] "n" indicates the number of the bytes for this instruction and it is affected by the addressing object. For the number of the bytes, n, refer to the section "CODES".

Description

This is the 16-bit (64K byte) space indirect call instruction, where the contents of the addressing object determine the destination address. The obj term in the function is generally enclosed in square brackets ([ ]) as shown in the obj section. The value inside the square brackets is the word long contents of the RAM that is addressed. The same contents of the RAM become the final address called.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	INT	EXT	EXT	← obj
							INT	EXT	INT	EXT	← (SSP)
A											
#N_{16}											
[erN]	44+N	23					8	12	12	16	
[DP]	92	23					8	12	—	—	
[X1]	90	23					8	12	—	—	
[X2]	91	23					8	12	—	—	
[USP]	A1	23					8	12	—	—	
[SSP]	A0	23					8	12	—	—	
[LRB]	A4	23					8	12	—	—	
[off N_8]	B4	N_8	23				10	14	15	19	
[N_8]	B5	N_8	23				10	14	—	—	
[DP]]	B2	23	N_8				10	14	14	18	
[$\pm N_8$[USP]]	B3	N_8	23				11	15	15	19	
[N_{16}[X1]]	B0	N_L	N_H	23			12	16	16	20	
[N_{16}[X2]]	B1	N_L	N_H	23			12	16	16	20	

CLR A

16-bit Clear

obj

Function

A ← 0 (word long)
DD ← 1

Description

This word long instruction clears the accumulator.

This instruction sets the data descriptor (DD), making DD=1.

This instruction functions identically to the instruction [L A, #0], including the effects on the flags. However, this instruction incurs comparatively fewer numbers of bytes and cycles.

Flags

Flags affected by execution:

ZF	CF	HC	DD
1			1

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	F9						2	—

CLR obj

16-bit Clear

obj

erN, DP, X1, X2,USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← 0 (word long)

Description

This word long instruction clears the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	15					4	12
DP	92	15					4	—
X1	90	15					4	—
X2	91	15					4	—
USP	A1	15					4	—
SSP	A0	15					4	—
LRB	A4	15					4	—
off N_8	B4	N_8	15				6	15
N_8	B5	N_8	15				6	—
[DP]	B2	15					6	14
$\pm N_8$[USP]	B3	N_8	15				7	15
N_{16}[X1]	B0	N_L	N_H	15			8	16
N_{16}[X2]	B1	N_L	N_H	15			8	16

CLRB A

8-bit Clear

obj

Function

$A_L \leftarrow 0$ (byte long)

Description

This instruction clears the lower byte of the accumulator (A_L), setting all 8 bits equal to 0.

This instruction resets the data descriptor, making DD=0.

This instruction functions identically to the instruction [LB A, #0] including the effects on the flags. However, this instruction incurs comparatively fewer numbers of bytes and cycles.

Flags

Flags affected by execution:

ZF	CF	HC	DD
1			0

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	FA						2	—

CLRB obj

8-bit Clear

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← 0 (byte long)

Description

This byte long instruction clears the addressing object, making all bits equal to 0.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	15					4	8
PSWH	A2	15					4	—
PSWL	A3	15					4	—
off N_8	C4	N_8	15				6	11
N_8	C5	N_8	15				6	—
[DP]	C2	15					6	10
$\pm N_8$[USP]	C3	N_8	15				7	11
N_{16}[X1]	C0	N_L	N_H	15			8	12
N_{16}[X2]	C1	N_L	N_H	15			8	12

CMP A, obj

16-bit Compare

obj

#N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

A - obj (word long)

Description

This instruction compares the contents of the accumulator to the addressing object in a word long operation. In the actual operation, the contents of the addressing object are subtracted from the contents of the accumulator, and the result of this subtraction determines the status of the zero flag (ZF) and the carry flag (CF). The contents of the accumulator are unchanged.

This instruction is affected by the data descriptor (DD).
For the instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•		

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}	C6	N_L	N_H				6	—
erN	48+N						3	7
DP	92	C2					4	—
X1	90	C2					4	—
X2	91	C2					4	—
USP	A1	C2					4	—
SSP	A0	C2					4	—
LRB	A4	C2					4	—
off N_8	C7	N_8					4	9
N_8	B5	N_8	C2				6	—
[DP]	B2	C2					6	10
$\pm N_8$[USP]	B3	N_8	C2				7	11
N_{16}[X1]	B0	N_L	N_H	C2			8	12
N_{16}[X2]	B1	N_L	N_H	C2			8	12

CMP obj, A

16-bit Compare

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj - A (word long)

Description

This instruction compares the contents of the addressing object to the accumulator in a word long operation.

In the actual operation, the contents of the accumulator are subtracted from the contents of the addressing object and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	C1					5	13
DP	92	C1					5	—
X1	90	C1					5	—
X2	91	C1					5	—
USP	A1	C1					5	—
SSP	A0	C1					5	—
LRB	A4	C1					5	—
off N_8	B4	N_8	C1				7	16
N_8	B5	N_8	C1				7	—
[DP]	B2	C1					7	15
$\pm N_8$[USP]	B3	N_8	C1				8	16
N_{16}[X1]	B0	N_L	N_H	C1			9	17
N_{16}[X2]	B1	N_L	N_H	C1			9	17

CMP obj, off N_8

16-bit Compare

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj - off N_8 (word long)

Description

This instruction compares the contents of the addressing object to the data memory specified by current page addressing (off N_8) in a word long operation.

In the actual operation, the contents of the data memory specified by the current page addressing are subtracted from the contents of the addressing object and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF). The contents of the addressing object are unchanged.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	INT	EXT	EXT	← obj
							INT	EXT	INT	EXT	← off N_8
A											
#N_{16}											
erN	44+N	C3	N_8				7	—	—	20	
DP	92	C3	N_8				7	16	—	—	
X1	90	C3	N_8				7	16	—	—	
X2	91	C3	N_8				7	16	—	—	
USP	A1	C3	N_8				7	16	—	—	
SSP	A0	C3	N_8				7	16	—	—	
LRB	A4	C3	N_8				7	16	—	—	
off N'_8	B4	N'_8	C3	N_8			9	—	—	23	
N'_8	B5	N'_8	C3	N_8			9	18	—	—	
[DP]	B2	C3	N_8				9	18	13	22	
$\pm N'_8$[USP]	B3	N'_8	C3	N_8			10	19	14	23	
N_{16}[X1]	B0	N_L	N_H	C3	N_8		11	20	15	24	
N_{16}[X2]	B1	N_L	N_H	C3	N_8		11	20	15	24	

CMP obj, #N_{16}

16-bit Compare

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N'$_{16}$[X1], N'$_{16}$[X2]

Function

obj - #N_{16} (word long)

Description

This instruction compares the contents of the addressing object (word long) to the 16-bit immediate value (#N_{16}).

In the actual operation, the 16 bit immediate value is subtracted from the contents of the addressing object, and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF). The contents of the addressing object are unchanged.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	C0	N_L	N_H			8	17
DP	92	C0	N_L	N_H			8	—
X1	90	C0	N_L	N_H			8	—
X2	91	C0	N_L	N_H			8	—
USP	A1	C0	N_L	N_H			8	—
SSP	A0	C0	N_L	N_H			8	—
LRB	A4	C0	N_L	N_H			8	—
off N_8	B4	N_8	C0	N_L	N_H		10	20
N_8	B5	N_8	C0	N_L	N_H		10	—
[DP]	B2	C0	N_L	N_H			10	19
±N_8[USP]	B3	N_8	C0	N_L	N_H		11	20
N'$_{16}$[X1]	B0	N'$_L$	N'$_H$	C0	N_L	N_H	12	21
N'$_{16}$[X2]	B1	N'$_L$	N'$_H$	C0	N_L	N_H	12	21

CMPB A, obj

8-bit Compare

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A_L - obj (byte long)

Description

This instruction compares the contents of the lower byte of the accumulator (A_L) to the addressing object (byte long).

In the actual operation, the contents of the addressing object are subtracted from the contents of the accumulator lower byte. The result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF). The contents of the accumulator are unchanged.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•		

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8	C6	N_8					4	—
rN	48+N						3	5
PSWH	A2	C2					4	—
PSWL	A3	C2					4	—
off N_8	C7	N_8					4	7
N_8	C5	N_8	C2				6	—
[DP]	C2	C2					6	—
±N_8[USP]	C3	N_8	C2				7	9
N_{16}[X1]	C0	N_L	N_H	C2			8	10
N_{16}[X2]	C1	N_L	N_H	C2			8	10

CMPB obj, A

8-bit Compare

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj - A_L (byte long)

Description

This instruction compares the contents (byte long) of the addressing object to the lower byte of the accumulator (A_L).

In the actual operation, the contents of the lower byte of the accumulator are subtracted from the contents of the addressing object, and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF). The contents of the addressing object are unchanged.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_8								
rN	20+N	C1					5	9
PSWH	A2	C1					5	—
PSWL	A3	C1					5	—
off N_8	C4	N_8	C1				7	12
N_8	C5	N_8	C1				7	—
[DP]	C2	C1					7	11
$\pm N_8$[USP]	C3	N_8	C1				8	12
N_{16}[X1]	C0	N_L	N_H	C1			9	13
N_{16}[X2]	C1	N_L	N_H	C1			9	13

CMPB obj, off N_8

8-bit Compare

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj - off N_8 (byte long)

Description

This instruction compares the contents of the addressing object to the data memory specified by the current page addressing (off N_8) .

In the actual operation, the contents of the data memory specified by current page addressing are subtracted from the contents of the addressing object, and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT ← obj	INT ← obj	EXT ← obj	EXT ← obj	INT ← off N_8	EXT ← off N_8
A												
#N_8												
rN	20+N	C3	N_8				7	—	—	14		
PSWH	A2	C3	N_8				7	12	—	—		
PSWL	A3	C3	N_8				7	12	—	—		
off N'_8	C4	N'_8	C3	N_8			9	—	—	17		
N'_8	C5	N'_8	C3	N_8			9	14	—	—		
[DP]	C2	C3	N_8				9	14	11	16		
$\pm N'_8$[USP]	C3	N'_8	C3	N_8			10	15	12	17		
N_{16}[X1]	C0	N_L	N_H	C3	N_8		11	16	13	18		
N_{16}[X2]	C1	N_L	N_H	C3	N_8		11	16	13	18		

CMPB obj, off N_8

8-bit Compare

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj - off N_8 (byte long)

Description

This instruction compares the contents of the addressing object to the data memory specified by the current page addressing (off N_8) .

In the actual operation, the contents of the data memory specified by current page addressing are subtracted from the contents of the addressing object, and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT ← obj	INT ← obj	EXT ← obj	EXT ← obj	INT ← off N_8	EXT ← off N_8
A												
#N_8												
rN	20+N	C3	N_8				7	—	—	14		
PSWH	A2	C3	N_8				7	12	—	—		
PSWL	A3	C3	N_8				7	12	—	—		
off N'_8	C4	N'_8	C3	N_8			9	—	—	17		
N'_8	C5	N'_8	C3	N_8			9	14	—	—		
[DP]	C2	C3	N_8				9	14	11	16		
$\pm N'_8$[USP]	C3	N'_8	C3	N_8			10	15	12	17		
N_{16}[X1]	C0	N_L	N_H	C3	N_8		11	16	13	18		
N_{16}[X2]	C1	N_L	N_H	C3	N_8		11	16	13	18		

CMPB obj, #N_8

8-bit Compare

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj - #N_8 (byte long)

Description

This instruction compares the contents of the addressing object (byte long) to the 8-bit immediate value (#N_8).

In the actual operation, the 8-bit immediate value is subtracted from the contents of the addressing object, and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF). The contents of the addressing object are unchanged.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_8								
rN	20+N	C0	N_8				6	11
PSWH	A2	C0	N_8				6	—
PSWL	A3	C0	N_8				6	—
off N'_8	C4	N'_8	C0	N_8			8	14
N'_8	C5	N'_8	C0	N_8			8	—
[DP]	C2	C0	N_8				8	13
$\pm N'_8$[USP]	C3	N'_8	C0	N_8			9	14
N_{16}[X1]	C0	N_L	N_H	C0	N_8		10	15
N_{16}[X2]	C1	N_L	N_H	C0	N_8		10	15

CMPC A, N*16

16-bit ROM Compare (direct)

obj

Function

A - N*16 (word long)

Description

This instruction compares the contents (N*16) of the ROM[1] specified by direct addressing to the accumulator in a word long operation.

In the actual operation, the contents of the addressing object are subtracted from the contents of the accumulator, and the result of the subtraction determines the status of the zero flag (ZF) and the carry flag (CF). The contents of the accumulator are unchanged.

[NOTE] [1] The ROM used means the program memory in the program space. The on-chip memory is always the ROM; however, the external memory, if used, may possibly include ROM, RAM, and the I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	90	9E	N*L	N*H			15	15

CMPC A, obj

16-bit ROM Compare (indirect)

obj

[erN], [DP], [X1], [X2], [USP], [SSP], [LRB], [off N_8], [N_8], [[DP]], [$\pm N_8$[USP]], [N_{16}[X1]], [N_{16}[X2]]

Function

A - obj (word long)

Description

This instruction compares the contents of the addressing object (word long) in the ROM[1] space to the accumulator.

In the actual operation, the contents of the addressing object are subtracted from the contents of the accumulator, and the result of the subtraction sets the Zero (ZF) and Carry (CF) flags at 0 or 1. The contents of the accumulator are unchanged.

[NOTE] [1] The ROM used means the program memory in the program space. The on-chip memory is always the ROM; however, the external memory, if used, may possibly include ROM, RAM, and the I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	INT EXT	EXT INT	EXT EXT	← ROM
							INT	EXT	INT	EXT	← RAM
[erN]	44+N	AC					11	15	11	15	
[DP]	92	AC					11	—	11	—	
[X1]	90	AC					11	—	11	—	
[X2]	91	AC					11	—	11	—	
[USP]	A1	AC					11	—	11	—	
[SSP]	A0	AC					11	—	11	—	
[LRB]	A4	AC					11	—	11	—	
[off N_8]	B4	N_8	AC				13	18	11	18	
[N_8]	B5	N_8	AC				13	—	13	—	
[[DP]]	B2	AC					13	17	13	17	
[$\pm N_8$[USP]]	B3	N_8	AC				14	18	14	18	
[N_{16}[X1]]	B0	N_L	N_H	AC			15	19	15	19	
[N_{16}[X2]]	B1	N_L	N_H	AC			15	19	15	19	

CMPC A, obj — 16-bit ROM Compare (indirect with 16-bit base)

obj

N^*_{16}[DP], N^*_{16}[X1], N^*_{16}[X2], N^*_{16}[USP], N^*_{16}[off N_8], N^*_{16}[N_8]

Function

A - obj (word long)

Description

This instruction compares the contents of the addressing object in the ROM[1] space (word long) to the accumulator.

In the actual operation, the contents of the addressing object are subtracted from the contents of the accumulator, and the result of the subtraction sets the zero flag (ZF) and the carry flag (CF) at 0 or 1. The contents of the accumulator are unchanged.

[NOTE] [1] The ROM used means the program memory in the program space. The on-chip memory is always the ROM; however, the external memory, if used, may possibly include ROM, RAM, and the I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES				
	BYTE						INT	INT	EXT	EXT	← ROM
	1	2	3	4	5	6	INT	EXT	INT	EXT	← RAM
N^*_{16}[DP]	92	AD	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[X1]	90	AD	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[X2]	91	AD	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[USP]	A1	AD	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[off N_8]	B4	N_8	AD	N^*_L	N^*_H		17	22	17	22	
N^*_{16}[N_8]	B5	N_8	AD	N^*_L	N^*_H		17	—	17	—	

CMPCB A, N*16

8-bit ROM Compare (direct)

obj

Function

A_L - N^*_{16} (byte long)

Description

This instruction compares the contents (N^*_{16}) of the ROM[1] specified by direct addressing to the lower byte of the accumulator (A_L) in a byte long operation.

In the actual operation, the contents of the addressing object are subtracted from the contents of the accumulator, and the result of the subtraction sets the zero flag (ZF) and the carry flag (CF) at 0 or 1. The contents of the accumulator are unchanged.

[NOTE] [1] The ROM used means the program memory in the program space. The on-chip memory is always the ROM; however, the external memory, if used, may possibly include ROM, RAM, and the I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	90	9F	N^*_L	N^*_H			13	13

CMPCB A, obj

8-bit ROM Compare (indirect)

obj

[erN], [DP], [X1], [X2], [USP], [SSP], [LRB], [off N_8], [N_8], [[DP]], [$\pm N_8$[USP]], [N_{16}[X1]], [N_{16}[X2]]

Function

A_L - obj (byte long)

Description

This instruction compares the contents of the addressing object in the ROM[1] space (byte long) to the lower byte of the accumulator (A_L).

In the actual operation, the contents of the addressing object are subtracted, in a byte long operation, from the contents of the accumulator, and the result of the subtraction sets the zero flag (ZF) and the carry flag (CF) at 0 or 1. The contents of the accumulator are unchanged.

[NOTE] [1] The ROM used means the program memory in the program space. The on-chip memory is always the ROM; however, the external memory, if used, may possibly include ROM, RAM, and the I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES				
	BYTE						INT	INT	EXT	EXT	← ROM
	1	2	3	4	5	6	INT	EXT	INT	EXT	← RAM
[erN]	44+N	AE					9	11	9	11	
[DP]	92	AE					9	—	9	—	
[X1]	90	AE					9	—	9	—	
[X2]	91	AE					9	—	9	—	
[USP]	A1	AE					9	—	9	—	
[SSP]	A0	AE					9	—	9	—	
[LRB]	A4	AE					9	—	9	—	
[off N_8]	B4	N_8	AE				11	16	11	16	
[N_8]	B5	N_8	AE				11	—	11	—	
[[DP]]	B2	AE					11	15	11	15	
[$\pm N_8$[USP]]	B3	N_8	AE				12	16	12	16	
[N_{16}[X1]]	B0	N_L	N_H	AE			13	17	13	17	
[N_{16}[X2]]	B1	N_L	N_H	AE			13	17	13	17	

CMPCB A, obj — 8-bit ROM Compare (indirect with 16-bit base)

obj

N^*_{16}[DP], N^*_{16}[X1], N^*_{16}[X2], N^*_{16}[USP], N^*_{16}[off N_8], N^*_{16}[N_8]

Function

A_L - obj (byte long)

Description

This instruction compares the contents (byte long) of the addressing object in the ROM[1] space and the contents of the lower byte of the accumulator (A_L).

In the actual operation, the contents of the addressing object are subtracted, in a byte long operation, from the contents of the lower byte of the accumulator, and the result of the subtraction sets the zero flag (ZF) and the carry flag (CF) at 0 or 1. The contents of the accumulator are unchanged.

[NOTE] [1] The ROM used means the program memory in the program space. The on-chip memory is always the ROM; however, the external memory, if used, may possibly include ROM, RAM, and the I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES				
	BYTE						INT	INT	EXT	EXT	← ROM
	1	2	3	4	5	6	INT	EXT	INT	EXT	← RAM
N^*_{16}[DP]	92	AF	N^*_L	N^*_H			13	—	13	—	
N^*_{16}[X1]	90	AF	N^*_L	N^*_H			13	—	13	—	
N^*_{16}[X2]	91	AF	N^*_L	N^*_H			13	—	13	—	
N^*_{16}[USP]	A1	AF	N^*_L	N^*_H			13	—	13	—	
N^*_{16}[off N_8]	B4	N_8	AF	N^*_L	N^*_H		17	22	17	22	
N^*_{16}[N_8]	B5	N_8	AF	N^*_L	N^*_H		17	—	17	—	

DAA

8-bit Decimal Adjustment for Addition

obj

Function

IF $(A_{L3-0} \geq 10)$ OR $(HC = 1)$ THEN $A_L \leftarrow A_L + 6$

IF $(A_{L7-4} \geq 10)$ OR $(C=1)$ THEN $A_{L7-4} \leftarrow A_{L7-4} + 6$

Description

This instruction adjusts the 8-bit value in the lower byte of the accumulator (A_L) resulting from an earlier binary addition of 2-byte long BCD codes, producing packed BCD codes. Following the execution of the adjusting instruction, when a carry from the lower nibble to the upper nibble occurs, the half carry (HC) is set to 1. Furthermore, when a carry occurs from a digit to the adjacent higher digit, the carry (C) is set to 1; such carries come from the upper nibble during the decimal adjusting manipulation or from the execution of the add instruction prior to the decimal adjustment.

It should be noted that this instruction is based on the following three premises. Two (byte long) BCD codes are added by using one of the byte long add instructions (ADDB, ADCB). The result of the addition is stored in the lower byte of the accumulator. The contents of the half carry and the carry maintain the status which existed at the end of the adding operation until the following adjustment operation revises the status. Below the actual process encountered in this (adjustment) instruction is described.

1. At first, the lower nibble (A_{L3-0}) of the accumulator is adjusted by adding 6 if either the lower nibble of the accumulator is more than 9 ($0A_H$—$0F_H$), or the half carry (HC) is equal to 1. This adding operation is a byte long operation, and when the addition causes an overflow from bit 3 to bit 4 of the accumulator, the half carry is set to 1 while the overflow from bit 7 sets the carry to 1.

   When the lower nibble of the accumulator does not exceed 9 and also the half carry (HC), prior to the execution of the adjust instruction, is 0, then no adjustment operation is applied to the lower nibble.

2. Secondly, the adjustment of the upper nibble (A_{L7-4}) of the accumulator begins by adding 6 to the upper nibble if either the upper nibble of the accumulator is more than 9 ($0A_H$ through $0F_H$), or the carry is 1, which may have existed either before or after the adding operation of step 1. The status of the half carry at this moment remains unaltered from the status which existed during the process of step 1. The overflow from bit 7, if it exists, sets the carry to 1.

   When the upper nibble of the accumulator does not exceed 9 and also the carry is 0 before and after the operation of step 1, then no adjustment operation is applied to the upper nibble.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*	*	

Flags affecting execution:

DD

Codes/Cycles

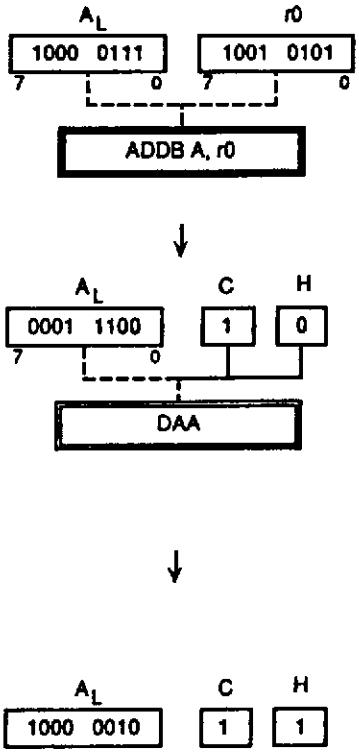
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	93						6	—

Examples

The execution process for this instruction is described below, where A_L and r0 are the BCD codes.

```
ADDB A, r0
DAA
```

[Example 1] 87 + 95

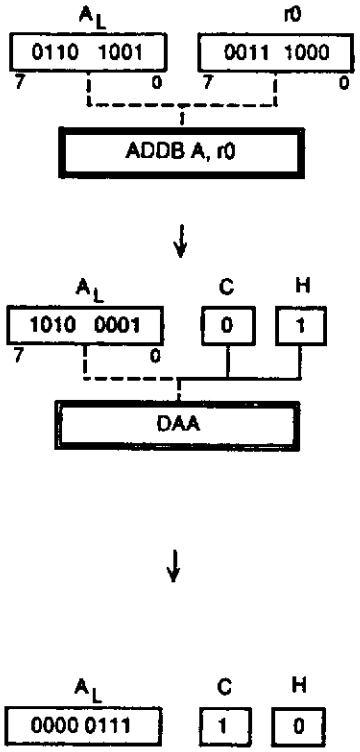


The BCD codes for decimal numbers 87 and 95 are respectively A_L and r0, and they are added using ADDB A, r0.

The results of the addition are that $1C_H$ is stored in A_L, and C is set to 1 because of the overflow from the bit 7.
There is no overflow from bit 3 and H is set to 0.

Decimal adjustment is performed by the DAA instruction.
At first, the lower nibble of A_L is more than 9, and therefore 6 is added to the lower nibble. As a result, A_L becomes 22_H, and the carry from bit 3 sets H to 1.
C is 1 prior to the execution of the instruction; therefore, 6 is added to the upper nibble and A_L becomes 82_H, and C remains at 1 (as it was at the beginning) following the execution of the instruction.
Finally, the BCD code for the decimal number 82 remains in A_L, and the carry to the next higher digit is set in C.

[Example 2] 69 + 38



A_L and r0 represent the BCD codes for decimal numbers 69 and 38 respectively which are added by the ADDB A, r0 instruction.

As a result of the addition, $A1_H$ is stored in A_L , and C is set to 0 because of no overflow from the bit 7.
The carry from bit 3 sets the H to 0.

Decimal adjustment is performed by the DAA instruction.
At first, 6 is added to A_L because H=1. A_L becomes $A7_H$ and there is no carry from bit 3, hence H is set to 0.
The upper nibble of A_L is more than 9, therefore 6 is added to the upper nibble of A_L making A_L 07_H. C is set to 1 because of the overflow from the upper nibble adjustment. Finally, the contents of A_L show- the BCD code for the decimal number 07 and the carry to the next higher digit is indicated by C=1.

DAS

8-bit Decimal Adjustment for Subtraction

obj

Function

IF $(A_{L3-0} \geq 10)$ OR $(HC = 1)$ THEN $A_L \leftarrow A_L - 6$

IF $(A_{L7-4} \geq 10)$ OR $(C=1)$ THEN $A_{L7-4} \leftarrow A_{L7-4} - 6$
$HC \leftarrow 0$

Description

This instruction adjusts the 8-bit value in the lower byte of the accumulator (A_L) resulting from the earlier binary subtraction of 2-byte long BCD codes, producing packed BCD codes

It should be noted that this instruction is based on the following three premises. Two (byte long) BCD code are subtracted by using one of the byte long subtract instruction (SUBB, SUCB). The result of the subtraction is stored in the lower byte of the accumulator. The contents of the half carry and the carry maintains the status which existed at the end of the subtracting operation until the following adjustment operation revises the status.

The following is the actual process encountered in this (adjustment) instruction.

1. At first, the lower nibble (A_{L3-0}) of the accumulator is augmented by subtracting 6 from the lower nibble of the accumulator if the half carry (HC) is equal to 1. When the half carry (HC), prior to the execution of the augment instruction, is 0, then no adjustment operation is applied to the lower nibble.

2. Secondly, the adjustment of the upper nibble (A_{L7-4}) of the accumulator begins by subtracting 6 from the upper nibble of the accumulator if the carry (C) is 1. When the carry (C) is 0, then no adjustment operation is applied to the lower nibble.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*	*	

Flags affecting execution:

DD

Codes/Cycles

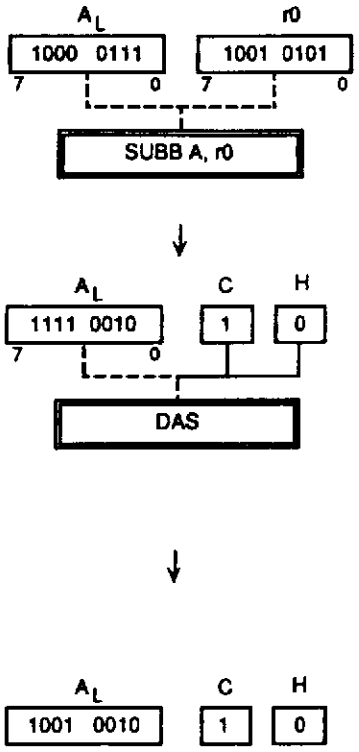
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	94						6	—

Examples

The execution process of this instruction is shown using the following case in which A_L and r0 are the BCD codes.

```
SUBB A, r0 (or SBCB A, r0)
DAS
```

[Example 1] 87 - 95

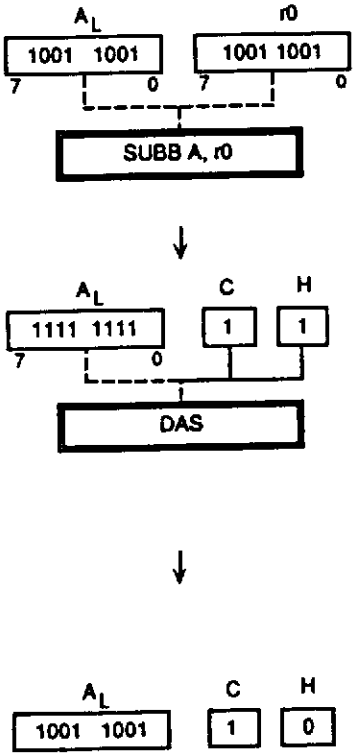


The BCD codes for decimal numbers 87 and 95 are respectively A_L and r0 and they are subtracted using SUBB A, r0.

As a result of the subtraction, F2 is stored in A_L and C is set to 1 because of the borrow from bit 7.
There is no borrow from bit 3 and H is set to 0.

Decimal adjustment is performed by the DAS instruction. Initially, since H=0, the lower nibble is not adjusted. Since C=1, 6 is subtracted from the upper nibble of A_L, and A_L becomes 92_H.
C maintains the status that exists prior to the execution of the augment instruction, namely C=1.
Finally, the BCD code for the decimal number 92 stays in A_L and the borrow from the next higher digit is indicated by C=1.

[Example 2] 99 - 99 with the borrow from the preceding step.



Both A_L and r0 are the BCD code for the decimal number 99 and the C is set to 1 indicating a borrow has occurred in the preceding step. Now the subtract instruction, SBCB A, r0, is executed.

The result of the subtraction, FF_H is stored in A_L. C is set to 1 because of the borrow from the bit 7.
H is set to 1 because of the borrow from the bit 3.

The decimal adjustment is performed by the DAS instruction.
At first, since H=1, 6 is subtracted from the lower nibble of A_L.
Since C=1, 6 is subtracted from the upper nibble of A_L and A_L becomes 99H.
The C has been 1 prior to the execution of the adjustment instruction, and remains 1.
Finally, the BCD code for the decimal number 99 stays in A_L and the borrow from the next higher digit is indicated by C=1.

DEC obj

16-bit Decrement

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + 1 (word long)

Description

This instruction subtracts 1 from the contents of the addressing object (word long).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*		*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	17					5	13
DP	82						3	—
X1	80						3	—
X2	81						3	—
USP	A1	17					5	—
SSP	A0	17					5	—
LRB	FE						3	—
off N_8	B4	N_8	17				7	16
N_8	B5	N_8	17				7	—
[DP]	B2	17					7	15
$\pm N_8$[USP]	B3	N_8	17				8	16
N_{16}[X1]	B0	N_L	N_H	17			9	17
N_{16}[X2]	B1	N_L	N_H	17			9	17

DECB obj

8-bit Decrement

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + 1 (byte long)

Description

This instruction subtracts 1 from the contents of the addressing object (byte long).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*		*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	B8+N						3	7
PSWH	A2	17					5	—
PSWL	A3	17					5	—
off N_8	C4	N_8	17				7	12
N_8	C5	N_8	17				7	—
[DP]	C2	17					7	11
$\pm N_8$[USP]	C3	N_8	17				8	12
N_{16}[X1]	C0	N_L	N_H	17			9	13
N_{16}[X2]	C1	N_L	N_H	17			9	13

DIV

16-bit Divide

obj

Function

(er0, A) ← (er0, A) + er2
er1 ← (er0, A) MOD er2

Description

32 bits divided by 16 bits →32 bits divide instruction

The 32-bit value (dividend), whose upper and lower 16 bits are given by the extended local register 0 (er0) and the accumulator respectively, is divided by the contents of the extended local register 2 (er2).

The results of the division are stored in three different locations. The upper 16 bits of the quotient is stored in the extended local register 0 (er0), the lower 16 bits of the quotient is stored in the accumulator, and the remainder (16bits) is stored in the extended local register 1 (er1).

[NOTE] When division by 0 is executed (i.e., er2 = 0), the quotient and the remainder become undetermined, and the carry flag is set to 1. In all other cases, the carry flag is set to 0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	90	37					47	63

DIVB — 8-bit Divide

obj

Function

A ← A ÷ r0
r1 ← A MOD r0

Description

16 bits divided by 8 bits →16 bits Divide Instruction

The contents (16 bits) of the accumulator are divided by the contents (8 bits) of the local register 0 (r0), when this instruction is executed.

The result of the division are stored in two locations. The quotient (16 bits) is stored in the accumulator and the remainder (8 bits) is stored in the local register 1 (r1).

[NOTE] When the division by 0 is executed (i.e., er2 = 0), the quotient and the remainder become undetermined, and the carry flag is set to 1. In all other cases, the carry flag is set to 0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	A2	36					29	33

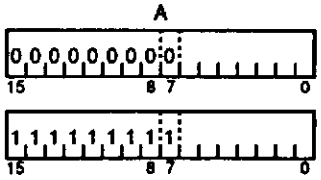
EXTND

Signed Byte to Word Extend

obj

Function

$A_{15-8} \leftarrow A_7$
DD $\leftarrow$ 1



Description

This instruction extends signed byte type data to signed word type data. The upper byte of the accumulator is filled with the highest bit (A_7), which is the sign bit of the byte type data. *At the same time, the data descriptor is set to 1 to indicate a word long operation.*

Flags

Flags affected by execution:

ZF	CF	HC	DD
			1

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	F8						2	—

INC obj

16-bit Increment

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + 1 (word long)

Description

This instruction adds 1 to the contents of the addressing object (word long).
If LRB is the addressing object in any addressing mode, ZF will not be affected.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•		•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	16					5	13
DP	72						3	—
X1	70						3	—
X2	71						3	—
USP	A1	16					5	—
SSP	A0	16					5	—
LRB	FD						3	—
off N_8	B4	N_8	16				7	16
N_8	B5	N_8	16				7	—
[DP]	B2	16					7	15
$\pm N_8$[USP]	B3	N_8	16				8	16
N_{16}[X1]	B0	N_L	N_H	16			9	17
N_{16}[X2]	B1	N_L	N_H	16			9	17

INC obj

16-bit Increment

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + 1 (word long)

Description

This instruction adds 1 to the contents of the addressing object (word long).
If LRB is the addressing object in any addressing mode, ZF will not be affected.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•		•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	16					5	13
DP	72						3	—
X1	70						3	—
X2	71						3	—
USP	A1	16					5	—
SSP	A0	16					5	—
LRB	FD						3	—
off N_8	B4	N_8	16				7	16
N_8	B5	N_8	16				7	—
[DP]	B2	16					7	15
$\pm N_8$[USP]	B3	N_8	16				8	16
N_{16}[X1]	B0	N_L	N_H	16			9	17
N_{16}[X2]	B1	N_L	N_H	16			9	17

INCB obj

8-bit Increment

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj + 1 (byte long)

Description

This instruction adds 1 to the contents of the addressing object (byte long).
If LRB's upper bits are the addressing object in any addressing mode, ZF will not be affected.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•		•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	A8+N						3	7
PSWH	A2	16					5	—
PSWL	A3	16					5	—
off N_8	C4	N_8	16				7	12
N_8	C5	N_8	16				7	—
[DP]	C2	16					7	11
±N_8[USP]	C3	N_8	16				8	12
N_{16}[X1]	C0	N_L	N_H	16			9	13
N_{16}[X2]	C1	N_L	N_H	16			9	13

J address

16-bit Space (64K byte) Direct Jump

obj

Function

PC ← address (16 bits)

Description

This instruction performs a direct jump to the program (memory) space of 64K bytes. The address is specified by the 16 bit address value (0000_H through $FFFF_H$) that is the destination address of the jump.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	03	adr_L	adr_H				7	—

[NOTE] adr_L and adr_H are respectively the lower 8 bits and the upper 8 bits of the address.

J obj

16-bit Space (64K byte) Direct Jump

obj

[erN], [DP], [X1], [X2], [USP], [SSP], [LRB], [off N_8], [N_8], [[DP]], [±N_8[USP]], [N_{16}[X1]], [N_{16}[X2]]

Function

PC ← obj (16 bits)

Description

This instruction performs an indirect jump to the 16 bit (64K byte) space where the destination is specified by the contents of the addressing object.

It should be noted that the obj in the preceding "Function" is expressed by enclosing the contents of the RAM addressing expression (word long) with brackets ([ ]), and the value is the 16-bit contents of the RAM which is the object of the addressing. In other words, the destination address of the jump is given by the 16-bit contents of the RAM which is addressed in the instruction.

[NOTE] This instruction may be easier to understand if it is thought of as a "MOV PC, obj" instruction with the brackets ([ ]) of the obj removed.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
[erN]	44+N	22					6	10
[DP]	92	22					6	—
[X1]	90	22					6	—
[X2]	91	22					6	—
[USP]	A1	22					6	—
[SSP]	A0	22					6	—
[LRB]	A4	22					6	—
[off N_8]	B4	N_8	22				8	13
[N_8]	B5	N_8	22				8	—
[[DP]]	B2	22					8	12
[±N_8[USP]]	B3	N_8	22				9	13
[N_{16}[X1]]	B0	N_L	N_H	22			10	14
[N_{16}[X2]]	B1	N_L	N_H	22			10	14

JBR off N_8.bit, address

Jump if Bit Reset

obj

Function

IF (off N_8.bit = 0) THEN PC ← address

where PC* - 128 ≤ address ≤ PC* + 127
PC* = starting address of the instruction following the JBR instruction
= (address of the first byte of the JBR instruction) + 3

Description

This instruction causes a jump if a specified bit of the data memory (off N_8) that is the object of direct page addressing has a value of 0.

The location of the specified bit is given by the "bit" (0 through 7) in the instruction. The destination address of the jump is given by the "address" in the instruction. The range of the jump is from -128 to +127 with the address of the instruction that follows next serving as the origin.

[NOTE] This jump instruction manipulates the PC based on the displacement. However, in the source program of the assembler (RAS66K), the destination address of the jump specified as a label and not as a displacement, is entered as the operand of the instruction. The assembler figures out the amount of the displacement, which, if it stays within the allowable range, is used to generate the machine codes. If the amount of the displacement is outside of the allowable range, the assembler issues an error warning.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES				
	BYTE						INT	INT	EXT	EXT	← obj
	1	2	3	4	5	6	INT	EXT	INT	EXT	← off N8
	D8+ n	N_8	N'_8				10	6	13	9	

[NOTE] 1. N'_8 is the 8-bit code to represent the displacement in the range from -128 to +127. Negative values are expressed as 2's complement.

[EXAMPLE]

Displacement	N'_8
0	00000000_B
+1	00000001_B
+127	01111111_B
-1	11111111_B
-127	10000001_B
-128	10000000_B

2. n is determined from the value of the "bit", and ranges from 0 to 7.

JBS off N_8.bit, address

Jump if Bit Set

obj

Function

IF (off N_8.bit = 1) THEN PC ← address

where PC* - 128 ≤ address ≤ PC* + 127
PC* = starting address of the instruction following the JBS instruction
= (address of the first byte of the JBS instruction) + 3

Description

This instruction causes a jump if a specified bit of the data memory (off N_8) that is the object of direct page addressing has a value of 1.

The location of the specified bit is given by the "bit" (0 through 7) in the instruction. The destination address of the jump is given by the "address" in the instruction. The range of the jump is from -128 to +127 with the address of the instruction that follows next serving as the origin.

[NOTE] This jump instruction manipulates the PC based on the displacement. However, in the source program of the assembler (RAS66K), the destination address of the jump specified as a label and not as a displacement, is entered as the operand of the instruction. The assembler figures out the amount of the displacement, which, if it stays within the allowable range, is used to generate the machine codes. If the amount of the displacement is outside of the allowable range, the assembler issues an error warning.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES				
	BYTE						INT	iNT	EXT	EXT	←off N_8
	1	2	3	4	5	6	INT	EXT	INT	EXT	← off N_8.bit
	E8+n	N_8	N'_8				10	6	13	9	

[NOTE] 1. N'_8 is the 8-bit code to represent the displacement in the range from -128 to +127. Negative values are expressed as 2's complement.

[EXAMPLE]

Displacement	N'_8
0	00000000_B
+1	00000001_B
+127	01111111_B
-1	11111111_B
-127	10000001_B
-128	10000000_B

2. n is determined from the value of the "bit", and ranges from 0 to 7.

JC condition, address

Conditional Jump

obj

Function

IF condition is true THEN PC ← address

where PC* - 128 ≤ address ≤ PC* + 127
PC* = starting address of the instruction following the JC instruction
= (address of the first byte of the JC instruction) + 2

Description

This instruction causes a jump to the address specified by the second operand "address" in the instruction, if the first operand condition" is true.

The range of the jump is from -128 to +127 with the origin at the address of the instruction that follows next.

The truth of the conditions is defined by the zero flag (ZF) and/or the carry flag (CF) as described in the following table, which also list the symbols for the conditions. The conditions and the terms that must be fulfilled for the conditions to be true are shown in the following table.

Symbols for Conditions	Terms to be fulfilled for condition to be true				
	Flag			Logical meaning	
EQ	Z = 1			=	Equal
NE	Z = 0			≠	Not Equal
LT			C = 1	<	Less Than
LE	Z = 1	or	C = 1	≤	Less or Equal
GT	Z = 0	or	C = 0	>	Greater Than
GE			C = 0	≥	Greater or Equal

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

CONDITION	CODE						CYCLES	
	BYTE						TRUE	FALSE
	1	2	3	4	5	6	(jump)	
EQ	C9	N_8					8	4
NE	CE	N_8					8	4
LT	CA	N_8					8	4
LE	CF	N_8					8	4
GT	C8	N_8					8	4
GE	CD	N_8					8	4

[NOTE] N_8 is the code for the displacement. See the following section, "Displacement."

Displacement

N_8

In the above table, N_8 is the 8-bit code to represent the displacement in the range from -128 to +127. Negative values are expressed as 2's complement.

[EXAMPLE]

Displacement	N_8
0	00000000_B
+1	00000001_B
+127	01111111_B
-1	11111111_B
-127	10000001_B
-128	10000000_B

Caution for entering address

This jump instruction manipulates the PC based on the displacement; however, in the source program of the assembler (RAS66K), the destination address of the jump specified as a label and not as a displacement, is entered as the operand of the instruction. The assembler figures out the amount of the displacement, which, if it stays within the allowable range, is used to generate the machine codes. If the amount of the displacement is outside of the range, the assembler issues an error warning.

JRNZ DP, address

Loop

obj

Function

$DP_L \leftarrow DP_L - 1$
IF $DP_L \neq 0$ THEN PC $\leftarrow$ address

where $PC^* - 128 \leq$ address $\leq PC^* + 127$
PC^* = starting address of the instruction following the JRNZ instruction
= (address of the first byte of the JRNZ instruction) + 2

Description

This instruction performs a jump to the address specified by the second operand "address" in the instruction if the content of the lower byte of the data pointer (DP_L) decremented by 1 is not 0. This instruction permits implementing a simple loop process for counting the lower byte of the data pointer. The range of the jump is from -128 to +127 with the address of the instruction that follows next serving as the origin.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE						$DP_L \neq 0$	$DP_L = 0$
	1	2	3	4	5	6	(jump)	(no jump)
	30	N_8					11	7

[NOTE] N_8 is the 8-bit code to represent the displacement in the range from -128 to +127. Negative values are expressed as 2's complement.

[EXAMPLE]

Displacement	N_8
0	00000000_8
+1	00000001_8
+127	01111111_8
-1	11111111_8
-127	10000001_8
-128	10000000_8

L A, obj

16-bit Load

obj

#N_{16}, erN, DP, X1, X2, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A ← obj (word long)
DD ← 1

Description

This instruction transfers the contents of the addressing object (word long) to the accumulator.

This instruction sets the data descriptor to 1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			1

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}	67	N_L	N_H				6	—
erN	34+N						2	6
DP	42						2	—
X1	40						2	—
X2	41						2	—
USP								
SSP								
LRB								
off N_8	E4	N_8					4	9
N_8	E5	N_8					4	—
[DP]	E2						4	8
±N_8[USP]	E3	N_8					5	9
N_{16}[X1]	E0	N_L					6	10
N_{16}[X2]	E1	N_L					6	10

LB A, obj

8-bit Load

obj

#N_8, rN, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A_L ← obj (byte long)
DD ← 0

Description

This instruction transfers the contents of the addressing object (byte long) to the lower byte (A_L) of the accumulator.

This instruction sets the data descriptor (DD) to 0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			0

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8	77	N_8					4	—
rN	78+N						2	4
PSWH								
PSWL								
off N_8	F4	N_8					4	7
N_8	F5	N_8					4	—
[DP]	F2						4	6
±N_8[USP]	F3	N_8					5	7
N_{16}[X1]	F0	N_L	N_H				6	8
N_{16}[X2]	F1	N_L	N_H				6	8

LC A, N'16

16-bit Load ROM Reference (direct)

obj

Function

A ← N'_{16} (word long)

Description

This instruction transfers the contents (N'_{16}) of the ROM[1] specified by direct addressing to the accumulator in word long operation.

[NOTE][1] ROM as used here means the program memory in the program space. The on-chip memory is always considered ROM; however, the external memory, if used, may possibly include ROM, RAM, and I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	90	9C	N'_L	N'_H			15	15

LC A, obj

16-bit Load ROM Reference (indirect)

obj

[erN], [DP], [X1], [X2], [USP], [SSP], [LRB], [off N_8], [N_8], [[DP]], [±N_8[USP]], [N_{16}[X1]], [N_{16}[X2]]

Function

A ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) in the ROM[1] space to the accumulator.

[NOTE][1] ROM as used here means the program memory in the program space. The on-chip memory is always considered ROM; however, the external memory, if used, may possibly include ROM, RAM, and I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	INT EXT	EXT INT	EXT EXT	← ROM / ← RAM
[erN]	44+N	A8					11	15	11	15	
[DP]	92	A8					11	—	11	—	
[X1]	90	A8					11	—	11	—	
[X2]	91	A8					11	—	11	—	
[USP]	A1	A8					11	—	11	—	
[SSP]	A0	A8					11	—	11	—	
[LRB]	A4	A8					11	—	11	—	
[off N_8]	B4	N_8	A8				13	18	13	18	
[N_8]	B5	N_8	A8				13	—	13	—	
[[DP]]	B2	A8					13	17	13	17	
[±N_8[USP]]	B3	N_8	A8				14	18	14	18	
[N_{16}[X1]]	B0	N_L	N_H	A8			15	19	15	19	
[N_{16}[X2]]	B1	N_L	N_H	A8			15	19	15	19	

LC A, obj

16-bit Load ROM Reference (indirect with 16-bit base)

obj

N^*_{16}[DP], N^*_{16}[X1], N^*_{16}[X2], N^*_{16}[USP], N^*_{16}[off N_8], N^*_{16}[N_8]

Function

A ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) in the ROM[1] space to the accumulator.

[NOTE][1] ROM as used here means the program memory in the program space. The on-chip memory is always considered ROM; however, the external memory, if used, may possibly include ROM, RAM, and I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES				
							INT	INT	EXT	EXT	← ROM
	1	2	3	4	5	6	INT	EXT	INT	EXT	← RAM
N^*_{16}[DP]	92	A9	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[X1]	90	A9	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[X2]	91	A9	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[USP]	A1	A9	N^*_L	N^*_H			15	—	15	—	
N^*_{16}[off N_8]	B4	N_8	A9	N^*_L	N^*_H		17	22	17	22	
N^*_{16}[N_8]	B5	N_8	A9	N^*_L	N^*_H		17	—	17	—	

LCB A, N*16

8-bit Load ROM Reference (direct)

obj

Function

$A_L \leftarrow N^*_{16}$ (byte long)

Description

This instruction transfers the contents (N^*_{16}) of the ROM[1] specified by direct addressing to the lower byte (A_L) of the accumulator in a byte long operation.

[NOTE][1] ROM as used here means the program memory in the program space. The on-chip memory is always considered ROM; however, the external memory, if used, may possibly include ROM, RAM, and I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	90	9D	N^*_L	N^*_H			13	13

LCB A, obj

8-bit Load ROM Reference (indirect)

obj

[erN], [DP], [X1], [X2], [off N_8], [N_8], [[DP]], [$\pm N_8$[USP]], [N_{16}[X1]], [N_{16}[X2]]

Function

$A_L \leftarrow$ obj (byte long)

Description

This instruction transfers the contents of the addressing object (byte long) in the ROM[1] space to the lower byte (A_L) of the accumulator.

[NOTE][1] ROM as used here means the program memory in the program space. The on-chip memory is always considered ROM; however, the external memory, if used, may possibly include ROM, RAM, and I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES INT	CYCLES INT	CYCLES EXT	CYCLES EXT	← ROM
	1	2	3	4	5	6	INT	EXT	INT	EXT	← RAM
[erN]	44+N	AA					9	11	9	11	
[DP]	92	AA					9	—	9	—	
[X1]	90	AA					9	—	9	—	
[X2]	91	AA					9	—	9	—	
[USP]	A1	AA					9	—	9	—	
[SSP]	A0	AA					9	—	9	—	
[LRB]	A4	AA					9	—	9	—	
[off N_8]	B4	N_8	AA				11	16	11	16	
[N_8]	B5	N_8	AA				11	—	11	—	
[[DP]]	B2	AA					11	15	11	15	
[$\pm N_8$[USP]]	B3	N_8	AA				12	16	12	16	
[N_{16}[X1]]	B0	N_L	N_H	AA			13	17	13	17	
[N_{16}[X2]]	B1	N_L	N_H	AA			13	17	13	17	

LCB A, obj

8-bit Load ROM Reference (indirect with 16-bit base)

obj

N^*_{16}[DP], N^*_{16}[X1], N^*_{16}[X2], N^*_{16}[USP], N^*_{16}[off N_8], N^*_{16}[N_8]

Function

$A_L \leftarrow$ obj (byte long)

Description

This instruction transfers the contents of the addressing object (byte long) in the ROM[1] space to the lower byte (A_L) of the accumulator.

[NOTE][1] ROM as used here means the program memory in the program space. The on-chip memory is always considered ROM; however, the external memory, if used, may possibly include ROM, RAM, and I/O units allocated to the program space.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES ← ROM			
							INT	INT	EXT	EXT
	1	2	3	4	5	6	INT	EXT	INT	EXT ← RAM
N^*_{16}[DP]	92	AB	N^*_L	N^*_H			13	—	13	—
N^*_{16}[X1]	90	AB	N^*_L	N^*_H			13	—	13	—
N^*_{16}[X2]	91	AB	N^*_L	N^*_H			13	—	13	—
N^*_{16}[USP]	A1	AB	N^*_L	N^*_H			13	—	13	—
N^*_{16}[off N_8]	B4	N_8	AB	N^*_L	N^*_H		17	22	17	22
N^*_{16}[N_8]	B5	N_8	AB	N^*_L	N^*_H		17	—	17	—

MB C, obj.bit

Move Bit (direct bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

C ← obj.bit

Description

This instruction transfers the contents of the specified bit of the operand "obj" (byte long) that is the addressing object in this instruction to the carry (C). The location of the bit is specified by the term "bit" whose value ranges from 0 to 7.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_8								
rN	20+N	28+n					5	7
PSWH	A2	28+n					5	—
PSWL	A3	28+n					5	—
off N_8	C4	N_8	28+n				7	10
N_8	C5	N_8	28+n				7	—
[DP]	C2	28+n					7	9
$\pm N_8$[USP]	C3	N_8	28+n				8	10
N_{16}[X1]	C0	N_L	N_H	28+n			9	11
N_{16}[X2]	C1	N_L	N_H	28+n			9	11

[NOTE] "n" is determined from the value of the "bit", and ranges from 0 to 7.

MB obj.bit, C

Move Bit (direct bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

obj.bit ← C

Description

This instruction transfers the contents of the carry (C) to the specified bit of the operand "obj" (byte long) that is the addressing object in this instruction. The location of the specified bit is given by the term "bit" in the instruction and its value ranges from 0 to 7.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	38+n					10	16
PSWH	A2	38+n					10	—
PSWL	A3	38+n					10	—
off N_8	C4	N_8	38+n				12	19
N_8	C5	N_8	38+n				12	—
[DP]	C2	38+n					12	18
±N_8[USP]	C3	N_8	38+n				13	19
N_{16}[X1]	C0	N_L	N_H	38+n			14	20
N_{16}[X2]	C1	N_L	N_H	38+n			14	20

[NOTE] "n" is determined from the value of the "bit", and ranges from 0 to 7.

MBR C, obj

Move Bit (register indirect bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

C ← obj.bit (A_{2-0})

Description

This instruction transfers the contents of the specified bit of the operand "obj" (byte long) that is the addressing object in this instruction to the carry (C). The location of the bit is specified at the execution time of the instruction by the contents of the lower 3 bits (A_{2-0}) of the accumulator.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	21					5	7
PSWH	A2	21					5	—
PSWL	A3	21					5	—
off N_8	C4	N_8	21				7	10
N_8	C5	N_8	21				7	—
[DP]	C2	21					7	9
$\pm N_8$[USP]	C3	N_8	21				8	10
N_{16}[X1]	C0	N_L	N_H	21			9	11
N_{16}[X2]	C1	N_L	N_H	21			9	11

MBR obj, C

Move Bit (register indirect bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj.bit (A_{2-0}) ← C

Description

This instruction transfers the contents of the carry (C) to the specified bit of the operand "obj" (byte long) that is the addressing object in this instruction. The location of the bit is specified at the execution time of the instruction by the contents of the lower 3 bits (A_{2-0}) of the accumulator.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	20					10	16
PSWH	A2	20					10	—
PSWL	A3	20					10	—
off N_8	C4	N_8	20				12	19
N_8	C5	N_8	20				12	—
[DP]	C2	20					12	18
$\pm N_8$[USP]	C3	N_8	20				13	19
N_{16}[X1]	C0	N_L	N_H	20			14	20
N_{16}[X2]	C1	N_L	N_H	20			14	20

MOV A, obj

16-bit Move

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

A ← obj (word long)
DD ← 1

Description

This instruction transfers the contents of the addressing object (word long) to the accumulator.

This instruction sets the data descriptor (DD) to 1.

Also, this instruction functionally overlaps part of the "L A, obj" instruction. This instruction differs from "L A, obj" instruction in that the execution of this instruction is not affected by the zero flag, and, also, this instruction incurs a comparatively large number of bytes and cycles.

Flags

Flags affected by execution:

ZF	CF	HC	DD
			1

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	99					4	8
DP	92	99					4	—
X1	90	99					4	—
X2	91	99					4	—
USP	A1	99					4	—
SSP	A0	99					4	—
LRB	A4	99					4	—
off N_8	B4	N_8	99				6	11
N_8	B5	N_8	99				6	—
[DP]	B2	99					6	10
$\pm N_8$[USP]	B3	N_8	99				7	11
N_{16}[X1]	B0	N_L	N_H	99			8	12
N_{16}[X2]	B1	N_L	N_H	99			8	12

MOV DP, obj

16-bit Move

obj

A, #N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

DP ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the data pointer.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	52						2	—
#N_{16}	62	N_L	N_H				6	—
erN	44+N	7A					4	8
DP	92	7A					4	—
X1	90	7A					4	—
X2	91	7A					4	—
USP	A1	7A					4	—
SSP	A0	7A					4	—
LRB	A4	7A					4	—
off N_8	B4	N_8	7A				6	11
N_8	B5	N_8	7A				6	—
[DP]	B2	7A					6	10
±N_8[USP]	B3	N_8	7A				7	11
N_{16}[X1]	B0	N_L	N_H	7A			8	12
N_{16}[X2]	B1	N_L	N_H	7A			8	12

MOV erN, obj

16-bit Move

obj

A, #N_{16}, erN', DP, X1, X2,USP,SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

erN ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the extended local register (erN).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES INT	CYCLES INT	CYCLES EXT	CYCLES EXT	← erN
	1	2	3	4	5	6	INT	EXT	INT	EXT	← obj
A	44+N	8A					4	—	—	12	
#N_{16}	44+N	98	N_L	N_H			8	—	—	16	
erN'	44+N'	48+N					4	—	—	12	
DP	92	48+N					4	—	8	—	
X1	90	48+N					4	—	8	—	
X2	91	48+N					4	—	8	—	
USP	A1	48+N					4	—	8	—	
SSP	A0	48+N					4	—	8	—	
LRB	A4	48+N					4	—	8	—	
off N_8	B4	N_8	48+N				6	—	—	15	
N_8	B5	N_8	48+N				6	—	10	—	
[DP]	B2	48+N					6	10	10	14	
±N_8[USP]	B3	N_8	48+N				7	11	11	15	
N_{16}[X1]	B0	N_L	N_H	48+N			8	12	12	16	
N_{16}[X2]	B1	N_L	N_H	48+N			8	12	12	16	

MOV LRB, obj

16-bit Move

obj

A, #N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

LRB ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the local register base (LRB).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	A4	8A					4	—
#N_{16}	57	N_L	N_H				6	—
erN	44+N	7F					4	8
DP	92	7F					4	—
X1	90	7F					4	—
X2	91	7F					4	—
USP	A1	7F					4	—
SSP	A0	7F					4	—
LRB	A4	7F					4	—
off N_8	B4	N_8	7F				6	11
N_8	B5	N_8	7F				6	—
[DP]	B2	7F					6	10
±N_8[USP]	B3	N_8	7F				7	11
N_{16}[X1]	B0	N_L	N_H	7F			8	12
N_{16}[X2]	B1	N_L	N_H	7F			8	12

MOV obj, A

16-bit Move

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← A (word long)

Description

This instruction transfers the contents of the accumulator to the addressing object (word long).

This instruction functionally overlap the part of the "ST A, obj" instruction. This instruction differs from the "ST A, obj" instruction in that the execution of this instruction is not affected by the data descriptor (DD). On the other hand, the instruction "ST A, obj" incurs fewer bytes and cycles, hence the use of the "ST A, obj" is preferred because it results in better program efficiency.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	8A					4	12
DP	52						2	—
X1	50						2	—
X2	51						2	—
USP	A1	8A					4	—
SSP	A0	8A					4	—
LRB	A4	8A					4	—
off N_8	B4	N_8	8A				6	15
N_8	B5	N_8	8A				6	—
[DP]	B2	8A					6	14
$\pm N_8$[USP]	B3	N_8	8A				7	15
N_{16}[X1]	B0	N_L	N_H	8A			8	16
N_{16}[X2]	B1	N_L	N_H	8A			8	16

MOV obj, #N_{16}

16-bit Move

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N'_{16}[X1], N'_{16}[X2]

Function

obj ← #N_{16} (word long)

Description

This instruction transfers the contents of the 16-bit immediate value (#N_{16}) to the addressing object (word long).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	98	N_L	N_H			8	16
DP	62	N_L	N_H				6	—
X1	60	N_L	N_H				6	—
X2	61	N_L	N_H				6	—
USP	A1	98	N_L	N_H			8	—
SSP	A0	98	N_L	N_H			8	—
LRB	57	N_L	N_H				6	—
off N_8	B4	N_8	98	N_L	N_H		10	19
N_8	B5	N_8	98	N_L	N_H		10	—
[DP]	B2	98	N_L	N_H			10	18
$\pm N_8$[USP]	B3	N_8	98	N_L	N_H		11	19
N'_{16}[X1]	B0	N'_L	N'_H	98	N_L	N_H	12	20
N'_{16}[X2]	B1	N'_L	N'_H	98	N_L	N_H	12	20

MOV off N_8, obj

16-bit Move

obj

A, #N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

off N_8 ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the data memory (off N_8) specified by direct page addressing.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES				
							INT	INT	EXT	EXT	← off N_8
	1	2	3	4	5	6	INT	EXT	INT	EXT	← obj
A	B4	N8	8A				6	—	15	—	
#N_{16}	B4	N_8	98	N_L	N_H		10	—	19	—	
erN	44+N	7C	N_8				6	—	—	12	
DP	92	7C	N_8				6	—	11	—	
X1	90	7C	N_8				6	—	11	—	
X2	91	7C	N_8				6	—	11	—	
USP	A1	7C	N_8				6	—	11	—	
SSP	A0	7C	N_8				6	—	11	—	
LRB	A4	7C	N_8				6	—	11	—	
off N'_8	B4	N'_8	7C	N_8			8	—	—	18	
N'_8	B5	N'_8	7C	N_8			8	—	13	—	
[DP]	B2	7C	N_8				8	12	13	17	
$\pm N'_8$[USP]	B3	N'_8	7C	N_8			9	13	14	18	
N_{16}[X1]	B0	N_L	N_H	7C	N_8		10	14	15	19	
N_{16}[X2]	B1	N_L	N_H	7C	N_8		10	14	15	19	

MOV PSW, obj

16-bit Move

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

PSW ← obj (word long)

Description

This instruction transfers the addressing object (word long) to the program status word (PSW).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	*

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	7D					4	8
DP	92	7D					4	—
X1	90	7D					4	—
X2	91	7D					4	—
USP	A1	7D					4	—
SSP	A0	7D					4	—
LRB	A4	7D					4	—
off N_8	B4	N_8	7D				6	11
N_8	B5	N_8	7D				6	—
[DP]	B2	7D					6	10
$\pm N_8$[USP]	B3	N_8	7D				7	11
N_{16}[X1]	B0	N_L	N_H	7D			8	12
N_{16}[X2]	B1	N_L	N_H	7D			8	12

MOV SSP, obj

16-bit Move

obj

A, #N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

SSP ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the system stack pointer (SSP).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	A0	8A					4	—
#N_{16}	A0	98	N_L	N_H			8	—
erN	44+N	7E					4	8
DP	92	7E					4	—
X1	90	7E					4	—
X2	91	7E					4	—
USP	A1	7E					4	—
SSP	A0	7E					4	—
LRB	A4	7E					4	—
off N_8	B4	N_8	7E				6	11
N_8	B5	N_8	7E				6	—
[DP]	B2	7E					6	10
±N_8[USP]	B3	N_8	7E				7	11
N_{16}[X1]	B0	N_L	N_H	7E			8	12
N_{16}[X2]	B1	N_L	N_H	7E			8	12

MOV USP, obj

16-bit Move

obj

A, #N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

USP ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the user stack pointer (USP).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	A1	8A					4	—
#N_{16}	A1	98	N_L	N_H			8	—
erN	44+N	7B					4	8
DP	92	7B					4	—
X1	90	7B					4	—
X2	91	7B					4	—
USP	A1	7B					4	—
SSP	A0	7B					4	—
LRB	A4	7B					4	—
off N_8	B4	N_8	7B				6	11
N_8	B5	N_8	7B				6	—
[DP]	B2	7B					6	10
±N_8[USP]	B3	N_8	7B				7	11
N_{16}[X1]	B0	N_L	N_H	7B			8	12
N_{16}[X2]	B1	N_L	N_H	7B			8	12

MOV X1, obj

16-bit Move

obj

A, #N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

X1 ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the index register (X1).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	50						2	—
#N_{16}	60	N_L	N_H				6	—
erN	44+N	78					4	8
DP	92	78					4	—
X1	90	78					4	—
X2	91	78					4	—
USP	A1	78					4	—
SSP	A0	78					4	—
LRB	A4	78					4	—
off N_8	B4	N_8	78				6	11
N_8	B5	N_8	78				6	—
[DP]	B2	78					6	10
±N_8[USP]	B3	N_8	78				7	11
N_{16}[X1]	B0	N_L	N_H	78			8	12
N_{16}[X2]	B1	N_L	N_H	78			8	12

MOV X2, obj

16-bit Move

obj

A, #N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

X2 ← obj (word long)

Description

This instruction transfers the contents of the addressing object (word long) to the index register (X2).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	51						2	—
#N_{16}	61	N_L	N_H				6	—
erN	44+N	79					4	8
DP	92	79					4	—
X1	90	79					4	—
X2	91	79					4	—
USP	A1	79					4	—
SSP	A0	79					4	—
LRB	A4	79					4	—
off N_8	B4	N_8	79				6	11
N_8	B5	N_8	79				6	—
[DP]	B2	79					6	10
±N_8[USP]	B3	N_8	79				7	11
N_{16}[X1]	B0	N_L	N_H	79			8	12
N_{16}[X2]	B1	N_L	N_H	79			8	12

MOVB A, obj

8-bit Move

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

A_L ← obj (byte long)

Description

This instruction transfers the contents (byte long) of the addressing object to the lower byte of the accumulator (A_L).

This instruction sets the data descriptor (DD) to 0.

Also, this instruction functionally overlaps part of the "LB A, obj" instruction. This instruction differs from "LB A, obj" instruction in that this instruction is not affected by the stack flag (SF) and does not affect the zero flag (ZF). On the other hand, the "LB A, obj" instruction incurs comparatively fewer numbers of bytes and cycles, hence the use of "LB A, obj" is preferred because it results in better program efficiency.

Flags

Flags affected by execution:

ZF	CF	HC	DD
			0

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	99					4	6
PSWH	A2	99					4	—
PSWL	A3	99					4	—
off N_8	C4	N_8	99				6	9
N_8	C5	N_8	99				6	—
[DP]	C2	99					6	8
$\pm N_8$[USP]	C3	N_8	99				7	9
N_{16}[X1]	C0	N_L	N_H	99			8	10
N_{16}[X2]	C1	N_L	N_H	99			8	10

MOVB obj, A

8-bit Move

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← A_L (byte long)

Description

This instruction transfers the contents of the lower byte of the accumulator (A_L) to the addressing object (byte long).

This instruction functionally overlaps part of the "STB A, obj" instruction. This instruction differs from "STB A, obj" instruction in that this instruction is not affected by the data descriptor (DD). On the other hand, "STB A, obj" incurs comparatively fewer numbers of bytes and cycles, hence the use of "STB A, obj" is preferred because of the resulting program efficiency.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	8A					4	8
PSWH	A2	8A					4	—
PSWL	A3	8A					4	—
off N_8	C4	N_8	8A				6	11
N_8	C5	N_8	8A				6	—
[DP]	C2	8A					6	10
$\pm N_8$[USP]	C3	N_8	8A				7	11
N_{16}[X1]	C0	N_L	N_H	8A			8	12
N_{16}[X2]	C1	N_L	N_H	8A			8	12

MOVB off N_8, obj

8-bit Move

obj

A, #N'_8, rN, PSWH, PSWL, off N'_8, N'_8, [DP], ±N'_8[USP], N_{16}[X1], N_{16}[X2]

Function

off N_8 ← obj (byte long)

Description

This instruction transfers the content (byte long) of the addressing object to the data memory specified by direct page addressing (off N_8).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES				
							INT	INT	EXT	EXT	← off N_8
	1	2	3	4	5	6	INT	EXT	INT	EXT	← obj
A	C4	N_8	8A				6	—	13	—	
#N'_8	C4	N_8	98	N'_8			8	—	13	—	
rN	20+N	7C	N_8				6	—	—	11	
PSWH	A2	7C	N_8				6	—	9	—	
PSWL	A3	7C	N_8				6	—	9	—	
off N'_8	C4	N'_8	7C	N_8			8	—	—	14	
N'_8	C5	N'_8	7C	N_8			8	—	11	—	
[DP]	C2	7C	N_8				8	10	11	13	
±N'_8[USP]	C3	N'_8	7C	N_8			9	11	12	14	
N_{16}[X1]	C0	N_L	N_H	7C	N_8		10	12	13	15	
N_{16}[X2]	C1	N_L	N_H	7C	N_8		10	12	13	15	

MOVB obj, #N_8

8-bit Move

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← #N_8 (byte long)

Description

This instruction transfers the byte long immediate value (#N_8) to the addressing object (byte long).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	98+N	N_8					4	6
PSWH	A2	98	N_8				6	—
PSWL	A3	98	N_8				6	—
off N'_8	C4	N'_8	98	N_8			8	13
N'_8	C5	N'_8	98	N_8			8	—
[DP]	C2	98	N_8				8	12
$\pm N'_8$[USP]	C3	N'_8	98	N_8			9	13
N_{16}[X1]	C0	N_L	N_H	98	N_8		10	14
N_{16}[X2]	C1	N_L	N_H	98	N_8		10	14

MOVB PSWH, obj

8-bit Move

obj

A, #N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

PSWH ← obj (byte long)

Description

This instruction transfers the addressing object (byte long) to the upper byte of the PSW (PSWH).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	*

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A	A2	8A					4	—
#N_8	A2	98	N_8				6	—
rN	20+N	89					4	8
PSWH	A2	89					4	—
PSWL	A3	89					4	—
off N_8	C4	N_8	89				8	9
N_8	C5	N_8	89				6	—
[DP]	C2	89					6	8
±N_8[USP]	C3	N_8	89				7	9
N_{16}[X1]	C0	N_L	N_H	89			8	10
N_{16}[X2]	C1	N_L	N_H	89			8	10

MOVB PSWL, obj

8-bit Move

obj

A, #N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

PSWL ← obj (byte long)

Description

This instruction transfers the addressing object (byte long) to the lower byte of the PSW (PSWL).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	A3	8A					4	—
#N_8	A3	98	N_8				6	—
rN	20+N	88					4	6
PSWH	A2	88					4	—
PSWL	A3	88					4	—
off N_8	C4	N_8	88				6	9
N_8	C5	N_8	88				6	—
[DP]	C2	88					6	8
±N_8[USP]	C3	N_8	88				7	9
N_{16}[X1]	C0	N_L	N_H	88			8	10
N_{16}[X2]	C1	N_L	N_H	88			8	10

MOVB rN, obj

8-bit Move (Byte)

obj

A, #N_8, rN', PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

rN ← obj (byte long)

Description

This instruction transfers the contents (byte long) of the addressing object to the local register (rN).

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES				
							INT	INT	EXT	EXT	← rN
	1	2	3	4	5	6	INT	EXT	INT	EXT	← obj
A	20+N	8A					4	—	8	—	
#N_8	98+N	N_8					4	—	6	—	
rN'	20+N'	48+N					4	—	—	8	
PSWH	A2	48+N					4	—	6	—	
PSWL	A3	48+N					4	—	6	—	
off N_8	C4	N_8	48+N				6	—	—	11	
N_8	C5	N_8	48+N				6	—	8	—	
[DP]	C2	48+N					6	8	8	10	
±N_8[USP]	C3	N_8	48+N				7	9	9	11	
N_{16}[X1]	C0	N_L	N_H	48+N			8	10	10	12	
N_{16}[X2]	C1	N_L	N_H	48+N			8	10	10	12	

MUL

16-bit Multiply

obj

Function

(er1, A) ← A × er0

Description

16 bits × 16 bits → 32-bit multiply instruction

This instruction performs multiplication between the contents of the accumulator and the extended local register (er0). The upper 16 bits of the resulting product is stored in the extended local register (er1) and the lower 16 bits of the product is stored in the accumulator.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES		
	BYTE								
	1	2	3	4	5	6	INT	EXT	← er0, er1
	90	35					27	35	

MULB

8-bit Multiply

obj

Function

$A \leftarrow A_L \times r0$

Description

8 bits × 8 bits ⟶ 16-bit multiply instruction

This instruction performs multiplication between the lower byte of the accumulator (A_L) and the local register (r0). The resulting product (16 bits) is stored in the accumulator.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	A2	34					19	21

NOP

No Operation

obj

Function

No Operation

Description

This instruction performs no operation, but consumes time for 2 cycles.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	00						2	

OR A, obj

16-bit logical OR

obj

#N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A ← A ∨ obj (word long)

Description

This instruction performs the logical OR operation on the contents of the accumulator and the addressing object. The result (16 bits) is stored in the accumulator.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}	E6	N_L	N_H				6	—
erN	68+N						3	7
DP	92	E2					4	—
X1	90	E2					4	—
X2	91	E2					4	—
USP	A1	E2					4	—
SSP	A0	E2					4	—
LRB	A4	E2					4	—
off N_8	B4	N_8	E2				4	9
N_8	B5	N_8	E2				6	—
[DP]	B2	E2					6	10
±N_8[USP]	B3	N_8	E2				7	11
N_{16}[X1]	B0	N_L	N_H	E2			8	12
N_{16}[X2]	B1	N_L	N_H	E2			8	12

OR obj, A

16-bit logical OR

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∨ A (word long)

Description

This instruction performs the word long logical OR operation on the contents of the addressing object and the accumulator.

The result (16 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	E1					5	13
DP	92	E1					5	—
X1	90	E1					5	—
X2	91	E1					5	—
USP	A1	E1					5	—
SSP	A0	E1					5	—
LRB	A4	E1					5	—
off N_8	B4	N_8	E1				7	16
N_8	B5	N_8	E1				7	—
[DP]	B2	E1					7	15
$\pm N_8$[USP]	B3	N_8	E1				8	16
N_{16}[X1]	B0	N_L	N_H	E1			9	17
N_{16}[X2]	B1	N_L	N_H	E1			9	17

OR obj, off N_8

16-bit logical OR

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∨ off N_8 (word long)

Description

This instruction performs the word long logical OR operation on the contents of the addressing object and the data memory specified by direct page addressing (off N_8).

The result (16 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	INT	EXT	EXT	← obj
							INT	EXT	INT	EXT	← off N_8
A											
#N_{16}											
erN	44+N	E3	N_8				7	—	—	20	
DP	92	E3	N_8				7	16	—	—	
X1	90	E3	N_8				7	16	—	—	
X2	91	E3	N_8				7	16	—	—	
USP	A1	E3	N_8				7	16	—	—	
SSP	A0	E3	N_8				7	16	—	—	
LRB	A4	E3	N_8				7	16	—	—	
off N'_8	B4	N'_8	E3	N_8			9	—	—	23	
N'_8	B5	N'_8	E3	N_8			9	18	—	—	
[DP]	B2	E3	N_8				9	18	13	22	
$\pm N'_8$[USP]	B3	N'_8	E3				10	19	14	23	
N_{16}[X1]	B0	N_L	N_H	E3	N_8		11	20	15	24	
N_{16}[X2]	B1	N_L	N_H	E3	N_8		11	20	15	24	

ORB A, obj

8-bit logical OR

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

$A_L \leftarrow A_L \vee$ obj (byte long)

Description

This instruction performs the logical OR operation on the contents of the lower byte of the accumulator (A_L) and the contents of the addressing object (byte long).

The result (8 bits) is stored in the lower byte of the accumulator (A_L). *This instruction is affected by the data descriptor (DD).*
For this instruction to be executed correctly it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_8	E6	N_8					4	—
rN	68+N						3	5
PSWH	A2	E2					4	—
PSWL	A3	E2					4	—
off N_8	E7	N_8					4	7
N_8	C5	N_8	E2				6	—
[DP]	C2	E2					6	—
±N_8[USP]	C3	N_8	E2				7	9
N_{16}[X1]	C0	N_L	N_H	E2			8	10
N_{16}[X2]	C1	N_L	N_H	E2			8	10

ORB obj, A

8-bit logical OR

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∨ A_L (byte long)

Description

This instruction performs the logical OR operation on the contents of the lower byte of the accumulator (A_L) and the contents of the addressing object (byte long).

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	E1					5	9
PSWH	A2	E1					5	—
PSWL	A3	E1					5	—
off N_8	C4	N_8	E1				7	12
N_8	C5	N_8	E1				7	—
[DP]	C2	E1					7	11
$\pm N_8$[USP]	C3	N_8	E1				8	12
N_{16}[X1]	C0	N_L	N_H	E1			9	13
N_{16}[X2]	C1	N_L	N_H	E1			9	13

ORB obj, off N_8

8-bit logical OR

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∨ off N_8 (byte long)

Description

This instruction performs the byte long logical OR operation on the contents of the addressing object and the data memory specified by the direct page addressing (off N_8).

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES INT	CYCLES INT	CYCLES EXT	CYCLES EXT
	1	2	3	4	5	6	INT	EXT	INT	EXT
A										
#N_8										
rN	20+N	E3	N_8				7	—	—	14
PSWH	A2	E3	N_8				7	12	—	—
PSWL	A3	E3	N_8				7	12	—	—
off N'_8	C4	N'_8	E3	N_8			9	—	—	17
N'_8	C5	N'_8	E3	N_8			9	14	—	—
[DP]	C2	E3	N_8				9	14	11	16
$\pm N'_8$[USP]	C3	N'_8	E3	N_8			10	15	12	17
N_{16}[X1]	C0	N_L	N_H	E3	N_8		11	16	13	18
N_{16}[X2]	C1	N_L	N_H	E3	N_8		11	16	13	18

ORB obj, #N_8

8-bit logical OR

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∨ #N_8 (byte long)

Description

This instruction performs the logical OR operation on the content of the addressing object (byte long) and the 8-bit immediate value (#N_8).

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	E0	N_8				6	11
PSWH	A2	E0	N_8				6	—
PSWL	A3	E0	N_8				6	—
off N'_8	C4	N'_8	E0	N_8			8	14
N'_8	C5	N'_8	E0	N_8			8	—
[DP]	C2	E0	N_8				8	13
$\pm N'_8$[USP]	C3	N'_8	E0	N_8			9	14
N_{16}[X1]	C0	N_L	N_H	E0	N_8		10	15
N_{16}[X2]	C1	N_L	N_H	E0	N_8		10	15

POPS A

16-bit Pop (system stack)

obj

Function

SSP ← SSP + 2
A ← (SSP) (word long)
DD ← 1

Description

This instruction pops the data from the system stack into the accumulator in a word long operation.

This instruction sets the data descriptor (DD) to 1.

[NOTE] Note that the "PUSHS A" instruction does not affect DD, while the "POPS A" instruction sets DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
			1

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	65						4	8
#N_{16}								
@N								
DP								
X1								
X2								
USP								
SSP								
LRB								
off N_8								
N_8								
[DP]								
±N_8[USP]								
N_{16}[X1]								
N_{16}[X2]								

POPS LRB

16-bit Pop (system stack)

obj

Function

SSP ← SSP + 2
LRB ← (SSP) (word long)

Description

This instruction pops the data from the system stack into LRB in a word long operation.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN								
DP								
X1								
X2								
USP								
SSP								
LRB	64						4	8
off N_8								
N_8								
[DP]								
$\pm N_8$[USP]								
N_{16}[X1]								
N_{16}[X2]								

PUSHS obj

16-bit Push (system stack)

obj

A, LRB

Function

(SSP) ← obj (word long)
SSP ← SSP - 2

Description

This instruction pushes the contents of the addressing object into the system stack in a word long operation.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	55						3	7
#N_{16}								
erN								
DP								
X1								
X2								
USP								
SSP								
LRB	54						3	7
off N_8								
N_8								
[DP]								
$\pm N_8$[USP]								
N_{16}[X1]								
N_{16}[X2]								

RB obj.bit

Reset Bit (direct bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

IF obj.bit = 0 THEN Z ← 1
ELSE Z ← 0
obj.bit ← 0

Description

This instruction resets the specified bit of the addressing object, "obj" (byte long), to 0. The location of the bit is specified by the "bit" in the operand. The value of the "bit" ranges from 0 to 7.

Prior to the execution of this instruction, the content of the specified bit is examined and the zero flag (ZF) is set accordingly. If the specified bit is 0 prior to the execution of the instruction, the zero flag is 1, otherwise the ZF=0.

Flags

Flags affected by execution:

ZF	CY	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	08+n					7	13
PSWH	A2	08+n					7	—
PSWL	A3	08+n					7	—
off N_8	C4	N_8	08+n				9	16
N_8	C5	N_8	08+n				9	—
[DP]	C2	08+n					9	15
$\pm N_8$[USP]	C3	N_8	08+n				10	16
N_{16}[X1]	C0	N_L	N_H	08+n			11	17
N_{16}[X2]	C1	N_L	N_H	08+n			11	17

[NOTE] "n" is determined from the value of the "bit", and ranges from 0 to 7.

RBR obj

Reset Bit (register indirect bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

IF obj.bit $(A_{2\text{-}0})$ = 0 THEN Z ← 1
ELSE Z ← 0
obj.bit $(A_{2\text{-}0})$ ← 0

Description

This instruction resets the specified bit of the operand "obj" (byte long) which is the addressing object. The location of the bit is specified by the contents of the lower 3 bits of the accumulator $(A_{2\text{-}0})$ at the execution time of the instruction.

Prior to the execution of the instruction, the contents of the specified bit is examined and the zero flag (ZF) is set accordingly. If the specified bit is 0 prior to the execution of the instruction, the zero flag is 1; otherwise ZF=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	12					7	13
PSWH	A2	12					7	—
PSWL	A3	12					7	—
off N_8	C4	N_8	12				9	16
N_8	C5	N_8	12				9	—
[DP]	C2	12					9	15
$\pm N_8$[USP]	C3	N_8	12				10	16
N_{16}[X1]	C0	N_L	N_H	12			11	17
N_{16}[X2]	C1	N_L	N_H	12			11	17

RC — Reset Carry

obj

Function

C ← 0

Description

This instruction resets the carry (C) by making C=0

Flags

Flags affected by execution:

ZF	CF	HC	DD
	0		

Flags affecting execution:

DD

Codes/Cycles

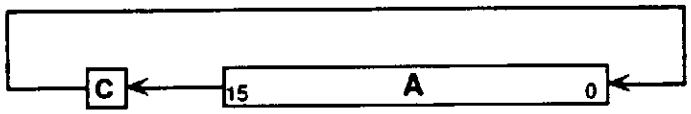
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	95						2	—

ROL A

16-bit Rotate Left

obj

Function



Description

This instruction rotates the contents (word long) of the accumulator including the carry, one bit to the left.

This instruction is affected by the data descriptor (DD).
For the instruction to be executed correctly, it is necessary to set DD=1. When DD=0, the instruction actually executed becomes "ROLB A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	33						2	—
#N_{16}								
erN								
DP								
X1								
X2								
USP								
SSP								
LRB								
off N_8								
N_8								
[DP]								
$\pm N_8$[USP]								
N_{16}[X1]								
N_{16}[X2]								

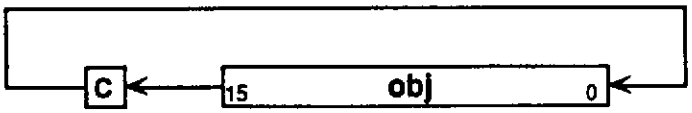
ROL obj

16-bit Rotate Left

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



Description

This instruction rotates the content (word long) of the addressing object including the carry, one bit to the left.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

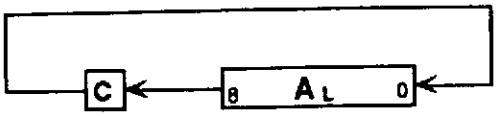
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	B7					5	13
DP	92	B7					5	—
X1	90	B7					5	—
X2	91	B7					5	—
USP	A1	B7					5	—
SSP	A0	B7					5	—
LRB	A4	B7					5	—
off N_8	B4	N_8	B7				7	16
N_8	B5	N_8	B7				7	—
[DP]	B2	B7					7	15
$\pm N_8$[USP]	B3	N_8	B7				8	16
N_{16}[X1]	B0	N_L	N_H	B7			9	17
N_{16}[X2]	B1	N_L	N_H	B7			9	17

ROLB A

8-bit Rotate Left

obj

Function



Description

This instruction rotates the contents of the lower byte of the accumulator (A_L) including the carry, one bit to the left.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0. When DD=1, the instruction actually executed becomes "ROL A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A	33						2	—
#N_8								
rN								
PSWH								
PSWL								
off N_8								
N_8								
[DP]								
±N_8[USP]								
N_{16}[X1]								
N_{16}[X2]								

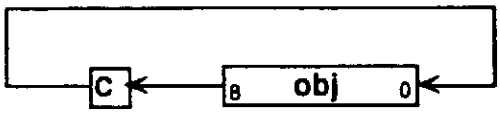
ROLB obj

8-bit Rotate Left

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



Description

This instruction rotates the contents of the addressing object (word long) including the carry, one bit to the left.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

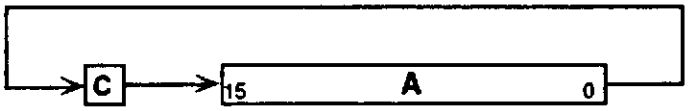
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	B7					5	9
PSWH	A2	B7					5	—
PSWL	A3	B7					5	—
off N_8	C4	N_8	B7				7	12
N_8	C5	N_8	B7				7	—
[DP]	C2	B7					7	11
$\pm N_8$[USP]	C3	N_8	B7				8	12
N_{16}[X1]	C0	N_L	N_H	B7			9	13
N_{16}[X2]	C1	N_L	N_H	B7			9	13

ROR A

16-bit Rotate Right

obj

Function



Description

This instruction rotates the contents (word long) of the accumulator including the carry, one bit to the right.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1. when DD=0, the instruction actually executed becomes "RORB A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	43						2	—
#N_{16}								
erN								
DP								
X1								
X2								
USP								
SSP								
LRB								
off N_8								
N_8								
[DP]								
$\pm N_8$[USP]								
N_{16}[X1]								
N_{16}[X2]								

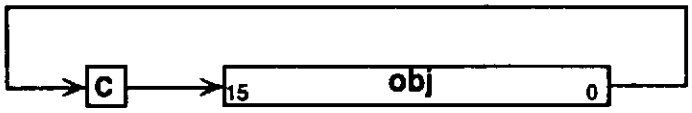
ROR obj

16-bit Rotate Right

obj

erN, DP, X1, X2, USP, SSP, LRB off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



Description

This instruction rotates the contents of the addressing object (word long) including the carry, one bit to the right.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

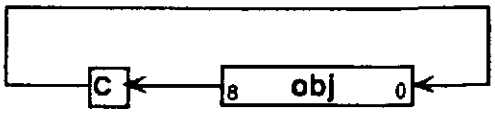
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	C7					5	13
DP	92	C7					5	—
X1	90	C7					5	—
X2	91	C7					5	—
USP	A1	C7					5	—
SSP	A0	C7					5	—
LRB	A4	C7					5	—
off N_8	B4	N_8	C7				7	16
N_8	B5	N_8	C7				7	—
[DP]	B2	C7					7	15
$\pm N_8$[USP]	B3	N_L	C7				8	16
N_{16}[X1]	B0	N_L	N_H	C7			9	17
N_{16}[X2]	B1	N_L	N_H	C7			9	17

RORB A

8-bit Rotate Right

obj

Function



Description

This instruction rotates the contents of the lower byte of the accumulator (A_L) including the carry, one bit to the right.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0. When DD=1, the instruction actually executed becomes "ROR A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A	43						2	—
#N_8								
rN								
PSWH								
PSWL								
off N_8								
N_8								
[DP]								
±N_8[USP]								
N_{16}[X1]								
N_{16}[X2]								

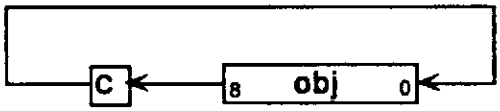
RORB obj

8-bit Rotate Right

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



Description

This instruction rotates the content (word long) of the addressing object including the carry, one bit to the right.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	C7					5	9
PSWH	A2	C7					5	—
PSWL	A3	C7					5	—
off N_8	C4	N_8	C7				7	12
N_8	C5	N_8	C7				7	—
[DP]	C2	C7					7	11
$\pm N_8$[USP]	C3	N_8	C7				8	12
N_{16}[X1]	C0	N_L	N_H	C7			9	13
N_{16}[X2]	C1	N_L	N_H	C7			9	13

RT

Return from Normal Subroutine

obj

Function

SSP ← SSP +2
PC ← (SSP)
SF ← 0

Description

This instruction is used to return from normal subroutines that are called by the instructions "SCAL" and "CAL". (See RTI Instruction)

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	01						7	11

RTI

Return from Interrupt Routine

obj

Function

SSP ← SSP +2
PSW ← (SSP)
SSP ← SSP +2
LRB ← (SSP)
SSP ← SSP +2
A ← (SSP)
SSP ← SSP +2
PC ← (SSP)

Description

This instruction is used to return from the interrupt routines. (See RT instruction.)

The data saved in the system stack are returned to the units in the order : PSW, LRB, A, and PC. The system stack pointer is ultimately incremented by 8.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	02						15	31

SB obj.bit

Set Bit (direct bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

IF obj.bit = 0 THEN Z ← 1
ELSE Z ← 0
obj.bit ← 1

Description

This instruction sets the specified bit of the addressing object (obj) to 1.

The location of the bit is specified by the bit in the second operand. The value of the "bit" is from 0 to 7. Prior to the execution of the instruction, the content of the specified bit is examined and the zero flag (ZF) is set accordingly. If the specified bit is 0 prior to the execution of the instruction, the zero flag is 1, otherwise ZF=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	18+n					7	13
PSWH	A2	18+n					7	—
PSWL	A3	18+n					7	—
off N_8	C4	N_8	18+n				9	16
N_8	C5	N_8	18+n				9	—
[DP]	C2	18+n					9	15
$\pm N_8$[USP]	C3	N_8	18+n				10	16
N_{16}[X1]	C0	N_L	N_H	18+n			11	17
N_{16}[X2]	C1	N_L	N_H	18+n			11	17

[NOTE] The value of n is determined following evaluation of "bit". The value of n ranges from 0 to 7.

SBC A, obj

16-bit Subtract with Carry

obj

#N_8, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A ← A - obj - C (word long)

Description

This instruction subtracts from the contents of the accumulator, the contents of the addressing object, and the contents of the carry (C) in the word long operation.

The resulting difference (16 bits) is stored in the accumulator. The borrow that occurs during the subtraction is stored in the carry (C).

The instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}	B6	N_L	N_H				6	—
erN	38+N						3	7
DP	92	B2					4	—
X1	90	B2					4	—
X2	91	B2					4	—
USP	A1	B2					4	—
SSP	A0	B2					4	—
LRB	A4	B2					4	—
off N_8	B7	N_8					4	9
N_8	B5	N_8	B2				6	—
[DP]	B2	B2					6	10
±N_8[USP]	B3	N_8	B2				7	11
N_{16}[X1]	B0	N_L	N_H	B2			8	12
N_{16}[X2]	B1	N_L	N_H	B2			8	12

SBC obj, A

16-bit Subtract with Carry

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - A - C (word long)

Description

This instruction subtracts from the contents of the addressing object, the contents of the accumulator, and the contents of the carry (C) in a word long operation.

The resulting difference (16 bits) is stored in the addressing object. The borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	B1					5	13
DP	92	B1					5	—
X1	90	B1					5	—
X2	91	B1					5	—
USP	A1	B1					5	—
SSP	A0	B1					5	—
LRB	A4	B1					5	—
off N_8	B4	N_8	B1				7	16
N_8	B5	N_8	B1				7	—
[DP]	B2	B1					7	15
$\pm N_8$[USP]	B3	N_8	B1				8	16
N_{16}[X1]	B0	N_L	N_H	B1			9	17
N_{16}[X2]	B1	N_L	N_H	B1			9	17

SBC obj, off N_8

16-bit Subtract with Carry

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - off N_8 - C (word long)

Description

This instruction subtracts from the contents of the addressing object, the contents of the data memory specified by direct page addressing (off N_8), and the contents of the carry (C) in a word long operation.

The resulting difference (16 bits) is stored in the addressing object, and the borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT ← obj	INT ← obj	EXT ← obj	EXT ← obj	INT ← off N_8	EXT ← off N_8	INT ← off N_8	EXT ← off N_8
A														
#N_{16}														
erN	44+N	B3	N_8				7	—	—	20				
DP	92	B3	N_8				7	16	—	—				
X1	90	B3	N_8				7	16	—	—				
X2	91	B3	N_8				7	16	—	—				
USP	A1	B3	N_8				7	16	—	—				
SSP	A0	B3	N_8				7	16	—	—				
LRB	A4	B3	N_8				7	16	—	—				
off N'_8	B4	N'_8	B3	N_8			9	—	—	23				
N'_8	B5	N'_8	B3	N_8			9	18	—	—				
[DP]	B2	B3	N_8				9	18	13	22				
$\pm N'_8$[USP]	B3	N'_8	B3	N_8			10	19	14	23				
N_{16}[X1]	B0	N_L	N_H	B3	N_8		11	20	15	24				
N_{16}[X2]	B1	N_L	N_H	B3	N_8		11	20	15	24				

SBC obj, #N_{16}

16-bit Subtract with Carry

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N'$_{16}$[X1], N'$_{16}$[X2]

Function

obj ← obj - #N_{16} - C (word long)

Description

This instruction subtracts from the contents of the addressing object (word long), the 16 bit immediate value (#N_{16}), and the contents of the carry (C).

The resulting difference (16 bits) is stored in the addressing object, and the borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	B0	N_L	N_H			8	17
DP	92	B0	N_L	N_H			8	—
X1	90	B0	N_L	N_H			8	—
X2	91	B0	N_L	N_H			8	—
USP	A1	B0	N_L	N_H			8	—
SSP	A0	B0	N_L	N_H			8	—
LRB	A4	B0	N_L	N_H			8	—
off N_8	B4	N_8	B0	N_L	N_H		10	20
N_8	B5	N_8	B0	N_L	N_H		10	—
[DP]	B2	B0	N_L	N_H			10	19
±N_8[USP]	B3	N_8	B0	N_L	N_H		11	20
N'$_{16}$[X1]	B0	N'$_L$	N'$_H$	B0	N_L	N_H	12	21
N'$_{16}$[X2]	B1	N'$_L$	N'$_H$	B0	N_L	N_H	12	21

SBCB A, obj

8-bit Subtract with Carry

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

$A_L \leftarrow A_L$ - obj - C (byte long)

Description

This instruction subtracts from the contents of the lower byte of the accumulator (A_L), the contents (byte long) of the addressing object, and the contents of the carry (C). The resulting sum (8 bits) is stored in the lower byte of the accumulator (A_L), and the carry from the highest (most significant) bit is stored in the carry (C).

This instruction is affected by the data descriptor (DD).
For the instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8	B6	N_8					4	—
rN	38+N						3	5
PSWH	A2	B2					4	—
PSWL	A3	B2					4	—
off N_8	B7	N_8					4	7
N_8	C5	N_8	B2				6	—
[DP]	C2	B2					6	—
±N_8[USP]	C3	N_8	B2				7	9
N_{16}[X1]	C0	N_L	N_H	B2			8	10
N_{16}[X2]	C1	N_L	N_H	B2			8	10

SBCB obj, A

8-bit Subtract with Carry

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - A_L - C (byte long)

Description

This instruction subtracts from the contents of the lower byte of the accumulator (A_L), the contents (byte long) of the addressing object, and the contents of the carry (C).

The resulting sum (8 bits) is stored in the addressing object, and the borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	B1					5	9
PSWH	A2	B1					5	—
PSWL	A3	B1					5	—
off N_8	C4	N_8	B1				7	12
N_8	C5	N_8	B1				7	—
[DP]	C2	B1					7	11
$\pm N_8$[USP]	C3	N_8	B1				8	12
N_{16}[X1]	C0	N_L	N_H	B1			9	13
N_{16}[X2]	C1	N_L	N_H	B1			9	13

SBCB obj, off N_8

8-bit Subtract with Carry

obj

rN, PSWH, PSWL, off N'8, N'8, [DP], ±N'8[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - off N_8 - C (byte long)

Description

This instruction subtracts from the contents (byte long) of the addressing object, the contents of the data memory specified by direct page addressing (off N_8), and the contents of the carry (C).

The resulting difference (8 bits) is stored in the addressing object, and the borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	INT EXT	EXT INT	EXT EXT	← obj / ← off N_8
A											
#N_8											
rN	20+N	B3	N_8				7	—	—	14	
PSWH	A2	B3	N_8				7	12	—	—	
PSWL	A3	B3	N_8				7	12	—	—	
off N'8	C4	N'8	B3	N_8			9	—	—	17	
N'8	C5	N'8	B3	N_8			9	14	—	—	
[DP]	C2	B3	N_8				9	14	11	16	
±N'8[USP]	C3	N'8	B3	N_8			10	15	12	17	
N_{16}[X1]	C0	N_L	N_H	B3	N_8		11	16	13	18	
N_{16}[X2]	C1	N_L	N_H	B3	N_8		11	16	13	18	

SBCB obj, #N8

8-bit Subtract with Carry

obj

rN, PSWH, PSWL, off N'8, N'8, [DP], ±N'8[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - #N8 - C (byte long)

Description

This instruction subtracts from the contents (byte long) of the addressing object, the 8 bit immediate value (#N8), and the contents of the carry (C).

The resulting difference (8 bits) is stored in the addressing object, and the borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N8								
rN	20+N	B0	N8				6	11
PSWH	A2	B0	N8				6	—
PSWL	A3	B0	N8				6	—
off N'8	C4	N'8	B0	N8			8	14
N'8	C5	N'8	B0	N8			8	—
[DP]	C2	B0	N8				8	13
±N'8[USP]	C3	N'8	B0	N8			9	14
N_{16}[X1]	C0	N_L	N_H	B0	N8		10	15
N_{16}[X2]	C1	N_L	N_H	B0	N8		10	15

SBR obj

Set Bit (register indirect bit addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

IF obj.bit (A_{2-0}) = 0 THEN Z ← 1
ELSE Z ← 0
obj.bit (A_{2-0}) ← 0

Description

This instruction sets the specified bit of the addressing object obj (byte long) to 1.

The bit location is specified by the contents of the lower 3 bits of the accumulator (A_{2-0}) at the execution time of the instruction.

Prior to the execution of the instruction, the contents of the specified bit is examined and the zero flag (ZF) is set accordingly. If the specified bit is 0 prior to the execution of the instruction, the zero flag is 1, otherwise ZF=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	11					7	13
PSWH	A2	11					7	—
PSWL	A3	11					7	—
off N_8	C4	N_8	11				9	16
N_8	C5	N_8	11				9	—
[DP]	C2	11					9	15
$\pm N_8$[USP]	C3	N_8	11				10	16
N_{16}[X1]	C0	N_L	N_H	11			11	17
N_{16}[X2]	C1	N_L	N_H	11			11	17

SC — Set Carry

obj

Function

C ← 1

Description

This instruction sets the carry (C) to 1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	1		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	85						2	—

SCAL address

Short Call

obj

Function

(SSP) ← PC + 2
SSP ← SSP - 2
SF ← 0
PC ← address

where PC* - 128 ≤ address ≤ PC* + 127
PC* = head address of the instruction following the SCAL instruction
= (address of the first byte of the SCAL instruction) + 2

Description

This instruction is a call instruction with a 7-bit displacement whose origin is at the starting address of the next instruction. The range of the call is from -128 to +127 with the starting address of the next instruction serving as the origin.

[NOTE] This call instruction manipulates the PC with the displacement. However, in the source program of the assembler (RAS66K), the destination address of a call, such as a label and not the actual displacement, is entered as the operand of the instruction. The assembler figures out the amount of the displacement, which if it stays within the allowable range, is used to generate the machine codes. If the amount of the displacement is outside of the allowable range, the assembler issues an error warning.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	31	N_8					9	13

[NOTE] N_8 is the 8-bit code to represent the displacement in the range from -128 to +127. Negative values are expressed as 2's complement.

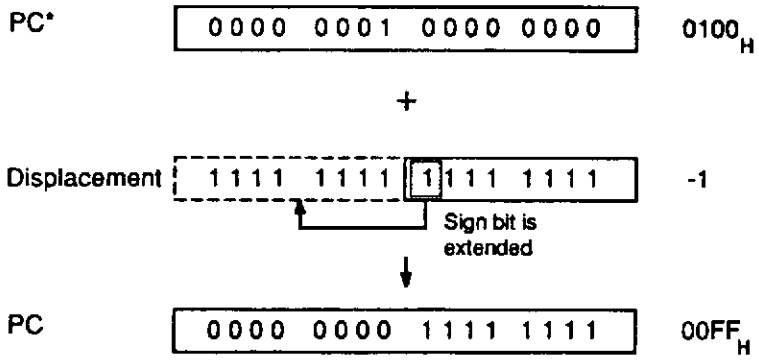
[EXAMPLE]

Displacement	N_8
0	00000000_B
+1	00000001_B
+127	01111111_B
-1	11111111_B
-127	10000001_B
-128	10000000_B

Example

Case where Displacement = -1, PC* = 0100H

Machine Code: CB_H FF_H



SJ address

Short Jump

obj

Function

PC ← address

where PC* - 128 ≤ address ≤ PC* + 127
PC* = head address of the instruction following the SCAL instruction
= (address of the first byte of the SCAL instruction) + 2

Description

This instruction performs a jump within a 7-bit displacement whose origin is at the starting address of the instruction that follows this SJ instruction. The range of the jump of this instruction is from -128 to +127 with the starting address of the next instruction serving as the origin.

[NOTE] This jump instruction manipulates the PC with the displacement. However, in the source program of the assembler (RAS66K), the destination address of the call such as a label and not the displacement, is entered as the operand of the instruction. The assembler figures out the amount of the displacement, which if it stays within the callable range, is used to generate the machine codes. If the amount of the displacement is outside of the callable range, the assembler issues an error warning.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	CB	N_8					8	—

[NOTE] N_8 is the 8-bit code to represent the displacement in the range from -128 to +127. Negative values are expressed as the complement of 2.

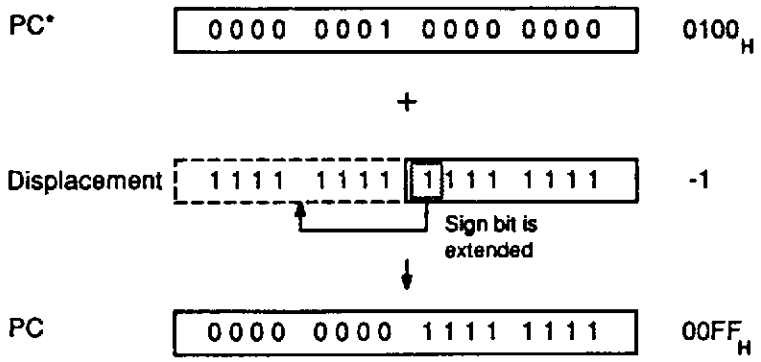
[EXAMPLE]

Displacement	N_8
0	00000000_B
+1	00000001_B
+127	01111111_B
-1	11111111_B
-127	10000001_B
-128	10000000_B

Example

Case where Displacement = -1, PC* = 0100H

Machine Code: CB_H FF_H

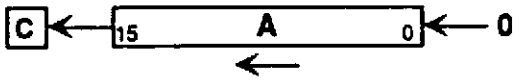


SLL A

16-bit Left Shift

obj

Function



$C \leftarrow A_{15}$
$A_n \leftarrow A_{n-1}$ (n=15—1)
$A_0 \leftarrow 0$

Description

This instruction shifts the contents (word long) of the accumulator one bit to the left. The overflow from A_{15} is entered in the carry (C).

0 (zero) is entered in A_0.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1. When DD=0, the instruction actually executed becomes "SLLB A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	53						2	—
#N_{16}								
erN								
DP								
X1								
X2								
USP								
SSP								
LRB								
off N_8								
N_8								
[DP]								
$\pm N_8$[USP]								
N_{16}[X1]								
N_{16}[X2]								

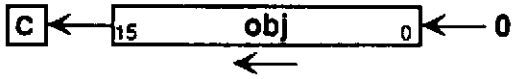
SLL obj

16-bit Left Shift

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



$C \leftarrow obj_{15}$
$obj_n \leftarrow obj_{n-1}$ (n=15—1)
$obj_0 \leftarrow 0$

Description

This instruction shifts the contents of the addressing object (word long) (obj) one bit to the left.

The overflow from obj_{15} is entered in the carry (C). Also, 0 (zero) is entered in obj_0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD

Codes/Cycles

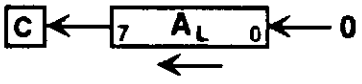
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	D7					5	13
DP	92	D7					5	—
X1	90	D7					5	—
X2	91	D7					5	—
USP	A1	D7					5	—
SSP	A0	D7					5	—
LRB	A4	D7					5	—
off N_8	B4	N_8	D7				7	16
N_8	B5	N_8	D7				7	—
[DP]	B2	D7					7	15
$\pm N_8$[USP]	B3	N_8	D7				8	16
N_{16}[X1]	B0	N_L	N_H	D7			9	17
N_{16}[X2]	B1	N_L	N_H	D7			9	17

SLLB A

8-bit Left Shift

obj

Function



$C \leftarrow A_7$
$A_n \leftarrow A_{n-1}$ (n=7—1)
$A_0 \leftarrow 0$

Description

This instruction shifts the contents of the lower byte of the accumulator (A_L) one bit to the left.

The overflow from A_7 is entered in the carry (C) and 0 (zero) is entered in A_0.

This instruction is affected by the data descriptor (DD).
For this instruction to be performed correctly, it is necessary to set DD=0. When DD=1, the instruction actually executed becomes "SLL A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	53						2	—
#N_8								
rN								
PSWH								
PSWL								
off N_8								
N_8								
[DP]								
±N_8[USP]								
N_{16}[X1]								
N_{16}[X2]								

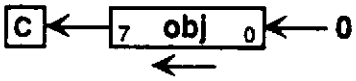
SLLB obj

8-bit Left Shift

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



$C \leftarrow obj_7$
$obj_n \leftarrow obj_{n-1}$ (n=7—1)
$obj_0 \leftarrow 0$

Description

This instruction shifts the contents (byte long) of the the addressing object (obj) one bit to the left.

The overflow from obj_7 is entered in the carry (C). Also, 0 (zero) is entered in obj_0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

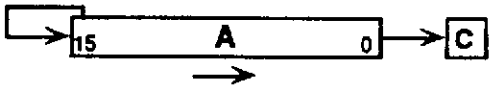
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	D7					5	9
PSWH	A2	D7					5	—
PSWL	A3	D7					5	—
off N_8	C4	N_8	D7				7	12
N_8	C5	N_8	D7				7	—
[DP]	C2	D7					7	11
$\pm N_8$[USP]	C3	N_8	D7				8	12
N_{16}[X1]	C0	N_L	N_H	D7			9	13
N_{16}[X2]	C1	N_L	N_H	D7			9	13

SRA A

16-bit Arithmetic Right Shift

obj

Function



$C \leftarrow A_0$
$A_n \leftarrow A_{n+1}$ (n=14—0)
$A_{15} \leftarrow A_{15}$

Description

This instruction arithmetically shifts the contents (word long) of the accumulator one bit to the right.

The overflow from A_0 is entered in the carry (C), while bit A_{15} following the shift is entered by A_{15} itself.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1. When DD=0, the instruction actually executed becomes "SRAB A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	73						2	—
#N_{16}								
erN								
DP								
X1								
X2								
USP								
SSP								
LRB								
off N_8								
N_8								
[DP]								
$\pm N_8$[USP]								
N_{16}[X1]								
N_{16}[X2]								

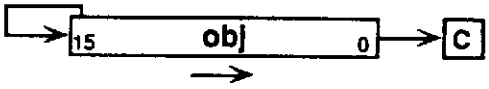
SRA obj

16-bit Arithmetic Right Shift

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



$C \leftarrow obj_0$
$obj_n \leftarrow obj_{n+1}$ (n=14—0)
$obj_{15} \leftarrow obj_{15}$

Description

This instruction arithmetically shifts the contents of the addressing object (word long) (obj) one bit to the right.

The overflow from obj0 is entered in the carry (C), while the bit obj_{15} following the shift is entered by obj_{15} itself.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

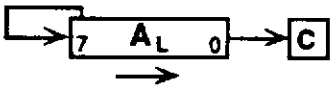
obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	F7					5	13
DP	92	F7					5	—
X1	90	F7					5	—
X2	91	F7					5	—
USP	A1	F7					5	—
SSP	A0	F7					5	—
LRB	A4	F7					5	—
off N_8	B4	N_8	F7				7	16
N_8	B5	N_8	F7				7	—
[DP]	B2	F7					7	15
$\pm N_8$[USP]	B3	N_8	F7				8	16
N_{16}[X1]	B0	N_L	N_H	F7			9	17
N_{16}[X2]	B1	N_L	N_H	F7			9	17

SRAB A

8-bit Arithmetic Right Shift

obj

Function



$C \leftarrow A_0$
$A_n \leftarrow A_{n+1}$ (n=6—0)
$A_7 \leftarrow A_7$

Description

This instruction arithmetically shifts the contents of the lower byte of the accumulator (A_L) one bit to the right.

The overflow from A_0 is entered in the carry (C), while bit A_7 following the shift is entered by A_7 itself.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0. When DD=1, the instruction actually executed becomes "SRA A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD
0

Codes/Cycles

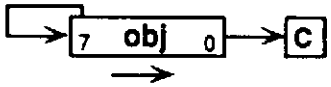
obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A	73						2	—
#N_8								
rN								
PSWH								
PSWL								
off N_8								
N_8								
[DP]								
±N_8[USP]								
N_{16}[X1]								
N_{16}[X2]								

SRAB obj

8-bit Arithmetic Right Shift

obj

Function



$C \leftarrow obj_0$
$obj_n \leftarrow obj_{n+1}$ (n=6—0)
$obj_7 \leftarrow obj_7$

Description

This instruction arithmetically shifts the contents (byte long) of the addressing object (obj) one bit to the right.

The overflow from obj_0 is entered in the carry (C), while the bit obj_7 following the shift is entered by obj_7 itself.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

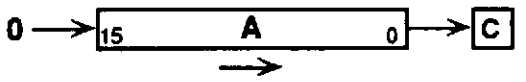
obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_8								
rN	20+N	F7					5	9
PSWH	A2	F7					5	—
PSWL	A3	F7					5	—
off N_8	C4	N_8	F7				7	12
N_8	C5	N_8	F7				7	—
[DP]	C2	F7					7	11
$\pm N_{8[USP]}$	C3	N_8	F7				8	12
N_{16}[X1]	C0	N_L	N_H	F7			9	13
N_{16}[X2]	C1	N_L	N_H	F7			9	13

SRL A

16-bit Right Shift

obj

Function



$C \leftarrow A_0$
$A_n \leftarrow A_{n+1}$ (n=14—0)
$A_{15} \leftarrow 0$

Description

This instruction shifts the contents (word long) of the accumulator one bit to the right.

The overflow from A_0 is entered in the carry (C). Also, 0 (zero) is entered in A_{15}.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1. When DD=0, the instruction actually executed becomes "SRLB A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	63						2	—
#N_{16}								
erN								
DP								
X1								
X2								
USP								
SSP								
LRB								
off N_8								
N_8								
[DP]								
$\pm N_8$[USP]								
N_{16}[X1]								
N_{16}[X2]								

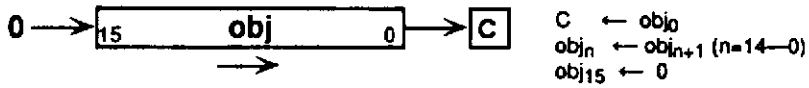
SRL obj

16-bit Right Shift

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function



C ← obj_0
obj_n ← obj_{n+1} (n=14—0)
obj_{15} ← 0

Description

This instruction shifts the contents of the addressing object (word long) (obj) one bit to the right.

The overflow from obj_0 is entered in the carry (C). Also, 0 (zero) is entered in obj_{15}.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD

Codes/Cycles

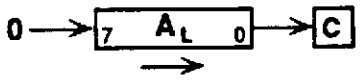
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	E7					5	13
DP	92	E7					5	—
X1	90	E7					5	—
X2	91	E7					5	—
USP	A1	E7					5	—
SSP	A0	E7					5	—
LRB	A4	E7					5	—
off N_8	B4	N_8	E7				7	16
N_8	B5	N_8	E7				7	—
[DP]	B2	E7					7	15
±N_8[USP]	B3	N_8	E7				8	16
N_{16}[X1]	B0	N_L	N_H	E7			9	17
N_{16}[X2]	B1	N_L	N_H	E7			9	17

SRLB A

8-bit Right Shift

obj

Function



$C \leftarrow A_0$
$A_n \leftarrow A_{n+1}$ (n=6—0)
$A_7 \leftarrow 0$

Description

This instruction shifts the contents of the lower byte of the accumulator (A_L) one bit to the right.

The overflow from A_0 is entered in the carry (C) and 0 (zero) is entered in A_7.

This instruction is affected by the data descriptor (DD).
For this instruction to be performed correctly, it is necessary to set DD=0. When DD=1, the instruction actually executed becomes "SRL A".

Flags

Flags affected by execution:

ZF	CF	HC	DD
	•		

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A	63						2	—
#N_8								
rN								
PSWH								
PSWL								
off N_8								
N_8								
[DP]								
±N_8[USP]								
N_{16}[X1]								
N_{16}[X2]								

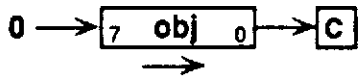
SRLB obj

8-bit Right Shift

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function



$C \leftarrow obj_0$
$obj_n \leftarrow obj_{n+1}$ (n=6—0)
$A_7 \leftarrow 0$

Description

This instruction shifts the contents (byte long) of the addressing object (obj)one bit to the right.

The overflow from obj_0 is entered in the carry (C). Also, 0 (zero) is entered in obj_7.

Flags

Flags affected by execution:

ZF	CF	HC	DD
	*		

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_8								
rN	20+N	E7					5	9
PSWH	A2	E7					5	—
PSWL	A3	E7					5	—
off N_8	C4	N_8	E7				7	12
N_8	C5	N_8	E7				7	—
[DP]	C2	E7					7	11
$\pm N_8$[USP]	C3	N_8	E7				8	12
N_{16}[X1]	C0	N_L	N_H	E7			9	13
N_{16}[X2]	C1	N_L	N_H	E7			9	13

ST A, obj

16-bit Transfer

obj

erN, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

A → obj (word long)

Description

This instruction transfers the contents of the accumulator to the addressing object (word long).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1. When DD=0, the instruction actually executed becomes "STB A, obj".

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	88+N						2	6
DP								
X1								
X2								
USP								
SSP								
LRB								
off N_8	D4	N_8					4	9
N_8	D5	N_8					4	—
[DP]	D2						4	8
$\pm N_8$[USP]	D3	N_8					5	9
N_{16}[X1]	D0	N_L	N_H				6	10
N_{16}[X2]	D1	N_L	N_H				6	10

STB A, obj

8-bit Transfer

obj

rN, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A_L → obj (byte long)

Description

This instruction transfers the contents of the lower byte of accumulator (A_L) to the addressing object (byte long).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0. When DD=1, the instructor actually executed becomes "ST A, obj".

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	88+N						2	4
PSWH								
PSWL								
off N_8	D4	N_8					4	7
N_8	D5	N_8					4	—
[DP]	D2						4	6
±N_8[USP]	D3	N_8					5	7
N_{16}[X1]	D0	N_L	N_H				6	8
N_{16}[X2]	D1	N_L	N_H				6	8

SUB A, obj

16-bit Subtract

obj

#N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A ← A - obj (word long)

Description

This instruction subtracts the contents of the addressing object from the contents of the accumulator in a word long operation. The resulting difference (16 bits) is stored in the accumulator. The borrow that occurs during the subtraction is stored in the carry.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}	A6	N_L	N_H				6	—
erN	28+N						3	7
DP	92	A2					4	—
X1	90	A2					4	—
X2	91	A2					4	—
USP	A1	A2					4	—
SSP	A0	A2					4	—
LRB	A4	A2					4	—
off N_8	A7	N_8					4	9
N_8	B5	N_8	A2				6	—
[DP]	B2	A2					6	10
±N_8[USP]	B3	N_8	A2				7	11
N_{16}[X1]	B0	N_L	N_H	A2			8	12
N_{16}[X2]	B1	N_L	N_H	A2			8	12

SUB obj, A

16-bit Subtract

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - A (word long)

Description

This instruction subtracts the contents of the accumulator from the contents of the addressing object in a word long operation.

The resulting difference (16 bits) is stored in the addressing object. The borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	A1					5	13
DP	92	A1					5	—
X1	90	A1					5	—
X2	91	A1					5	—
USP	A1	A1					5	—
SSP	A0	A1					5	—
LRB	A4	A1					5	—
off N_8	B4	N_8	A1				7	16
N_8	B5	N_8	A1				7	—
[DP]	B2	A1					7	15
$\pm N_8$[USP]	B3	N_8	A1				8	16
N_{16}[X1]	B0	N_L	N_H	A1			9	17
N_{16}[X2]	B1	N_L	N_H	A1			9	17

SUB obj, off N_8

16-bit Subtract

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - off N_8 (word long)

Description

This instruction subtracts the contents of the data memory specified by direct page addressing (off N_8) from the contents of the addressing object in a word long operation.

The resulting difference (16 bits) is stored in the addressing object. The borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	INT EXT	EXT INT	EXT EXT	← obj
							INT	EXT	INT	EXT	← off N_8
A											
#N_{16}											
erN	44+N	A3	N_8				7	—	—	20	
DP	92	A3	N_8				7	16	—	—	
X1	90	A3	N_8				7	16	—	—	
X2	91	A3	N_8				7	16	—	—	
USP	A1	A3	N_8				7	16	—	—	
SSP	A0	A3	N_8				7	16	—	—	
LRB	A4	A3	N_8				7	16	—	—	
off N'_8	B4	N'_8	A3	N_8			9	—	—	23	
N'_8	B5	N'_8	A3	N_8			9	18	—	—	
[DP]	B2	A3	N_8				9	18	13	22	
$\pm N'_8$[USP]	B3	N'_8	A3				10	19	14	23	
N_{16}[X1]	B0	N_L	N_H	A3	N_8		11	20	15	24	
N_{16}[X2]	B1	N_L	N_H	A3	N_8		11	20	15	24	

SUB obj, #N_{16}

16-bit Subtract

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N'_{16}[X1], N'_{16}[X2]

Function

obj ← obj - #N_{16} (word long)

Description

This instruction subtracts the 16 bit immediate value (#N_{16}) from the contents of the addressing object (word long).

The resulting difference (16 bits) is stored in the addressing object. The borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	A0	N_L	N_H			8	17
DP	92	A0	N_L	N_H			8	—
X1	90	A0	N_L	N_H			8	—
X2	91	A0	N_L	N_H			8	—
USP	A1	A0	N_L	N_H			8	—
SSP	A0	A0	N_L	N_H			8	—
LRB	A4	A0	N_L	N_H			8	—
off N_8	B4	N_8	A0	N_L	N_H		10	20
N_8	B5	N_8	A0	N_L	N_H		10	—
[DP]	B2	A0	N_L	N_H			10	19
$\pm N_8$[USP]	B3	N_8	A0	N_L	N_H		11	20
N'_{16}[X1]	B0	N'_L	N'_H	A0	N_L	N_H	12	21
N'_{16}[X2]	B1	N'_L	N'_H	A0	N_L	N_H	12	21

SUBB A, obj

8-bit Subtract

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A_L ← A_L - obj (byte long)

Description

This instruction subtracts the contents (byte long) of the addressing object from the contents of the lower byte of the accumulator (A_L).

The resulting difference (8 bits) is stored in the lower byte of the accumulator (A_L). The borrow that occurs during the subtraction is stored in the carry (C).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8	A6	N_8					4	—
rN	28+N						3	5
PSWH	A2	A2					4	—
PSWL	A3	A2					4	—
off N_8	C4	N_8					4	7
N_8	C5	N_8	A2				6	—
[DP]	C2	A2					6	—
±N_8[USP]	C3	N_8	A2				7	9
N_{16}[X1]	C0	N_L	N_H	A2			8	10
N_{16}[X2]	C1	N_L	N_H	A2			8	10

SUBB obj, A

8-bit Subtract

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - A_L (byte long)

Description

This instruction subtracts the contents of the lower byte of the accumulator (A_L) from the contents (byte long) of the addressing object.

The resulting difference (8 bits) is stored in the addressing object. The borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
*	*	*	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	A1					5	9
PSWH	A2	A1					5	—
PSWL	A3	A1					5	—
off N_8	C4	N_8	A1				7	12
N_8	C5	N_8	A1				7	—
[DP]	C2	A1					7	11
±N_8[USP]	C3	N_8	A1				8	12
N_{16}[X1]	C0	N_L	N_H	A1			9	13
N_{16}[X2]	C1	N_L	N_H	A1			9	13

SUBB obj, off N_8

8-bit Subtract

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - off N_8 (byte long)

Description

This instruction subtracts from the addressing object, the data memory specified by direct page addressing (off N_8).

The resulting difference (8 bits) is stored in the addressing object. The borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES INT	CYCLES INT	CYCLES EXT	CYCLES EXT	← obj
	1	2	3	4	5	6	INT	EXT	INT	EXT	← off N_8
A											
#N_8											
rN	20+N	A3	N_8				7	—	—	14	
PSWH	A2	A3	N_8				7	12	—	—	
PSWL	A3	A3	N_8				7	12	—	—	
off N'_8	C4	N'_8	A3	N_8			9	—	—	17	
N'_8	C5	N'_8	A3	N_8			9	14	—	—	
[DP]	C2	A3	N_8				9	14	11	16	
$\pm N'_8$[USP]	C3	N'_8	A3	N_8			10	15	12	17	
N_{16}[X1]	C0	N_L	N_H	A3	N_8		11	16	13	18	
N_{16}[X2]	C1	N_L	N_H	A3	N_8		11	16	13	18	

SUBB obj, #N_8

8-bit Subtract

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj - #N_8 (byte long)

Description

This instruction subtracts the 8-bit immediate value (#N_8) from the addressing object (byte long).

The resulting difference (8 bits) is stored in the addressing object. The borrow that occurs during the subtraction is stored in the carry (C).

Flags

Flags affected by execution:

ZF	CF	HC	DD
•	•	•	

Flags affecting execution:

DD

Codes/Cycles

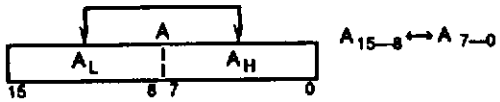
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	tNT	EXT
A								
#N_8								
rN	20+N	A0	N_8				6	11
PSWH	A2	A0	N_8				6	—
PSWL	A3	A0	N_8				6	—
off N'_8	C4	N'_8	A0	N_8			8	14
N'_8	C5	N'_8	A0	N_8			8	—
[DP]	C2	A0	N_8				8	13
$\pm N'_8$[USP]	C3	N'_8	A0	N_8			9	14
N_{16}[X1]	C0	N_L	N_H	A0	N_8		10	15
N_{16}[X2]	C1	N_L	N_H	A0	N_8		10	15

SWAP

16-bit Swap

obj

Function



Description

This instruction exchanges the upper byte (A_L) and the lower byte (A_H) of the accumulator.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1. When DD=0, the instruction actually executed becomes "SWAPB".

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD
1

Codes/Cycles

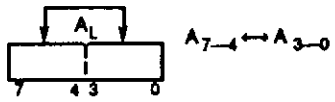
obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	83						2	—

SWAPB

8-bit Swap

obj

Function



Description

This instruction exchanges the upper 4 bits and the lower 4 bits of the lower byte of the accumulator (A_L).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0. When DD=1, the instruction actually executed becomes "SWAP".

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	83						2	—

TBR obj

Test Bit (Register Indirect Bit Addressing)

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

IF obj.bit (A_{2-0}) = 0 THEN Z ← 1
ELSE Z ← 0

Description

This instruction tests the contents of the specified bit of the the addressing object (byte long).

The location of the bit is specified by the contents of the lower 3 bits of the accumulator (A_{2-0}) at the execution time of the instruction.

If the specified bit is 0, the zero flag (Z) is set to 1; otherwise the zero flag is set to 0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	13					4	6
PSWH	A2	13					4	—
PSWL	A3	13					4	—
off N_8	C4	N_8	13				6	9
N_8	C5	N_8	13				6	—
[DP]	C2	13					6	8
$\pm N_8$[USP]	C3	N_8	13				7	9
N_{16}[X1]	C0	N_L	N_H	13			8	10
N_{16}[X2]	C1	N_L	N_H	13			8	10

VCAL table-address

Vector Call

obj

Function

(SSP) ← PC + 1
SSP ← SSP - 2
SF ← 0
PC ← (table-address)

where "table-address" must be an even-numbered address of the VCAL table space (28_H—37_H).

Description

This instruction calls the vector address in the VCAL space.

After the return address is saved in the system stack and SSP is revised, the contents of the ROM at the address specified by "table-address" is stored in the lower 8 bits of the PC, while the contents of the ROM whose address follows the first ROM address is stored in the upper 8 bits of the PC.

This instruction clears the stack flag (SF) by setting it to 0.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	10+n						11	15

[NOTE] The value of n varies with the "table-address" as shown in the following table.

table-address	n	table-address	n
0028_H	0	0030_H	4
$002A_H$	1	0032_H	5
$002C_H$	2	0034_H	6
$002E_H$	3	0036_H	7

XCHG A, obj

16-bit Exchange

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

A ⟷ obj (word long)

Description

This instruction exchanges the contents of the addressing object and the accumulator in a word long operation.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	10					5	13
DP	92	10					5	—
X1	90	10					5	—
X2	91	10					5	—
USP	A1	10					5	—
SSP	A0	10					5	—
LRB	A4	10					5	—
off N_8	B4	N_8	10				7	16
N_8	B5	N_8	10				7	—
[DP]	B2	10					7	15
$\pm N_8$[USP]	B3	N_8	10				8	16
N_{16}[X1]	B0	N_L	N_H	10			9	17
N_{16}[X2]	B1	N_L	N_H	10			9	17

XCHGB A, obj

8-bit Exchange

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

$A_L \longleftrightarrow$ obj (byte long)

Description

This instruction exchanges the contents of the addressing object and the lower byte of the accumulator (A_L) in a byte long operation.

This instruction is affected by the data description (DD).
For this instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	10					5	9
PSWH	A2	10					5	—
PSWL	A3	10					5	—
off N_8	C4	N_8	10				7	12
N_8	C5	N_8	10				7	—
[DP]	C2	10					7	11
$\pm N_8$[USP]	C3	N_8	10				8	12
N_{16}[X1]	C0	N_L	N_H	10			9	13
N_{16}[X2]	C1	N_L	N_H	10			9	13

XNBL A, off N_8

4-bit (Nibble) Exchange

obj

Function

$A_{3—0} \leftrightarrow$ off $N_{8\ 3—0}$

Description

This instruction exchanges the lower 4 bits of the data memory specified by direct addressing (off N_8) and the lower 4 bits of the accumulator.

Flags

Flags affected by execution:

ZF	CF	HC	DD

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
	84	N_8					5	10

XOR A, obj

16-bit Exclusive OR

obj

#N_{16}, erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N_{16}[X1], N_{16}[X2]

Function

A ← A ∀ obj (word long)

Description

This instruction performs an exclusive OR operation on the contents of the accumulator and the addressing object. The result (16 bits) is stored in the accumulator.

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=1.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD
1

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}	F6	N_L	N_H				6	—
erN	68+N						4	8
DP	92	F2					4	—
X1	90	F2					4	—
X2	91	F2					4	—
USP	A1	F2					4	—
SSP	A0	F2					4	—
LRB	A4	F2					4	—
off N_8	F7	N_8					4	9
N_8	B5	N_8	F2				6	—
[DP]	B2	F2					6	10
±N_8[USP]	B3	N_8	F2				7	11
N_{16}[X1]	B0	N_L	N_H	F2			8	12
N_{16}[X2]	B1	N_L	N_H	F2			8	12

XOR obj, A

16-bit Exclusive OR

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∀ A (word long)

Description

This instruction performs a word long exclusive OR operation on the contents of the addressing object and the accumulator.

The result (16 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_{16}								
erN	44+N	F1					5	13
DP	92	F1					5	—
X1	90	F1					5	—
X2	91	F1					5	—
USP	A1	F1					5	—
SSP	A0	F1					5	—
LRB	A4	F1					5	—
off N_8	B4	N_8	F1				7	16
N_8	B5	N_8	F1				7	—
[DP]	B2	F1					7	15
$\pm N_8$[USP]	B3	N_8	F1				8	16
N_{16}[X1]	B0	N_L	N_H	F1			9	17
N_{16}[X2]	B1	N_L	N_H	F1			9	17

XOR obj, off N_8

16-bit Exclusive OR

obj

erN, DP, X1, X2, USP, SSP, LRB, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∀ off N_8 (word long)

Description

This instruction performs a word long exclusive OR operation on the contents of the addressing object and the data memory specified by direct page addressing (off N_8).

The result (16 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT (← obj)	INT (← off N_8)	EXT (← obj)	EXT (← off N_8)
A										
#N_{16}										
erN	44+N	F3	N_8				7	—	—	20
DP	92	F3	N_8				7	16	—	—
X1	90	F3	N_8				7	16	—	—
X2	91	F3	N_8				7	16	—	—
USP	A1	F3	N_8				7	16	—	—
SSP	A0	F3	N_8				7	16	—	—
LRB	A4	F3	N_8				7	16	—	—
off N'_8	B4	N'_8	F3	N_8			9	—	—	23
N'_8	B5	N'_8	F3	N_8			9	18	—	—
[DP]	B2	F3	N_8				9	18	13	22
$\pm N'_8$[USP]	B3	N'_8	F3				10	19	14	23
N_{16}[X1]	B0	N_L	N_H	F3	N_8		11	20	15	24
N_{16}[X2]	B1	N_L	N_H	F3	N_8		11	20	15	24

XOR obj, #N_{16}

16-bit Exclusive OR

obj

erN, DP, X1, X2, USP, SSP, LRB, off N_8, N_8, [DP], ±N_8[USP], N'_{16}[X1], N'_{16}[X2]

Function

obj ← obj ∀ #N_{16} (word long)

Description

This instruction performs an exclusive OR operation on the contents of the addressing object (word long) and the 16-bit immediate value (#N_{16}).

The result is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE 1	2	3	4	5	6	CYCLES INT	EXT
A								
#N_{16}								
erN	44+N	F0	N_L	N_H			8	17
DP	92	F0	N_L	N_H			8	—
X1	90	F0	N_L	N_H			8	—
X2	91	F0	N_L	N_H			8	—
USP	A1	F0	N_L	N_H			8	—
SSP	A0	F0	N_L	N_H			8	—
LRB	A4	F0	N_L	N_H			8	—
off N_8	B4	N_8	F0	N_L	N_H		10	20
N_8	B5	N_8	F0	N_L	N_H		10	—
[DP]	B2	F0	N_L	N_H			10	19
±N_8[USP]	B3	N_8	F0	N_L	N_H		11	20
N'_{16}[X1]	B0	N'_L	N'_H	F0	N_L	N_H	12	21
N'_{16}[X2]	B1	N'_L	N'_H	F0	N_L	N_H	12	21

XORB A, obj

8-bit Exclusive OR

obj

#N_8, rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

$A_L \leftarrow A_L \forall$ obj (byte long)

Description

This instruction performs an exclusive OR operation on the contents of the lower byte of the accumulator (A_L) and the addressing object (byte long).

The result (8 bits) is stored in the lower byte of the accumulator (A_L).

This instruction is affected by the data descriptor (DD).
For this instruction to be executed correctly, it is necessary to set DD=0.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD
0

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_8	F6	N_8					4	—
rN	20+N	F2					4	6
PSWH	A2	F2					4	—
PSWL	A3	F2					4	—
off N_8	F7	N_8					4	7
N_8	C5	N_8	F2				6	—
[DP]	C2	F2					6	—
$\pm N_8$[USP]	C3	N_8	F2				7	9
N_{16}[X1]	C0	N_L	N_H	F2			8	10
N_{16}[X2]	C1	N_L	N_H	F2			8	10

XORB obj, A

8-bit Exclusive OR

obj

rN, PSWH, PSWL, off N_8, N_8, [DP], $\pm N_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∀ A_L (byte long)

Description

This instruction performs an exclusive OR operation on the contents of the lower byte of the accumulator (A_L) and the addressing object (byte long).

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES	
	BYTE							
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N						5	9
PSWH	A2	F1					5	—
PSWL	A3	F1					5	—
off N_8	C4	N_8	F1				7	12
N_8	C5	N_8	F1				7	—
[DP]	C2	F1					7	11
$\pm N_8$[USP]	C3	N_8	F1				8	12
N_{16}[X1]	C0	N_L	N_H	F1			9	13
N_{16}[X2]	C1	N_L	N_H	F1			9	13

XORB obj, off N_8

8-bit Exclusive OR

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∀ off N_8 (byte long)

Description

This instruction performs a byte long exclusive OR operation on the contents of the addressing object and the data memory specified by direct page addressing (off N_8).

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
•			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE						CYCLES				← obj
	BYTE						INT	INT	EXT	EXT	
	1	2	3	4	5	6	INT	EXT	INT	EXT	← off N_8
A											
#N_8											
rN	20+N	F3	N_8				7	—	—	14	
PSWH	A2	F3	N_8				7	12	—	—	
PSWL	A3	F3	N_8				7	12	—	—	
off N'_8	C4	N'_8	F3	N_8			9	—	—	17	
N'_8	C5	N'_8	F3	N_8			9	14	—	—	
[DP]	C2	F3	N_8				9	14	11	16	
$\pm N'_8$[USP]	C3	N'_8	F3	N_8			10	15	12	17	
N_{16}[X1]	C0	N_L	N_H	F3	N_8		11	16	13	18	
N_{16}[X2]	C1	N_L	N_H	F3	N_8		11	16	13	18	

XORB obj, #N_8

8-bit Exclusive OR

obj

rN, PSWH, PSWL, off N'_8, N'_8, [DP], $\pm N'_8$[USP], N_{16}[X1], N_{16}[X2]

Function

obj ← obj ∀ #N_8 (byte long)

Description

This instruction performs an exclusive OR operation on the contents (byte long) of the addressing object and the 8-bit immediate value (#N_8).

The result (8 bits) is stored in the addressing object.

Flags

Flags affected by execution:

ZF	CF	HC	DD
*			

Flags affecting execution:

DD

Codes/Cycles

obj	CODE BYTE						CYCLES	
	1	2	3	4	5	6	INT	EXT
A								
#N_8								
rN	20+N	F0	N_8				8	11
PSWH	A2	F0	N_8				6	—
PSWL	A3	F0	N_8				6	—
off N'_8	C4	N'_8	F0	N_8			8	14
N'_8	C5	N'_8	F0	N_8			8	—
[DP]	C2	F0	N_8				8	13
$\pm N'_8$[USP]	C3	N'_8	F0	N_8			9	14
N_{16}[X1]	C0	N_L	N_H	F0	N_8		10	15
N_{16}[X2]	C1	N_L	N_H	F0	N_8		10	15

TABLE 3-1. Data Transfer Instructions (1)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1	L A, #N_{16}	A ← #N_{16}	6	—			3	•			1	70
	L A, erN	A ← erN	2	6			1	•			1	70
	L A, DP	A ← DP	2	—			1	•			1	70
	L A, X1	A ← X1	2	—			1	•			1	70
	L A, X2	A ← X2	2	—			1	•			1	70
	L A, off N_8	A ← off N_8	4	9			2	•			1	70
	L A, N_8	A ← N_8	4	—			2	•			1	70
	L A, [DP]	A ← [DP]	4	8			1	•			1	70
	L A, ±N_8[USP]	A ← ±N_8[USP]	5	9			2	•			1	70
	L A, N_{16}[X1]	A ← N_{16}[X1]	6	10			3	•			1	70
	L A, N_{16}[X2]	A ← N_{16}[X2]	6	10			3	•			1	70
12	LB A, #N_8	A_L ← #N_8	4	—			2	•			0	71
	LB A, rN	A_L ← rN	2	4			1	•			0	71
	LB A, off N_8	A_L ← off N_8	4	7			2	•			0	71
	LB A, N_8	A_L ← N_8	4	—			2	•			0	71
	LB A, [DP]	A_L ← [DP]	4	6			1	•			0	71
	LB A, ±N_8[USP]	A_L ← ±N_8[USP]	5	7			2	•			0	71
	LB A, N_{16}[X1]	A_L ← N_{16}[X1]	6	8			3	•			0	71
	LB A, N_{16}[X2]	A_L ← N_{16}[X2]	6	8			3	•			0	71
20	ST A, erN	A → erN	2	6			1					155
	ST A, off N_8	A → off N_8	4	9			2					155
	ST A, N_8	A → N_8	4	—			2					155
	ST A, [DP]	A → [DP]	4	8			1					155
	ST A, ±N_8[USP]	A → ±N_8[USP]	5	9			2					155
	ST A, N_{16}[X1]	A → N_{16}[X1]	6	10			3					155
	ST A, N_{16}[X2]	A → N_{16}[X2]	6	10			3					155
27	STB A, rN	A_L → rN	2	4			1					156
	STB A, off N_8	A_L → off N_8	4	7			2					156
	STB A, N_8	A_L → N_8	4	—			2					156
	STB A, [DP]	A_L → [DP]	4	6			1					156
	STB A, ±N_8[USP]	A_L → ±N_8[USP]	5	7			2					156
	STB A, N_{16}[X1]	A_L → N_{16}[X1]	6	8			3					156
	STB A, N_{16}[X2]	A_L → N_{16}[X2]	6	8			3					156
34	MOV A, erN	A ← erN	4	8			2				1	82
	MOV A, DP	A ← DP	4	—			2				1	82
	MOV A, X1	A ← X1	4	—			2				1	82
	MOV A, X2	A ← X2	4	—			2				1	82
	MOV A, USP	A ← USP	4	—			2				1	82
	MOV A, SSP	A ← SSP	4	—			2				1	82
	MOV A, LRB	A ← LRB	4	—			2				1	82
	MOV A, off N_8	A ← off N_8	6	11			3				1	82
	MOV A, N_8	A ← N_8	6	—			3				1	82
	MOV A, [DP]	A ← [DP]	6	10			2				1	82
	MOV A, ±N_8[USP]	A ← ±N_8[USP]	7	11			3				1	82
	MOV A, N_{16}[X1]	A ← N_{16}[X1]	8	12			4				1	82
	MOV A, N_{16}[X2]	A ← N_{16}[X2]	8	12			4				1	82
47	MOV erN, A	erN ← A	4		12		2					85
	MOV DP, A	DP ← A	2		—		1					85
	MOV X1, A	X1 ← A	2		—		1					85
	MOV X2, A	X2 ← A	2		—		1					85
	MOV USP, A	USP ← A	4		—		2					85
	MOV SSP, A	SSP ← A	4		—		2					85
	MOV LRB, A	LRB ← A	4		—		2					85
	MOV off N_8, A	off N_8 ← A	6		15		3					85
	MOV N_8, A	N_8 ← A	6		—		3					85
	MOV [DP], A	[DP] ← A	6		14		2					85
	MOV ±N_8[USP], A	±N_8[USP] ← A	7		15		3					85
	MOV N_{16}[X1], A	N_{16}[X1] ← A	8		16		4					85
	MOV N_{16}[X2], A	N_{16}[X2] ← A	8		16		4					85

TABLE 3-2. Data Transfer Instructions (2)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
80	MOV erN, #N_{16}	erN ← #N_{16}	8		16		4					87
	MOV DP, #N_{16}	DP ← #N_{16}	6		—		3					87
	MOV X1, #N_{16}	X1 ← #N_{16}	6		—		3					87
	MOV X2, #N_{16}	X2 ← #N_{16}	6		—		3					87
	MOV USP, #N_{16}	USP ← #N_{16}	8		—		4					87
	MOV SSP, #N_{16}	SSP ← #N_{16}	8		—		4					87
	MOV LRB, #N_{16}	LRB ← #N_{16}	6		—		3					87
	MOV off N_8, #N_{16}	off N_8 ← #N_{16}	10		19		5					87
	MOV N_8, #N_{16}	N_8 ← #N_{16}	10		—		5					87
	MOV [DP], #N_{16}	[DP] ← #N_{16}	10		18		4					87
	MOV ±N_8[USP], #N_{16}	±N_8[USP] ← #N_{16}	11		19		5					87
	MOV N'_{16}[X1], #N_{16}	N'_{16}[X1] ← #N_{16}	12		20		6					87
	MOV N'_{16}[X2], #N_{16}	N'_{16}[X2] ← #N_{16}	12		20		6					87
73	MOV erN, A	erN ← A	4	—	—	12	2					84
	MOV erN, #N_{16}	erN ← #N_{16}	8	—	—	16	4					84
	MOV erN, erN'	erN ← erN'	4	—	—	12	2					84
	MOV erN, DP	erN ← DP	4	—	8	—	2					84
	MOV erN, X1	erN ← X1	4	—	8	—	2					84
	MOV erN, X2	erN ← X2	4	—	8	—	2					84
	MOV erN, USP	erN ← USP	4	—	8	—	2					84
	MOV erN, SSP	erN ← SSP	4	—	8	—	2					84
	MOV erN, LRB	erN ← LRB	4	—	8	—	2					84
	MOV erN, off N_8	erN ← off N_8	6	—	—	15	3					84
	MOV erN, N_8	erN ← N_8	6	—	10	—	3					84
	MOV erN, [DP]	erN ← [DP]	6	10	10	14	2					84
	MOV erN, ±N_8[USP]	erN ← ±N_8[USP]	7	11	11	15	3					84
	MOV erN, N_{16}[X1]	erN ← N_{16}[X1]	8	12	12	16	4					84
	MOV erN, N_{16}[X2]	erN ← N_{16}[X2]	8	12	12	16	4					84
88	MOV off N_8, A	off N_8 ← A	6	—	15	—	3					88
	MOV off N_8, #N_{16}	off N_8 ← #N_{16}	10	—	19	—	5					88
	MOV off N_8, erN	off N_8 ← erN	6	—	—	15	3					88
	MOV off N_8, DP	off N_8 ← DP	6	—	11	—	3					88
	MOV off N_8, X1	off N_8 ← X1	6	—	11	—	3					88
	MOV off N_8, X2	off N_8 ← X2	6	—	11	—	3					88
	MOV off N_8, USP	off N_8 ← USP	6	—	11	—	3					88
	MOV off N_8, SSP	off N_8 ← SSP	6	—	11	—	3					88
	MOV off N_8, LRB	off N_8 ← LRB	6	—	11	—	3					88
	MOV off N_8, off N'_8	off N_8 ← off N'_8	8	—	—	18	4					88
	MOV off N_8, N'_8	off N_8 ← N'_8	8	—	13	—	4					88
	MOV off N_8, [DP]	off N_8 ← [DP]	8	12	13	17	3					88
	MOV off N_8, ±N'_8[USP]	off N_8 ← ±N'_8[USP]	9	13	14	18	4					88
	MOV off N_8, N_{16}[X1]	off N_8 ← N_{16}[X1]	10	14	15	19	5					88
	MOV off N_8, N_{16}[X2]	off N_8 ← N_{16}[X2]	10	14	15	19	5					88
103	MOV DP, A	DP ← A	2	—			1					83
	MOV DP, #N_{16}	DP ← #N_{16}	6	—			3					83
	MOV DP, erN	DP ← erN	4	8			2					83
	MOV DP, DP	DP ← DP	4	—			2					83
	MOV DP, X1	DP ← X1	4	—			2					83
	MOV DP, X2	DP ← X2	4	—			2					83
	MOV DP, USP	DP ← USP	4	—			2					83
	MOV DP, SSP	DP ← SSP	4	—			2					83
	MOV DP, LRB	DP ← LRB	4	—			2					83
	MOV DP, off N_8	DP ← off N_8	6	11			3					83
	MOV DP, N_8	DP ← N_8	6	—			3					83
	MOV DP, [DP]	DP ← [DP]	6	10			2					83
	MOV DP, ±N_8[USP]	DP ← ±N_8[USP]	7	11			3					83
	MOV DP, N_{16}[X1]	DP ← N_{16}[X1]	8	12			4					83
	MOV DP, N_{16}[X2]	DP ← N_{16}[X2]	8	12			4					83

TABLE 3-3. Data Transfer Instructions (3)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
118	MOV X1, A	X1 ← A	2	—			1					
	MOV X1, #N_{16}	X1 ← #N_{16}	6	—			3					
	MOV X1, erN	X1 ← erN	4	8			2					
	MOV X1, DP	X1 ← DP	4	—			2					
	MOV X1, X1	X1 ← X1	4	—			2					
	MOV X1, X2	X1 ← X2	4	—			2					
	MOV X1, USP	X1 ← USP	4	—			2					
	MOV X1, SSP	X1 ← SSP	4	—			2					
	MOV X1, LRB	X1 ← LRB	4	—			2					
	MOV X1, off N_8	X1 ← off N_8	6	11			3					
	MOV X1, N_8	X1 ← N_8	6	—			3					
	MOV X1, [DP]	X1 ← [DP]	6	10			2					
	MOV X1, ±N_8[USP]	X1 ← ±N_8[USP]	7	11			3					
	MOV X1, N_{16}[X1]	X1 ← N_{16}[X1]	8	12			4					
	MOV X1, N_{16}[X2]	X1 ← N_{16}[X2]	8	12			4					
133	MOV X2, A	X2 ← A	2	—			1					
	MOV X2, #N_{16}	X2 ← #N_{16}	6	—			3					
	MOV X2, erN	X2 ← erN	4	8			2					
	MOV X2, DP	X2 ← DP	4	—			2					
	MOV X2, X1	X2 ← X1	4	—			2					
	MOV X2, X2	X2 ← X2	4	—			2					
	MOV X2, USP	X2 ← USP	4	—			2					
	MOV X2, SSP	X2 ← SSP	4	—			2					
	MOV X2, LRB	X2 ← LRB	4	—			2					
	MOV X2, off N_8	X2 ← off N_8	6	11			3					
	MOV X2, N_8	X2 ← N_8	6	—			3					
	MOV X2, [DP]	X2 ← [DP]	6	10			2					
	MOV X2, ±N_8[USP]	X2 ← ±N_8[USP]	7	11			3					
	MOV X2, N_{16}[X1]	X2 ← N_{16}[X1]	8	12			4					
	MOV X2, N_{16}[X2]	X2 ← N_{16}[X2]	8	12			4					
148	MOV USP, A	USP ← A	4	—			2					
	MOV USP, #N_{16}	USP ← #N_{16}	8	—			4					
	MOV USP, erN	USP ← erN	4	8			2					
	MOV USP, DP	USP ← DP	4	—			2					
	MOV USP, X1	USP ← X1	4	—			2					
	MOV USP, X2	USP ← X2	4	—			2					
	MOV USP, USP	USP ← USP	4	—			2					
	MOV USP, SSP	USP ← SSP	4	—			2					
	MOV USP, LRB	USP ← LRB	4	—			2					
	MOV USP, off N_8	USP ← off N_8	6	11			3					
	MOV USP, N_8	USP ← N_8	6	—			3					
	MOV USP, [DP]	USP ← [DP]	6	10			2					
	MOV USP, ±N_8[USP]	USP ← ±N_8[USP]	7	11			3					
	MOV USP, N_{16}[X1]	USP ← N_{16}[X1]	8	12			4					
	MOV USP, N_{16}[X2]	USP ← N_{16}[X2]	8	12			4					
163	MOV PSW, erN	PSW ← erN	4	8			2	•	•	•	•	
	MOV PSW, DP	PSW ← DP	4	—			2	•	•	•	•	
	MOV PSW, X1	PSW ← X1	4	—			2	•	•	•	•	
	MOV PSW, X2	PSW ← X2	4	—			2	•	•	•	•	
	MOV PSW, USP	PSW ← USP	4	—			2	•	•	•	•	
	MOV PSW, SSP	PSW ← SSP	4	—			2	•	•	•	•	
	MOV PSW, LRB	PSW ← LRB	4	—			2	•	•	•	•	
	MOV PSW, off N_8	PSW ← off N_8	6	11			3	•	•	•	•	
	MOV PSW, N_8	PSW ← N_8	6	—			3	•	•	•	•	
	MOV PSW, [DP]	PSW ← [DP]	6	10			2	•	•	•	•	
	MOV PSW, ±N_8[USP]	PSW ← ±N_8[USP]	7	11			3	•	•	•	•	
	MOV PSW, N_{16}[X1]	PSW ← N_{16}[X1]	8	12			4	•	•	•	•	
	MOV PSW, N_{16}[X2]	PSW ← N_{16}[X2]	8	12			4	•	•	•	•	

TABLE 3-4. Data Transfer Instructions (4)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
176	MOV SSP, A	SSP ← A	4	—			2					93
	MOV SSP, #N_{16}	SSP ← #N_{16}	6	—			4					93
	MOV SSP, erN	SSP ← erN	4	8			2					93
	MOV SSP, DP	SSP ← DP	4	—			2					93
	MOV SSP, X1	SSP ← X1	4	—			2					93
	MOV SSP, X2	SSP ← X2	4	—			2					93
	MOV SSP, USP	SSP ← USP	4	—			2					93
	MOV SSP, SSP	SSP ← SSP	4	—			2					93
	MOV SSP, LRB	SSP ← LRB	4	—			2					93
	MOV SSP, off N_8	SSP ← off N_8	6	11			3					93
	MOV SSP, N_8	SSP ← N_8	6	—			3					93
	MOV SSP, [DP]	SSP ← [DP]	6	10			2					93
	MOV SSP, ±N_8[USP]	SSP ← ±N_8[USP]	7	11			3					93
	MOV SSP, N_{16}[X1]	SSP ← N_{16}[X1]	8	12			4					93
	MOV SSP, N_{16}[X2]	SSP ← N_{16}[X2]	8	12			4					93
191	MOV LRB, A	LRB ← A	4	—			2					95
	MOV LRB, #N_{16}	LRB ← #N_{16}	6	—			3					95
	MOV LRB, erN	LRB ← erN	4	8			2					95
	MOV LRB, DP	LRB ← DP	4	—			2					95
	MOV LRB, X1	LRB ← X1	4	—			2					95
	MOV LRB, X2	LRB ← X2	4	—			2					95
	MOV LRB, USP	LRB ← USP	4	—			2					95
	MOV LRB, SSP	LRB ← SSP	4	—			2					95
	MOV LRB, LRB	LRB ← LRB	4	—			2					95
	MOV LRB, off N_8	LRB ← off N_8	6	11			3					95
	MOV LRB, N_8	LRB ← N_8	6	—			3					95
	MOV LRB, [DP]	LRB ← [DP]	6	10			2					95
	MOV LRB, ±N_8[USP]	LRB ← ±N_8[USP]	7	11			3					95
	MOV LRB, N_{16}[X1]	LRB ← N_{16}[X1]	8	12			4					95
	MOV LRB, N_{16}[X2]	LRB ← N_{16}[X2]	8	12			4					95
206	MOVB A, rN	A_L ← rN	4	6			2				0	94
	MOVB A, PSWH	A_L ← PSWH	4	—			2				0	94
	MOVB A, PSWL	A_L ← PSWL	4	—			2				0	94
	MOVB A, off N_8	A_L ← off N_8	6	9			3				0	94
	MOVB A, N_8	A_L ← N_8	6	—			3				0	94
	MOVB A, [DP]	A_L ← [DP]	6	8			2				0	94
	MOVB A, ±N_8[USP]	A_L ← ±N_8[USP]	7	9			3				0	94
	MOVB A, N_{16}[X1]	A_L ← N_{16}[X1]	8	10			4				0	94
	MOVB A, N_{16}[X2]	A_L ← N_{16}[X2]	8	10			4				0	94
215	MOVB rN, A	rN ← A_L	4		8		2					95
	MOVB PSWH, A	PSWH ← A_L	4		—		2					95
	MOVB PSWL, A	PSWL ← A_L	4		—		2					95
	MOVB off N_8, A	off N_8 ← A_L	6		11		3					95
	MOVB N_8, A	N_8 ← A_L	6		—		3					95
	MOVB [DP], A	[DP] ← A_L	6		10		2					95
	MOVB ±N_8[USP], A	±N_8[USP] ← A_L	7		11		3					95
	MOVB N_{16}[X1], A	N_{16}[X1] ← A_L	8		12		4					95
	MOVB N_{16}[X2], A	N_{16}[X2] ← A_L	8		12		4					95
224	MOVB rN, #N_8	rN ← #N_8	4	6			2					97
	MOVB PSWH, #N_8	PSWH ← #N_8	6	—			3					97
	MOVB PSWL, #N_8	PSWL ← #N_8	6	—			3					97
	MOVB off N'_8, #N_8	off N'_8 ← #N_8	8	13			4					97
	MOVB N'_8, #N_8	N'_8 ← #N_8	6	—			4					97
	MOVB [DP], #N_8	[DP] ← #N_8	8	12			3					97
	MOVB ±N'_8[USP], #N_8	±N'_8[USP] ← #N_8	9	13			4					97
	MOVB N_{16}[X1], #N_8	N_{16}[X1] ← #N_8	10	14			5					97
	MOVB N_{16}[X2], #N_8	N_{16}[X2] ← #N_8	10	14			5					97

TABLE 3-5. Data Transfer Instructions (5)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
233	MOVB rN, A	rN ← A	4	—	8	—	2					100
	MOVB rN, #N_8	rN ← #N_8	4	—	6	—	2					100
	MOVB rN, rN'	rN ← rN'	4	—	—	8	2					100
	MOVB rN, PSWH	rN ← PSWH	4	—	6	—	2					100
	MOVB rN, PSWL	rN ← PSWL	4	—	6	—	2					100
	MOVB rN, off N_8	rN ← off N_8	6	—	—	11	3					100
	MOVB rN, N_8	rN ← N_8	6	—	8	—	3					100
	MOVB rN, [DP]	rN ← [DP]	6	8	8	10	2					100
	MOVB rN, ±N_8[USP]	rN ← ±N_8[USP]	7	9	9	11	3					100
	MOVB rN, N_{16}[X1]	rN ← N_{16}[X1]	8	10	10	12	4					100
	MOVB rN, N_{16}[X2]	rN ← N_{16}[X2]	8	10	10	12	4					100
244	MOVB off N_8, A	off N_8 ← A	4	—	8	—	2					95
	MOVB off N_8, #N'$_8$	off N_8 ← #N'$_8$	4	—	6	—	2					95
	MOVB off N_8, rN	off N_8 ← rN	4	—	—	8	2					95
	MOVB off N_8, PSWH	off N_8 ← PSWH	4	—	6	—	2					95
	MOVB off N_8, PSWL	off N_8 ← PSWL	4	—	6	—	2					95
	MOVB off N_8, off N'$_8$	off N_8 ← off N'$_8$	6	—	—	11	3					95
	MOVB off N_8, N'$_8$	off N_8 ← N'$_8$	6	—	8	—	3					95
	MOVB off N_8, [DP]	off N_8 ← [DP]	6	8	8	10	2					95
	MOVB off N_8, ±N_8[USP]	off N_8 ← ±N_8[USP]	7	9	9	11	3					95
	MOVB off N_8, N_{16}[X1]	off N_8 ← N_{16}[X1]	8	10	10	12	4					95
	MOVB off N_8, N_{16}[X2]	off N_8 ← N_{16}[X2]	8	10	10	12	4					95
255	MOVB PSWH, A	PSWH ← A	4	—			2	·	·	·	·	'98
	MOVB PSWH, #N_8	PSWH ← #N_8	6	—			3	·	·	·	·	'98
	MOVB PSWH, rN	PSWH ← rN	4	6			2	·	·	·	·	'98
	MOVB PSWH, PSWH	PSWH ← PSWH	4	—			2	·	·	·	·	'98
	MOVB PSWH, PSWL	PSWH ← PSWL	4	—			2	·	·	·	·	'98
	MOVB PSWH, off N_8	PSWH ← off N_8	6	9			3	·	·	·	·	'98
	MOVB PSWH, N_8	PSWH ← N_8	6	—			3	·	·	·	·	'98
	MOVB PSWH, [DP]	PSWH ← [DP]	6	8			2	·	·	·	·	'98
	MOVB PSWH, ±N_8[USP]	PSWH ← ±N_8[USP]	7	9			3	·	·	·	·	'98
	MOVB PSWH, N_{16}[X1]	PSWH ← N_{16}[X1]	8	10			4	·	·	·	·	'98
	MOVB PSWH, N_{16}[X2]	PSWH ← N_{16}[X2]	8	10			4	·	·	·	·	'98
266	MOVB PSWL, A	PSWL ← A	4	—			2					99
	MOVB PSWL, #N_8	PSWL ← #N_8	6	—			3					99
	MOVB PSWL, rN	PSWL ← rN	4	6			2					99
	MOVB PSWL, PSWH	PSWL ← PSWH	4	—			2					99
	MOVB PSWL, PSWL	PSWL ← PSWL	4	—			2					99
	MOVB PSWL, off N_8	PSWL ← off N_8	6	9			3					99
	MOVB PSWL, N_8	PSWL ← N_8	6	—			3					99
	MOVB PSWL, DP]	PSWL ← [DP]	6	8			2					99
	MOVB PSWL, ±N_8[USP]	PSWL ← ±N_8[USP]	7	9			3					99
	MOVB PSWL, N_{16}[X1]	PSWL ← N_{16}[X1]	8	10			4					99
	MOVB PSWL, N_{16}[X2]	PSWL ← N_{16}[X2]	8	10			4					99

TABLE 3-6. Clear and Exchange Instructions

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
277	CLR A	A ← 0 DD ← 1	2				1	1			1	31
278	CLR erN	erN ← 0	4		12		2					32
	CLR DP	DP ← 0	4		—		2					32
	CLR X1	X1 ← 0	4		—		2					32
	CLR X2	X2 ← 0	4		—		2					32
	CLR USP	USP ← 0	4		—		2					32
	CLR SSP	SSP ← 0	4		—		2					32
	CLR LRB	LRB ← 0	4		—		2					32
	CLR off N_8	off N_8 ← 0	6		15		3					32
	CLR N_8	N_8 ← 0	6		—		3					32
	CLR [DP]	[DP] ← 0	6		14		2					32
	CLR ±N_8[USP]	±N_8[USP] ← 0	7		15		3					32
	CLR N_{16}[X1]	N_{16}[X1] ← 0	8		16		4					32
	CLR N_{16}[X2]	N_{16}[X2] ← 0	8		16		4					32
291	CLRB A	A_L ← 0	2				1	1			0	33
292	CLRB rN	rN ← 0	4		8		2					34
	CLRB PSWH	PSWH ← 0	4		—		2					34
	CLRB PSWL	PSWL ← 0	4		—		2					34
	CLRB off N_8	off N_8 ← 0	6		11		3					34
	CLRB N_8	N_8 ← 0	6		—		3					34
	CLRB [DP]	[DP] ← 0	6		10		2					34
	CLRB ±N_8[USP]	±N_8[USP] ← 0	7		11		3					34
	CLRB N_{16}[X1]	N_{16}[X1] ← 0	8		12		4					34
	CLRB N_{16}[X2]	N_{16}[X2] ← 0	8		12		4					34
301	SWAP	A_{15-8} ↔ A_{7-0}	2				1					165
	SWAPB	A_{7-4} ↔ A_{3-0}	2				1					166
303	XCHG A, erN	A ↔ erN	5	13			2					169
	XCHG A, DP	A ↔ DP	5	—			2					169
	XCHG A, X1	A ↔ X1	5	—			2					169
	XCHG A, X2	A ↔ X2	5	—			2					169
	XCHG A, USP	A ↔ USP	5	—			2					169
	XCHG A, SSP	A ↔ SSP	5	—			2					169
	XCHG A, LRB	A ↔ LRB	5	—			2					169
	XCHG A, off N_8	A ↔ off N_8	7	16			3					169
	XCHG A, N_8	A ↔ N_8	7	—			3					169
	XCHG A, [DP]	A ↔ [DP]	7	15			2					169
	XCHG A, ±N_8[USP]	A ↔ ±N_8[USP]	8	16			3					169
	XCHG A, N_{16}[X1]	A ↔ N_{16}[X1]	9	17			4					169
	XCHG A, N_{16}[X2]	A ↔ N_{16}[X2]	9	17			4					169
316	XCHGB A, rN	A_L ↔ rN	5	9			2					170
	XCHGB A, PSWH	A ↔ PSWH	5	—			2					170
	XCHGB A, PSWL	A ↔ PSWL	5	—			2					170
	XCHGB A, off N_8	A ↔ off N_8	7	12			3					170
	XCHGB A, N_8	A ↔ N_8	7	—			3					170
	XCHGB A, [DP]	A ↔ [DP]	7	11			2					170
	XCHGB A, ±N_8[USP]	A ↔ ±N_8[USP]	8	12			3					170
	XCHGB A, N_{16}[X1]	A ↔ N_{16}[X1]	9	13			4					170
	XCHGB A, N_{16}[X2]	A ↔ N_{16}[X2]	9	13			4					170
325	XNBL A, off N_8	A_{3-0} ↔ off $N_{8\,3-0}$	5	10			2					200

TABLE 3-7. Stack Operation Instructions

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
326	PUSHS A PUSHS LRB	(SSP) ← obj (word) SSP ← USP - 2	3 3		7 7		1 1					114
328	POPS A	SSP ← SSP + 2 A ← (SSP) (word) DD ← 1	4	8			1				1	112
329	POPS LRB	SSP ← SSP + 2 LRB ← (SSP)	4	8			1					113

TABLE 3-8. Rotate Instructions

*1 - the first operand
*2 - the second operand

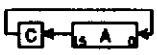
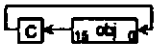
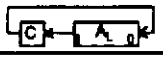
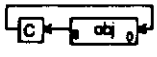
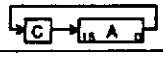
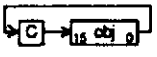
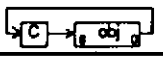
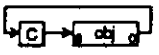
No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
330	ROL A		2				1		•			118
331	ROL erN		5		13		2		•			119
	ROL DP		5		—		2		•			119
	ROL X1		5		—		2		•			119
	ROL X2		5		—		2		•			119
	ROL USP		5		—		2		•			119
	ROL SSP		5		—		2		•			119
	ROL LRB		5		—		2		•			119
	ROL off N_8		7		16		3		•			119
	ROL N_8		7		—		3		•			119
	ROL [DP]		7		15		2		•			119
	ROL $\pm N_8$[USP]		8		16		3		•			119
	ROL N_{16}[X1]		9		17		4		•			119
	ROL N_{16}[X2]		9		17		4		•			119
344	ROLB A		2				1		•			120
345	ROLB rN		5		9		2		•			121
	ROLB PSWH		5		—		2		•			121
	ROLB PSWL		5		—		2		•			121
	ROLB off N_8		7		12		3		•			121
	ROLB N_8		7		—		3		•			121
	ROLB [DP]		7		11		2		•			121
	ROLB $\pm N_8$[USP]		8		12		3		•			121
	ROLB N_{16}[X1]		9		13		4		•			121
	ROLB N_{16}[X2]		9		13		4		•			121
354	ROR A		2				1		•			122
368	ROR erN		5		13		2		•			123
	ROR DP		5		—		2		•			123
	ROR X1		5		—		2		•			123
	ROR X2		5		—		2		•			123
	ROR USP		5		—		2		•			123
	ROR SSP		5		—		2		•			123
	ROR LRB		5		—		2		•			123
	ROR off N_8		7		16		3		•			123
	ROR N_8		7		—		3		•			123
	ROR [DP]		7		15		2		•			123
	ROR $\pm N_8$[USP]		8		16		3		•			123
	ROR N_{16}[X1]		9		17		4		•			123
	ROR N_{16}[X2]		9		17		4		•			123
368	RORB A		2				1		•			124
369	RORB rN		5		9		2		•			125
	RORB PSWH		5		—		2		•			125
	RORB PSWL		5		—		2		•			125
	RORB off N_8		7		12		3		•			125
	RORB N_8		7		—		3		•			125
	RORB [DP]		7		11		2		•			125
	RORB $\pm N_8$[USP]		8		12		3		•			125
	RORB N_{16}[X1]		9		13		4		•			125
	RORB N_{16}[X2]		9		13		4		•			125

TABLE 3-9. Shift Instructions (1)

*1 - the first operand
*2 - the second operand

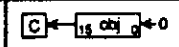
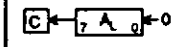
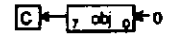
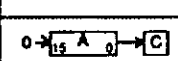
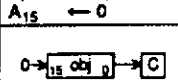
No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
378	SLL A	 C ← A_{15} A_n ← A_{n-1} (n = 15—1) A_0 ← 0	2				1		*			143
379	SLL erN	 C ← obj_{15} obj_n ← obj_{n-1} (n = 15—1) obj_0 ← 0	5		13		2		*			144
	SLL DP		5		—		2		*			144
	SLL X1		5		—		2		*			144
	SLL X2		5		—		2		*			144
	SLL USP		5		—		2		*			144
	SLL SSP		5		—		2		*			144
	SLL LRB		5		—		2		*			144
	SLL off N_8		7		16		3		*			144
	SLL N_8		7		—		3		*			144
	SLL [DP]		7		15		2		*			144
	SLL ±N_8[USP]		8		16		3		*			144
	SLL N_{16}[X1]		9		17		4		*			144
	SLL N_{16}[X2]		9		17		4		*			144
392	SLLB A	 C ← A_7 A_n ← A_{n-1} (n = 7—1) A_0 ← 0	2				1		*			145
393	SLLB rN	 C ← obj_7 obj_n ← obj_{n-1} (n = 7—1) obj_0 ← 0	5		9		2		*			146
	SLLB PSWH		5		—		2		*			146
	SLLB PSWL		5		—		2		*			146
	SLLB off N_8		7		12		3		*			146
	SLLB N_8		7		—		3		*			146
	SLLB [DP]		7		11		2		*			146
	SLLB ±N_8[USP]		8		12		3		*			146
	SLLB N_{16}[X1]		9		13		4		*			146
	SLLB N_{16}[X2]		9		13		4		*			146
402	SRL A	 C ← A_0 A_n ← A_{n+1} (n = 14—0) A_{15} ← 0	2				1		*			151
403	SRL erN	 C ← obj_0 obj_n ← obj_{n+1} (n = 14—0) obj_{15} ← 0	5		13		2		*			152
	SRL DP		5		—		2		*			152
	SRL X1		5		—		2		*			152
	SRL X2		5		—		2		*			152
	SRL USP		5		—		2		*			152
	SRL SSP		5		—		2		*			152
	SRL LRB		5		—		2		*			152
	SRL off N_8		7		16		3		*			152
	SRL N_8		7		—		3		*			152
	SRL [DP]		7		15		2		*			152
	SRL ±N_8[USP]		8		16		3		*			152
	SRL N_{16}[X1]		9		17		4		*			152
	SRL N_{16}[X2]		9		17		4		*			152

TABLE 3-10. Shift Instructions (2)

*1 - the first operand
*2 - the second operand

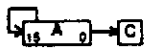
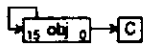
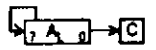
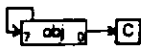
No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
416	SRLB A	 C ← A_0 A_n ← A_{n+1} (n = 6—0) A_7 ← 0	2				1		•			153
417	SRLB rN SRLB PSWH SRLB PSWL SRLB off N_8 SRLB N_8 SRLB [DP] SRLB ±N_8[USP] SRLB N_{16}[X1] SRLB N_{16}[X2]	 obj_{in} ← obj_{in+1} (n = 6—0) obj_7 ← 0	5 5 5 7 7 7 8 9 9		9 — — 12 — 11 12 13 13		2 2 2 3 3 2 3 4 4		• • • • • • • • •			154 154 154 154 154 154 154 154 154
426	SRA A	 C ← A_0 A_n ← A_{n+1} (n = 14—0) A_{15} ← A_{15}	2				1		•			151
427	SRA rN SRA DP SRA X1 SRA X2 SRA USP SRA SSP SRA LRB SRA off N_8 SRA N_8 SRA [DP] SRA ±N_8[USP] SRA N_{16}[X1] SRA N_{16}[X2]	 C ← obj_0 obj_{in} ← obj_{in+1} (n = 14—0) obj_{15} ← obj_{15}	5 5 5 5 5 5 5 7 7 7 8 9 9		13 — — — — — — 16 — 15 16 17 17		2 2 2 2 2 2 2 3 3 2 3 4 4		• • • • • • • • • • • • •			148 148 148 148 148 148 148 148 148 148 148 148 148
440	SRAB A	 C ← A_0 A_n ← A_{n+1} (n = 6—0) A_7 ← A_7	2				1		•			149
441	SRAB rN SRAB PSWH SRAB PSWL SRAB off N_8 SRAB N_8 SRAB [DP] SRAB ±N_8[USP] SRAB N_{16}[X1] SRAB N_{16}[X2]	 C ← obj_0 obj_{in} ← obj_{in+1} (n = 6—0) obj_7 ← obj_7	5 5 5 7 7 7 8 9 9		9 — — 12 — 11 12 13 13		2 2 2 3 3 2 3 4 4		• • • • • • • • •			150 150 150 150 150 150 150 150 150

TABLE 3-11. Increment/Decrement Instructions

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	OD	Page
450	INC erN	erN ← erN + 1	5		13		2	•		•		60
	INC DP	DP ← DP + 1	3		—		1	•		•		60
	INC X1	X1 ← X1 + 1	3		—		1	•		•		60
	INC X2	X2 ← X2 + 1	3		—		1	•		•		60
	INC USP	USP ← USP + 1	5		—		2	•		•		60
	INC SSP	SSP ← SSP + 1	5		—		2	•		•		60
	INC LRB	LRB ← LRB + 1	3		—		1	•		•		60
	INC off N_8	off N_8 ← off N_8 + 1	7		16		3	•		•		60
	INC N_8	N_8 ← N_8 + 1	7		—		3	•		•		60
	INC [DP]	[DP] ← [DP] + 1	7		15		2	•		•		60
	INC ±N_8[USP]	±N_8[USP] ← ±N_8[USP] + 1	8		16		3	•		•		60
	INC N_{16}[X1]	N_{16}[X1] ← N_{16}[X1] + 1	9		17		4	•		•		60
	INC N_{16}[X2]	N_{16}[X2] ← N_{16}[X2] + 1	9		17		4	•		•		60
463	INCB rN	rN ← rN + 1	3		7		1	•		•		61
	INCB PSWH	PSWH ← PSWH + 1	5		—		2	•		•		61
	INCB PSWL	PSWL ← PSWL + 1	5		—		2	•		•		61
	INCB off N_8	off N_8 ← off N_8 + 1	7		12		3	•		•		61
	INCB N_8	N_8 ← N_8 + 1	7		—		3	•		•		61
	INCB [DP]	[DP] ← [DP] + 1	7		11		2	•		•		61
	INCB ±N_8[USP]	±N_8[USP] ← ±N_8[USP] + 1	8		12		3	•		•		61
	INCB N_{16}[X1]	N_{16}[X1] ← N_{16}[X1] + 1	9		13		4	•		•		61
	INCB N_{16}[X2]	N_{16}[X2] ← N_{16}[X2] + 1	9		13		4	•		•		61
472	DEC erN	erN ← erN + 1	5		13		2	•		•		65
	DEC DP	DP ← DP + 1	3		—		1	•		•		65
	DEC X1	X1 ← X1 + 1	3		—		1	•		•		65
	DEC X2	X2 ← X2 + 1	3		—		1	•		•		65
	DEC USP	USP ← USP + 1	5		—		2	•		•		65
	DEC SSP	SSP ← SSP + 1	5		—		2	•		•		65
	DEC LRB	LRB ← LRB + 1	3		—		1	•		•		65
	DEC off N_8	off N_8 ← off N_8 + 1	7		16		3	•		•		65
	DEC N_8	N_8 ← N_8 + 1	7		—		3	•		•		65
	DEC [DP]	[DP] ← [DP] + 1	7		15		2	•		•		65
	DEC ±N_8[USP]	±N_8[USP] ← ±N_8[USP] + 1	8		16		3	•		•		65
	DEC N_{16}[X1]	N_{16}[X1] ← N_{16}[X1] + 1	9		17		4	•		•		65
	DEC N_{16}[X2]	N_{16}[X2] ← N_{16}[X2] + 1	9		17		4	•		•		65
485	DECB rN	rN ← rN + 1	3		7		1	•		•		55
	DECB PSWH	PSWH ← PSWH + 1	5		—		2	•		•		55
	DECB PSWL	PSWL ← PSWL + 1	5		—		2	•		•		55
	DECB off N_8	off N_8 ← off N_8 + 1	7		12		3	•		•		55
	DECB N_8	N_8 ← N_8 + 1	7		—		3	•		•		55
	DECB [DP]	[DP] ← [DP] + 1	7		11		2	•		•		55
	DECB ±N_8[USP]	±N_8[USP] ← ±N_8[USP] + 1	8		12		3	•		•		55
	DECB N_{16}[X1]	N_{16}[X1] ← N_{16}[X1] + 1	9		13		4	•		•		55
	DECB N_{16}[X2]	N_{16}[X2] ← N_{16}[X2] + 1	9		13		4	•		•		55

TABLE 3-12. ROM Table Reference Instructions (1)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
494	LC A, N'$_{16}$	A ← N'$_{16}$ (word)	15		15		4	•				72
495	LC A, [erN]	A ← [erN]	11	15	11	15	2	•				73
	LC A, [DP]	A ← [DP]	11	—	11	—	2	•				73
	LC A, [X1]	A ← [X1]	11	—	11	—	2	•				73
	LC A, [X2]	A ← [X2]	11	—	11	—	2	•				73
	LC A, [USP]	A ← [USP]	11	—	11	—	2	•				73
	LC A, [SSP]	A ← [SSP]	11	—	11	—	2	•				73
	LC A, [LRB]	A ← [LRB]	11	—	11	—	2	•				73
	LC A, [off N$_8$]	A ← [off N$_8$]	13	18	13	18	3	•				73
	LC A, [N$_8$]	A ← [N$_8$]	13	—	13	—	3	•				73
	LC A, [[DP]]	A ← [[DP]]	13	17	13	17	2	•				73
	LC A, [±N$_8$[USP]]	A ← [±N$_8$[USP]]	14	18	14	18	3	•				73
	LC A, [N$_{16}$[X1]]	A ← [N$_{16}$[X1]]	15	19	15	19	4	•				73
	LC A, [N$_{16}$[X2]]	A ← [N$_{16}$[X2]]	15	19	15	19	4	•				73
508	LC A, N'$_{16}$[DP]	A ← N'$_{16}$[DP]	15	—	15	—	4	•				74
	LC A, N'$_{16}$[X1]	A ← N'$_{16}$[X1]	15	—	15	—	4	•				74
	LC A, N'$_{16}$[X2]	A ← N'$_{16}$[X2]	15	—	15	—	4	•				74
	LC A, N'$_{16}$[USP]	A ← N'$_{16}$[USP]	15	—	15	—	4	•				74
	LC A, N'$_{16}$[off N$_8$]	A ← N'$_{16}$[off N$_8$]	17	22	17	22	5	•				74
	LC A, N'$_{16}$[N$_8$]	A ← N'$_{16}$[N$_8$]	17	—	17	—	5	•				74
514	LCB A, N'$_{16}$	A$_L$ ← N'$_{16}$ (byte)	13		13		4	•				82
515	LCB A, [erN]	A$_L$ ← [erN]	9	11	9	11	2	•				76
	LCB A, [DP]	A$_L$ ← [DP]	9	—	9	—	2	•				76
	LCB A, [X1]	A$_L$ ← [X1]	9	—	9	—	2	•				76
	LCB A, [X2]	A$_L$ ← [X2]	9	—	9	—	2	•				76
	LCB A, [USP]	A$_L$ ← [USP]	9	—	9	—	2	•				76
	LCB A, [SSP]	A$_L$ ← [SSP]	9	—	9	—	2	•				76
	LCB A, [LRB]	A$_L$ ← [LRB]	9	—	9	—	2	•				76
	LCB A, [off N$_8$]	A$_L$ ← [off N$_8$]	11	16	11	16	3	•				76
	LCB A, [N$_8$]	A$_L$ ← [N$_8$]	11	—	11	—	3	•				76
	LCB A, [[DP]]	A$_L$ ← [[DP]]	11	15	11	15	2	•				76
	LCB A, [±N$_8$[USP]]	A$_L$ ← [±N$_8$[USP]]	12	16	12	16	3	•				76
	LCB A, [N$_{16}$[X1]]	A$_L$ ← [N$_{16}$[X1]]	13	17	13	17	4	•				76
	LCB A, [N$_{16}$[X2]]	A$_L$ ← [N$_{16}$[X2]]	13	17	13	17	4	•				76
528	LCB A, N'$_{16}$[DP]	A$_L$ ← N'$_{16}$[DP]	13	—	13	—	4	•				77
	LCB A, N'$_{16}$[X1]	A$_L$ ← N'$_{16}$[X1]	13	—	13	—	4	•				77
	LCB A, N'$_{16}$[X2]	A$_L$ ← N'$_{16}$[X2]	13	—	13	—	4	•				77
	LCB A, N'$_{16}$[USP]	A$_L$ ← N'$_{16}$[USP]	13	—	13	—	4	•				77
	LCB A, N'$_{16}$[off N$_8$]	A$_L$ ← N'$_{16}$[off N$_8$]	17	22	17	22	5	•				77
	LCB A, N'$_{16}$[N$_8$]	A$_L$ ← N'$_{16}$[N$_8$]	17	—	17	—	5	•				77
534	CMPC A, N'$_{16}$	A - N'$_{16}$ (word)	15		15		4	•	•			50
535	CMPC A, [erN]	A - [erN]	11	15	11	15	2	•	•			44
	CMPC A, [DP]	A - [DP]	11	—	11	—	2	•	•			44
	CMPC A, [X1]	A - [X1]	11	—	11	—	2	•	•			44
	CMPC A, [X2]	A - [X2]	11	—	11	—	2	•	•			44
	CMPC A, [USP]	A - [USP]	11	—	11	—	2	•	•			44
	CMPC A, [SSP]	A - [SSP]	11	—	11	—	2	•	•			44
	CMPC A, [LRB]	A - [LRB]	11	—	11	—	2	•	•			44
	CMPC A, [off N$_8$]	A - [off N$_8$]	13	18	13	18	3	•	•			44
	CMPC A, [N$_8$]	A - [N$_8$]	13	—	13	—	3	•	•			44
	CMPC A, [[DP]]	A - [[DP]]	13	17	13	17	2	•	•			44
	CMPC A, [±N$_8$[USP]]	A - [±N$_8$[USP]]	14	18	14	18	3	•	•			44
	CMPC A, [N$_{16}$[X1]]	A - [N$_{16}$[X1]]	15	19	15	19	4	•	•			44
	CMPC A, [N$_{16}$[X2]]	A - [N$_{16}$[X2]]	15	19	15	19	4	•	•			44

TABLE 3-13. ROM Table Reference Instructions (2)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
548	CMPC A, N^*_{16}[DP]	A - N^*_{16}[DP]	15	—	15	—	4	•	•			45
	CMPC A, N^*_{16}[X1]	A - N^*_{16}[X1]	15	—	15	—	4	•	•			45
	CMPC A, N^*_{16}[X2]	A - N^*_{16}[X2]	15	—	15	—	4	•	•			45
	CMPC A, N^*_{16}[USP]	A - N^*_{16}[USP]	15	—	15	—	4	•	•			45
	CMPC A, N^*_{16}[off N_8]	A - N^*_{16}[off N_8]	17	22	17	22	5	•	•			45
	CMPC A, N^*_{16}[N_8]	A - N^*_{16}[N_8]	17	—	17	—	5	•	•			45
554	CMPCB A, N^*_{16}	A_L - N^*_{16} (byte)	13		13		4	•	•			45
555	CMPCB A, [erN]	A_L - [erN]	9	11	9	11	2	•	•			47
	CMPCB A, [DP]	A_L - [DP]	9	—	9	—	2	•	•			47
	CMPCB A, [X1]	A_L - [X1]	9	—	9	—	2	•	•			47
	CMPCB A, [X2]	A_L - [X2]	9	—	9	—	2	•	•			47
	CMPCB A, [USP]	A_L - [USP]	9	—	9	—	2	•	•			47
	CMPCB A, [SSP]	A_L - [SSP]	9	—	9	—	2	•	•			47
	CMPCB A, [LRB]	A_L - [LRB]	9	—	9	—	2	•	•			47
	CMPCB A, [off N_8]	A_L - [off N_8]	11	16	11	16	3	•	•			47
	CMPCB A, [N_8]	A_L - [N_8]	11	—	11	—	3	•	•			47
	CMPCB A, [[DP]]	A_L - [[DP]]	11	15	11	15	2	•	•			47
	CMPCB A, [±N_8[USP]]	A_L - [±N_8[USP]]	12	16	12	16	3	•	•			47
	CMPCB A, [N_{16}[X1]]	A_L - [N_{16}[X1]]	13	17	13	17	4	•	•			47
	CMPCB A, [N_{16}[X2]]	A_L - [N_{16}[X2]]	13	17	13	17	4	•	•			47
568	CMPCB A, N^*_{16}[DP]	A_L - N^*_{16}[DP]	13	—	13	—	4	•	•			48
	CMPCB A, N^*_{16}[X1]	A_L - N^*_{16}[X1]	13	—	13	—	4	•	•			48
	CMPCB A, N^*_{16}[X2]	A_L - N^*_{16}[X2]	13	—	13	—	4	•	•			48
	CMPCB A, N^*_{16}[USP]	A_L - N^*_{16}[USP]	13	—	13	—	4	•	•			48
	CMPCB A, N^*_{16}[off N_8]	A_L - N^*_{16}[off N_8]	17	22	17	22	5	•	•			48
	CMPCB A, N^*_{16}[N_8]	A_L - N^*_{16}[N_8]	17	—	17	—	5	•	•			48

TABLE 3-14. Arithmetic Calculation Instructions (1)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
574	MUL	(er1,A) ← A x er0	27		35		2	•				101
575	MULB	A ← A_L x r0	19		21		2	•				102
576	DIV	(er0,A) ← (er0,A) ÷ er2 er1 ← (er0,A) MOD er2	40		63		2	•	•			57
577	DIVB	A ← A ÷ r0 r1 ← A MOD r0	29		33		2	•	•			58
578	ADD A, #N_{16}	A ← A + #N_{16}	6	—			3	•	•	•		13
	ADD A, erN	A ← A + erN	3	7			1	•	•	•		13
	ADD A, DP	A ← A + DP	4	—			2	•	•	•		13
	ADD A, X1	A ← A + X1	4	—			2	•	•	•		13
	ADD A, X2	A ← A + X2	4	—			2	•	•	•		13
	ADD A, USP	A ← A + USP	4	—			2	•	•	•		13
	ADD A, SSP	A ← A + SSP	4	—			2	•	•	•		13
	ADD A, LRB	A ← A + LRB	4	—			2	•	•	•		13
	ADD A, off N_8	A ← A + off N_8	4	9			2	•	•	•		13
	ADD A, N_8	A ← A + N_8	6	—			3	•	•	•		13
	ADD A, [DP]	A ← A + [DP]	6	10			2	•	•	•		13
	ADD A, ±N_8[USP]	A ← A + ±N_8[USP]	7	11			3	•	•	•		13
	ADD A, N_{16}[X1]	A ← A + N_{16}[X1]	8	12			4	•	•	•		13
	ADD A, N_{16}[X2]	A ← A + N_{16}[X2]	8	12			4	•	•	•		13
605	ADD erN, off N_8	erN ← erN + off N_8	7	—	—	20	3	•	•	•		14
	ADD DP, off N_8	DP ← DP + off N_8	7	13	—	—	3	•	•	•		14
	ADD X1, off N_8	X1 ← X1 + off N_8	7	13	—	—	3	•	•	•		14
	ADD X2, off N_8	X2 ← X2 + off N_8	7	13	—	—	3	•	•	•		14
	ADD USP, off N_8	USP ← USP + off N_8	7	13	—	—	3	•	•	•		14
	ADD SSP, off N_8	SSP ← SSP + off N_8	7	13	—	—	3	•	•	•		14
	ADD LRB, off N_8	LRB ← LRB + off N_8	7	13	—	—	3	•	•	•		14
	ADD off N'_8, off N_8	off N'_8 ← off N'_8 + off N_8	9	—	—	23	4	•	•	•		14
	ADD N'_8, off N_8	N'_8 ← N'_8 + off N_8	9	18	—	—	4	•	•	•		14
	ADD [DP], off N_8	[DP] ← [DP] + off N_8	9	18	13	22	3	•	•	•		14
	ADD ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP]+ off N_8	10	19	14	23	4	•	•	•		14
	ADD N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] + off N_8	11	20	15	24	5	•	•	•		14
	ADD N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] + off N_8	11	20	15	24	5	•	•	•		14
618	ADD erN, #N_{16}	erN ← erN + #N_{16}	8		17		4	•	•	•		15
	ADD DP, #N_{16}	DP ← DP + #N_{16}	8		—		4	•	•	•		15
	ADD X1, #N_{16}	X1 ← X1 + #N_{16}	8		—		4	•	•	•		15
	ADD X2, #N_{16}	X2 ← X2 + #N_{16}	8		—		4	•	•	•		15
	ADD USP, #N_{16}	USP ← USP + #N_{16}	8		—		4	•	•	•		15
	ADD SSP, #N_{16}	SSP ← SSP + #N_{16}	8		—		4	•	•	•		15
	ADD LRB, #N_{16}	LRB ← LRB + #N_{16}	8		—		4	•	•	•		15

TABLE 3-15. Arithmetic Calculation Instructions (2)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
625	ADD off N_8, #N_{16}	off N_8 ← off N_8 + #N_{16}	10		20		5	•	•	•		15
	ADD N_8, #N_{16}	N_8 ← N_8 + #N_{16}	10		—		5	•	•	•		15
	ADD [DP], #N_{16}	[DP] ← [DP] + #N_{16}	10		19		4	•	•	•		15
	ADD ±N_8[USP], #N_{16}	±N_8[USP] ← ±N_8[USP]+ #N_{16}	11		20		5	•	•	•		15
	ADD N'_{16}[X1], #N_{16}	N'_{16}[X1] ← N'_{16}[X1] + #N_{16}	12		21		6	•	•	•		15
	ADD N'_{16}[X2], #N_{16}	N'_{16}[X2] ← N'_{16}[X2] + #N_{16}	12		21		6	•	•	•		15
631	ADDB A, #N_8	A_L ← A_L + #N_8	4	—			2	•	•	•		16
	ADDB A, rN	A_L ← A_L + rN	3	5			1	•	•	•		16
	ADDB A, PSWH	A_L ← A_L + PSWH	4	—			2	•	•	•		16
	ADDB A, PSWL	A_L ← A_L + PSWL	4	—			2	•	•	•		16
	ADDB A, off N_8	A_L ← A_L + off N_8	4	7			2	•	•	•		16
	ADDB A, N_8	A_L ← A_L + N_8	6	—			3	•	•	•		16
	ADDB A, [DP]	A_L ← A_L + [DP]	6	—			2	•	•	•		16
	ADDB A, ±N_8[USP]	A_L ← A_L + ±N_8[USP]	7	9			3	•	•	•		16
	ADDB A, N_{16}[X1]	A_L ← A_L + N_{16}[X1]	8	10			4	•	•	•		16
	ADDB A, N_{16}[X2]	A_L ← A_L + N_{16}[X2]	8	10			4	•	•	•		16
641	ADDB rN, A	rN ← rN + A_L	5		9		2	•	•	•		17
	ADDB PSWH, A	PSWH ← PSWH + A_L	5		—		2	•	•	•		17
	ADDB PSWL, A	PSWL ← PSWL + A_L	5		—		2	•	•	•		17
	ADDB off N_8, A	off N_8 ← off N_8 + A_L	7		12		3	•	•	•		17
	ADDB N_8, A	N_8 ← N_8 + A_L	7		—		3	•	•	•		17
	ADDB [DP], A	[DP] ← [DP] + A_L	7		11		2	•	•	•		17
	ADDB ±N_8[USP], A	±N_8[USP]←±N_8[USP]+ A_L	8		12		3	•	•	•		17
	ADDB N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1]+ A_L	9		13		4	•	•	•		17
	ADDB N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2]+ A_L	9		13		4	•	•	•		17
650	ADDB rN, off N_8	rN ← rN + off N_8	7	—	—	14	3	•	•	•		18
	ADDB PSWH, off N_8	PSWH ← PSWH+off N_8	7	12	—	—	3	•	•	•		18
	ADDB PSWL, off N_8	PSWL ← PSWL+off N_8	7	12	—	—	3	•	•	•		18
	ADDB off N'_8, off N_8	off N'_8 ← off N'_8 + off N_8	9	—	—	17	4	•	•	•		18
	ADDB N'_8, off N_8	N'_8 ← N'_8 + off N_8	9	14	—	—	4	•	•	•		18
	ADDB [DP], off N_8	[DP] ← [DP] + off N_8	9	14	11	13	3	•	•	•		18
	ADDB ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP]+ off N_8	10	15	12	17	4	•	•	•		18
	ADDB N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] + off N_8	11	13	13	16	5	•	•	•		18
	ADDB N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] + off N_8	11	13	13	16	5	•	•	•		18
659	ADDB rN, #N_8	rN ← rN + #N_8	6		11		3	•	•	•		19
	ADDB PSWH, #N_8	PSWH ← PSWH + #N_8	6		—		3	•	•	•		19
	ADDB PSWL, #N_8	PSWL ← PSWL + #N_8	6		—		3	•	•	•		19
	ADDB off N'_8, #N_8	off N'_8 ← off N'_8 + #N_8	8		14		4	•	•	•		19
	ADDB N'_8, #N_8	N'_8 ← N'_8 + #N_8	8		—		4	•	•	•		19
	ADDB [DP], #N_8	[DP] ← [DP] + #N_8	8		13		3	•	•	•		19
	ADDB ±N'_8[USP], #N_8	±N'_8[USP] ← ±N'_8[USP]+ #N_8	9		14		4	•	•	•		19
	ADDB N_{16}[X1], #N_8	N_{16}[X1] ← N_{16}[X1] + #N_8	10		15		5	•	•	•		19
	ADDB N_{16}[X2], #N_8	N_{16}[X2] ← N_{16}[X2] + #N_8	10		15		5	•	•	•		19

TABLE 3-16. Arithmetic Calculation Instructions (3)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
668	ADC A, #N_{16}	A ← A + #N_{16} + C	6	—			3	•	•	•		5
	ADC A, erN	A ← A + erN + C	3	7			1	•	•	•		5
	ADC A, DP	A ← A + DP + C	4	—			2	•	•	•		5
	ADC A, X1	A ← A + X1 + C	4	—			2	•	•	•		5
	ADC A, X2	A ← A + X2 + C	4	—			2	•	•	•		5
	ADC A, USP	A ← A + USP + C	4	—			2	•	•	•		5
	ADC A, SSP	A ← A + SSP + C	4	—			2	•	•	•		5
	ADC A, LRB	A ← A + LRB + C	4	—			2	•	•	•		5
	ADC A, off N_8	A ← A + off N_8 + C	4	9			2	•	•	•		5
	ADC A, N_8	A ← A + N_8 + C	6	—			3	•	•	•		5
	ADC A, [DP]	A ← A + [DP] + C	6	10			2	•	•	•		5
	ADC A, ±N_8[USP]	A ← A + ±N_8[USP] + C	7	11			3	•	•	•		5
	ADC A, N_{16}[X1]	A ← A + N_{16}[X1] + C	8	12			4	•	•	•		5
	ADC A, N_{16}[X2]	A ← A + N_{16}[X2] + C	8	12			4	•	•	•		5
682	ADC erN, A	erN ← erN + A + C	5		13		2	•	•	•		6
	ADC DP, A	DP ← DP + A + C	5		—		2	•	•	•		6
	ADC X1, A	X1 ← X1 + A + C	5		—		2	•	•	•		6
	ADC X2, A	X2 ← X2 + A + C	5		—		2	•	•	•		6
	ADC USP, A	USP ← USP + A + C	5		—		2	•	•	•		6
	ADC SSP, A	SSP ← SSP + A + C	5		—		2	•	•	•		6
	ADC LRB, A	LRB ← LRB + A + C	5		—		2	•	•	•		6
	ADC off N_8, A	off N_8 ← off N_8 + A + C	7		13		3	•	•	•		6
	ADC N_8, A	N_8 ← N_8 + A + C	7		—		3	•	•	•		6
	ADC [DP], A	[DP] ← [DP] + A + C	7		15		2	•	•	•		6
	ADC ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] + A + C	8		13		3	•	•	•		6
	ADC N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] + A + C	9		17		4	•	•	•		6
	ADC N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] + A + C	9		17		4	•	•	•		6
695	ADC erN, off N_8	erN ← erN + off N_8 + C	7	—	—	20	3	•	•	•		7
	ADC DP, off N_8	DP ← DP + off N_8 + C	7	13	—	—	3	•	•	•		7
	ADC X1, off N_8	X1 ← X1 + off N_8 + C	7	13	—	—	3	•	•	•		7
	ADC X2, off N_8	X2 ← X2 + off N_8 + C	7	13	—	—	3	•	•	•		7
	ADC USP, off N_8	USP ← USP + off N_8 + C	7	13	—	—	3	•	•	•		7
	ADC SSP, off N_8	SSP ← SSP + off N_8 + C	7	13	—	—	3	•	•	•		7
	ADC LRB, off N_8	LRB ← LRB + off N_8 + C	7	13	—	—	3	•	•	•		7
	ADC off N'_8, off N_8	off N'_8 ← off N'_8 + off N_8 + C	9	—	—	23	4	•	•	•		7
	ADC N'_8, off N_8	N'_8 ← N'_8 + off N_8 + C	9	18	—	—	4	•	•	•		7
	ADC [DP], off N_8	[DP] ← [DP] + off N_8 + C	9	18	13	22	3	•	•	•		7
	ADC ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP] + off N_8 + C	10	19	14	23	4	•	•	•		7
	ADC N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] + off N_8 + C	11	20	15	24	5	•	•	•		7
	ADC N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] + off N_8 + C	11	20	15	24	5	•	•	•		7
708	ADC erN, #N_{16}	erN ← erN + #N_{16} + C	8		17		4	•	•	•		8
	ADC DP, #N_{16}	DP ← DP + #N_{16} + C	8		—		4	•	•	•		8
	ADC X1, #N_{16}	X1 ← X1 + #N_{16} + C	8		—		4	•	•	•		8
	ADC X2, #N_{16}	X2 ← X2 + #N_{16} + C	8		—		4	•	•	•		8
	ADC USP, #N_{16}	USP ← USP + #N_{16} + C	8		—		4	•	•	•		8
	ADC SSP, #N_{16}	SSP ← SSP + #N_{16} + C	8		—		4	•	•	•		8
	ADC LRB, #N_{16}	LRB ← LRB + #N_{16} + C	8		—		4	•	•	•		8
	ADC off N_8, #N_{16}	off N_8 ← off N_8 + #N_{16} + C	10		20		5	•	•	•		8
	ADC N_8, #N_{16}	N_8 ← N_8 + #N_{16} + C	10		—		5	•	•	•		8
	ADC [DP], #N_{16}	[DP] ← [DP] + #N_{16} + C	10		19		4	•	•	•		8
	ADC ±N_8[USP], #N_{16}	±N_8[USP] ← ±N_8[USP] + #N_{16} + C	11		20		5	•	•	•		8

TABLE 3-17. Arithmetic Calculation Instructions (4)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
719	ADC N'_{16}[X1], #N_{16}	N'_{16}[X1] ← N'_{16}[X1] + #N_{16} + C	12		21		6	•	•	•		8
	ADC N'_{16}[X2], #N_{16}	N'_{16}[X2] ← N'_{16}[X2] + #N_{16} + C	12		21		6	•	•	•		8
721	ADCB A, #N_8	A_L ← A_L + #N_8 + C	4	—			2	•	•	•		9
	ADCB A, rN	A_L ← A_L + rN + C	3	5			1	•	•	•		9
	ADCB A, PSWH	A_L ← A_L + PSWH + C	4	—			2	•	•	•		9
	ADCB A, PSWL	A_L ← A_L + PSWL + C	4	—			2	•	•	•		9
	ADCB A, off N_8	A_L ← A_L + off N_8 + C	4	7			2	•	•	•		9
	ADCB A, N_8	A_L ← A_L + N_8 + C	6	—			3	•	•	•		9
	ADCB A, [DP]	A_L ← A_L + [DP] + C	6	—			2	•	•	•		9
	ADCB A, ±N_8[USP]	A_L ← A_L + ±N_8[USP] + C	7	9			3	•	•	•		9
	ADCB A, N_{16}[X1]	A_L ← A_L + N_{16}[X1] + C	8	10			4	•	•	•		9
	ADCB A, N_{16}[X2]	A_L ← A_L + N_{16}[X2] + C	8	10			4	•	•	•		9
731	ADCB rN, A	rN ← rN+A_L+C	5		9		2	•	•	•		13
	ADCB PSWH, A	PSWH ← PSWH+A_L+C	5		—		2	•	•	•		10
	ADCB PSWL, A	PSWL ← PSWL+A_L+C	5		—		2	•	•	•		10
	ADCB off N_8, A	off N_8 ← off N_8+A_L+C	7		12		3	•	•	•		10
	ADCB N_8, A	N_8 ← N_8+A_L+C	7		—		3	•	•	•		10
	ADCB [DP], A	[DP] ← [DP]+A_L+C	7		11		2	•	•	•		10
	ADCB ±N_8[USP], A	±N_8[USP] ← ±N_8[USP]+A_L+C			12		3	•	•	•		10
	ADCB N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1]+ A_L+C	9		13		4	•	•	•		10
	ADCB N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2]+ A_L+C	9		13		4	•	•	•		10
740	ADCB rN, off N_8	rN ← rN+off N_8+C	7	—	—	14	3	•	•	•		11
	ADCB PSWH, off N_8	PSWH ←PSWH+off N_8+C	7	12	—	—	3	•	•	•		11
	ADCB PSWL, off N_8	PSWL ←PSWL+off N_8+C	7	12	—	—	3	•	•	•		11
	ADCB off N'_8, off N_8	off N'_8 ← off N'_8+off N_8+C	9	—	—	17	4	•	•	•		11
	ADCB N'_8, off N_8	N'_8 ← N'_8+off N_8+C	9	144	—	—	4	•	•	•		11
	ADCB [DP], off N_8	[DP] ← [DP]+off N_8+C	9	14	11	13	3	•	•	•		11
	ADCB ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP]+ off N_8+C	10	15	12	17	4	•	•	•		11
	ADCB N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] + off N_8+C	11	13	13	18	5	•	•	•		11
	ADCB N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] + off N_8+C	11	13	13	18	5	•	•	•		11
743	ADCB rN, #N_8	rN ← rN+#N_8+C	6		11		3	•	•	•		11
	ADCB PSWH, #N_8	PSWH ←PSWH+#N_8+C	6		—		3	•	•	•		11
	ADCB PSWL, #N_8	PSWL ←PSWL+#N_8+C	6		—		3	•	•	•		11
	ADCB off N'_8, #N_8	off N'_8 ← off N'_8+#N_8+C	8		14		4	•	•	•		11
	ADCB N'_8, #N_8	N'_8 ← N'_8+#N_8+C	8		—		4	•	•	•		11
	ADCB [DP], #N_8	[DP] ← [DP]+#N_8+C	8		13		3	•	•	•		11
	ADCB ±N'_8[USP], #N_8	±N'_8[USP] ← ±N'_8[USP]+ #N_8+C	9		14		4	•	•	•		11
	ADCB N_{16}[X1], #N_8	N_{16}[X1] ← N_{16}[X1] + #N_8+C	10		15		5	•	•	•		11
	ADCB N_{16}[X2], #N_8	N_{16}[X2] ← N_{16}[X2] + #N_8+C	10		15		5	•	•	•		11

TABLE 3-18. Arithmetic Calculation Instructions (5)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
758	SUB A, #N_{16}	A ← A - #N_{16}	6	—			3	•	•	•		157
	SUB A, erN	A ← A - erN	3	7			1	•	•	•		157
	SUB A, DP	A ← A - DP	4	—			2	•	•	•		157
	SUB A, X1	A ← A - X1	4	—			2	•	•	•		157
	SUB A, X2	A ← A - X2	4	—			2	•	•	•		157
	SUB A, USP	A ← A - USP	4	—			2	•	•	•		157
	SUB A, SSP	A ← A - SSP	4	—			2	•	•	•		157
	SUB A, LRB	A ← A - LRB	4	—			2	•	•	•		157
	SUB A, off N_8	A ← A - off N_8	4	9			2	•	•	•		157
	SUB A, N_8	A ← A - N_8	6	—			3	•	•	•		157
	SUB A, [DP]	A ← A - [DP]	6	10			2	•	•	•		157
	SUB A, ±N_8[USP]	A ← A - ±N_8[USP]	7	11			3	•	•	•		157
	SUB A, N_{16}[X1]	A ← A - N_{16}[X1]	8	12			4	•	•	•		157
	SUB A, N_{16}[X2]	A ← A - N_{16}[X2]	8	12			4	•	•	•		157
772	SUB erN, A	erN ← erN - A	5		13		2	•	•	•		158
	SUB DP, A	DP ← DP - A	5		—		2	•	•	•		158
	SUB X1, A	X1 ← X1 - A	5		—		2	•	•	•		158
	SUB X2, A	X2 ← X2 - A	5		—		2	•	•	•		158
	SUB USP, A	USP ← USP - A	5		—		2	•	•	•		158
	SUB SSP, A	SSP ← SSP - A	5		—		2	•	•	•		158
	SUB LRB, A	LRB ← LRB - A	5		—		2	•	•	•		158
	SUB off N_8, A	off N_8 ← off N_8 - A	7		13		3	•	•	•		158
	SUB N_8, A	N_8 ← N_8 - A	7		—		3	•	•	•		158
	SUB [DP], A	[DP] ← [DP] - A	7		15		2	•	•	•		158
	SUB ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] - A	8		13		3	•	•	•		158
	SUB N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] - A	9		17		4	•	•	•		158
	SUB N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] - A	9		17		4	•	•	•		158
785	SUB erN, off N_8	erN ← erN - off N_8	7	—	—	20	3	•	•	•		159
	SUB DP, off N_8	DP ← DP - off N_8	7	13	—	—	3	•	•	•		159
	SUB X1, off N_8	X1 ← X1 - off N_8	7	13	—	—	3	•	•	•		159
	SUB X2, off N_8	X2 ← X2 - off N_8	7	13	—	—	3	•	•	•		159
	SUB USP, off N_8	USP ← USP - off N_8	7	13	—	—	3	•	•	•		159
	SUB SSP, off N_8	SSP ← SSP - off N_8	7	13	—	—	3	•	•	•		159
	SUB LRB, off N_8	LRB ← LRB - off N_8	7	13	—	—	3	•	•	•		159
	SUB off N'$_8$, off N_8	off N'$_8$ ← off N'$_8$ - off N_8	9	—	—	23	4	•	•	•		159
	SUB N'$_8$, off N_8	N'$_8$ ← N'$_8$ - off N_8	9	18	—	—	4	•	•	•		159
	SUB [DP], off N_8	[DP] ← [DP] - off N_8	9	18	13	22	3	•	•	•		159
	SUB ±N'$_8$[USP], off N_8	±N'$_8$[USP] ← ±N'$_8$[USP] - off N_8	10	19	14	23	4	•	•	•		159
	SUB N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] - off N_8	11	20	15	24	5	•	•	•		159
	SUB N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] - off N_8	11	20	15	24	5	•	•	•		159
798	SUB erN, #N_{16}	erN ← erN - #N_{16}	8		17		4	•	•	•		160
	SUB DP, #N_{16}	DP ← DP - #N_{16}	8		—		4	•	•	•		160
	SUB X1, #N_{16}	X1 ← X1 - #N_{16}	8		—		4	•	•	•		160
	SUB X2, #N_{16}	X2 ← X2 - #N_{16}	8		—		4	•	•	•		160
	SUB USP, #N_{16}	USP ← USP - #N_{16}	8		—		4	•	•	•		160
	SUB SSP, #N_{16}	SSP ← SSP - #N_{16}	8		—		4	•	•	•		160
	SUB LRB, #N_{16}	LRB ← LRB - #N_{16}	8		—		4	•	•	•		160
	SUB off N_8, #N_{16}	off N_8 ← off N_8 - #N_{16}	10		20		5	•	•	•		160
	SUB N_8, #N_{16}	N_8 ← N_8 - #N_{16}	10		—		5	•	•	•		160
	SUB [DP], #N_{16}	[DP] ← [DP] - #N_{16}	10		19		4	•	•	•		160
	SUB ±N_8[USP], #N_{16}	±N_8[USP] ← ±N_8[USP] - #N_{16}	11		20		5	•	•	•		160

TABLE 3-19. Arithmetic Calculation Instructions (6)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
809	SUB N'_{16}[X1], #N_{16}	N'_{16}[X1] ← N'_{16}[X1] - #N_{16}	12		21		6	•	•	•		160
	SUB N'_{16}[X2], #N_{16}	N'_{16}[X2] ← N'_{16}[X2] - #N_{16}	12		21		6	•	•	•		160
811	SUBB A, #N_8	A_L ← A_L - #N_8	4	—			2	•	•	•		161
	SUBB A, rN	A_L ← A_L - rN	3	5			1	•	•	•		161
	SUBB A, PSWH	A_L ← A_L - PSWH	4	—			2	•	•	•		161
	SUBB A, PSWL	A_L ← A_L - PSWL	4	—			2	•	•	•		161
	SUBB A, off N_8	A_L ← A_L - off N_8	4	7			2	•	•	•		161
	SUBB A, N_8	A_L ← A_L - N_8	6	—			3	•	•	•		161
	SUBB A, [DP]	A_L ← A_L - [DP]	6	—			2	•	•	•		161
	SUBB A, ±N_8[USP]	A_L ← A_L - ±N_8[USP]	7	9			3	•	•	•		161
	SUBB A, N_{16}[X1]	A_L ← A_L - N_{16}[X1]	8	10			4	•	•	•		161
	SUBB A, N_{16}[X2]	A_L ← A_L - N_{16}[X2]	8	10			4	•	•	•		161
821	SUBB rN, A	rN ← rN - A_L	5		9		2	•	•	•		162
	SUBB PSWH, A	PSWH ← PSWH - A_L	5		—		2	•	•	•		162
	SUBB PSWL, A	PSWL ← PSWL - A_L	5		—		2	•	•	•		162
	SUBB off N_8, A	off N_8 ← off N_8 - A_L	7		12		3	•	•	•		162
	SUBB N_8, A	N_8 ← N_8 - A_L	7		—		3	•	•	•		162
	SUBB [DP], A	[DP] ← [DP] - A_L	7		11		2	•	•	•		162
	SUBB ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] - A_L	8		12		3	•	•	•		162
	SUBB N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] - A_L	9		13		4	•	•	•		162
	SUBB N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] - A_L	9		13		4	•	•	•		162
830	SUBB rN, off N_8	rN ← rN - off N_8	7	—	—	14	3	•	•	•		163
	SUBB PSWH, off N_8	PSWH ← PSWH - off N_8	7	12	—	—	3	•	•	•		163
	SUBB PSWL, off N_8	PSWL ← PSWL - off N_8	7	12	—	—	3	•	•	•		163
	SUBB off N'_8, off N_8	off N'_8 ← off N'_8 - off N_8	9	—	—	17	4	•	•	•		163
	SUBB N'_8, off N_8	N'_8 ← N'_8 - off N_8	9	14	—	—	4	•	•	•		163
	SUBB [DP], off N_8	[DP] ← [DP] - off N_8	9	14	11	13	3	•	•	•		163
	SUBB ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP] - off N_8	10	15	12	17	4	•	•	•		163
	SUBB N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] - off N_8	11	13	13	18	5	•	•	•		163
	SUBB N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] - off N_8	11	13	13	18	5	•	•	•		163
839	SUBB rN, #N_8	rN ← rN - #N_8	6		11		3	•	•	•		164
	SUBB PSWH, #N_8	PSWH ← PSWH - #N_8	6		—		3	•	•	•		164
	SUBB PSWL, #N_8	PSWL ← PSWL - #N_8	6		—		3	•	•	•		164
	SUBB off N'_8, #N_8	off N'_8 ← off N'_8 - #N_8	6		14		4	•	•	•		164
	SUBB N'_8, #N_8	N'_8 ← N'_8 - #N_8	8		—		4	•	•	•		164
	SUBB [DP], #N_8	[DP] ← [DP] - #N_8	8		13		3	•	•	•		164
	SUBB ±N'_8[USP], #N_8	±N'_8[USP] ← ±N'_8[USP] - #N_8	9		14		4	•	•	•		164
	SUBB N_{16}[X1], #N_8	N_{16}[X1] ← N_{16}[X1] - #N_8	10		15		5	•	•	•		164
	SUBB N_{16}[X2], #N_8	N_{16}[X2] ← N_{16}[X2] - #N_8	10		15		5	•	•	•		164
848	SBC A, #N_{16}	A ← A - #N_{16} - C	6	—			3	•	•	•		129
	SBC A, erN	A ← A - erN - C	3	7			1	•	•	•		129
	SBC A, DP	A ← A - DP - C	4	—			2	•	•	•		129
	SBC A, X1	A ← A - X1 - C	4	—			2	•	•	•		129
	SBC A, X2	A ← A - X2 - C	4	—			2	•	•	•		129
	SBC A, USP	A ← A - USP - C	4	—			2	•	•	•		129
	SBC A, SSP	A ← A - SSP - C	4	—			2	•	•	•		129
	SBC A, LRB	A ← A - LRB - C	4	—			2	•	•	•		129

TABLE 3-20. Arithmetic Calculation Instructions (7)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
856	SBC A, off N_8	A ← A - off N_8 - C	4	9			2	•	•	•		129
	SBC A, N_8	A ← A - N_8 - C	6	—			3	•	•	•		129
	SBC A, [DP]	A ← A - [DP] - C	6	10			2	•	•	•		129
	SBC A, ±N_8[USP]	A ← A - ±N_8[USP] - C	7	11			3	•	•	•		129
	SBC A, N_{16}[X1]	A ← A - N_{16}[X1] - C	8	12			4	•	•	•		129
	SBC A, N_{16}[X2]	A ← A - N_{16}[X2] - C	8	12			4	•	•	•		129
852	SBC erN, A	erN ← erN - A - C	5		13		2	•	•	•		130
	SBC DP, A	DP ← DP - A - C	5		—		2	•	•	•		130
	SBC X1, A	X1 ← X1 - A - C	5		—		2	•	•	•		130
	SBC X2, A	X2 ← X2 - A - C	5		—		2	•	•	•		130
	SBC USP, A	USP ← USP - A - C	5		—		2	•	•	•		130
	SBC SSP, A	SSP ← SSP - A - C	5		—		2	•	•	•		130
	SBC LRB, A	LRB ← LRB - A - C	5		—		2	•	•	•		130
	SBC off N_8, A	off N_8 ← off N_8 - A - C	7		13		3	•	•	•		130
	SBC N_8, A	N_8 ← N_8 - A - C	7		—		3	•	•	•		130
	SBC [DP], A	[DP] ← [DP] - A - C	7		15		2	•	•	•		130
	SBC ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] - A - C	8		13		3	•	•	•		130
	SBC N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] - A - C	9		17		4	•	•	•		130
	SBC N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] - A - C	9		17		4	•	•	•		130
875	SBC erN, off N_8	erN ← erN - off N_8 - C	7	—	—	20	3	•	•	•		131
	SBC DP, off N_8	DP ← DP - off N_8 - C	7	13	—	—	3	•	•	•		131
	SBC X1, off N_8	X1 ← X1 - off N_8 - C	7	13	—	—	3	•	•	•		131
	SBC X2, off N_8	X2 ← X2 - off N_8 - C	7	13	—	—	3	•	•	•		131
	SBC USP, off N_8	USP ← USP - off N_8 - C	7	13	—	—	3	•	•	•		131
	SBC SSP, off N_8	SSP ← SSP - off N_8 - C	7	13	—	—	3	•	•	•		131
	SBC LRB, off N_8	LRB ← LRB - off N_8 - C	7	13	—	—	3	•	•	•		131
	SBC off N'$_8$, off N_8	off N'$_8$ ← off N'$_8$ - off N_8 - C	9	—	—	23	4	•	•	•		131
	SBC N'$_8$, off N_8	N'$_8$ ← N'$_8$ - off N_8 - C	9	18	—	—	4	•	•	•		131
	SBC [DP], off N_8	[DP] ← [DP] - off N_8 - C	9	18	13	22	3	•	•	•		131
	SBC ±N'$_8$[USP], off N_8	±N'$_8$[USP] ← ±N'$_8$[USP] - off N_8 - C	10	19	14	23	4	•	•	•		131
	SBC N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] - off N_8 - C	11	20	15	24	5	•	•	•		131
	SBC N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] - off N_8 - C	11	20	15	24	5	•	•	•		131
868	SBC erN, #N_{16}	erN ← erN - #N_{16} - C	8		17		4	•	•	•		132
	SBC DP, #N_{16}	DP ← DP - #N_{16} - C	8		—		4	•	•	•		132
	SBC X1, #N_{16}	X1 ← X1 - #N_{16} - C	8		—		4	•	•	•		132
	SBC X2, #N_{16}	X2 ← X2 - #N_{16} - C	8		—		4	•	•	•		132
	SBC USP, #N_{16}	USP ← USP - #N_{16} - C	8		—		4	•	•	•		132
	SBC SSP, #N_{16}	SSP ← SSP - #N_{16} - C	8		—		4	•	•	•		132
	SBC LRB, #N_{16}	LRB ← LRB - #N_{16} - C	8		—		4	•	•	•		132
	SBC off N_8, #N_{16}	off N_8 ← off N_8 - #N_{16} - C	10		20		5	•	•	•		132
	SBC N_8, #N_{16}	N_8 ← N_8 - #N_{16} - C	10		—		5	•	•	•		132
	SBC [DP], #N_{16}	[DP] ← [DP] - #N_{16} - C	10		19		4	•	•	•		132
	SBC ±N_8[USP], #N_{16}	±N_8[USP] ← ±N_8[USP] - #N_{16} - C	11		20		5	•	•	•		132
	SBC N'$_{16}$[X1], #N_{16}	N'$_{16}$[X1] ← N'$_{16}$[X1] - #N_{16} - C	12		21		6	•	•	•		132
	SBC N'$_{16}$[X2], #N_{16}	N'$_{16}$[X2] ← N'$_{16}$[X2] - #N_{16} - C	12		21		6	•	•	•		132

TABLE 3-21. Arithmetic Calculation Instructions (8)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
901	SBCB A, #N_8	$A_L \leftarrow A_L - \#N_8 - C$	4	—			2	•	•	•		133
	SBCB A, rN	$A_L \leftarrow A_L - rN - C$	3	5			1	•	•	•		133
	SBCB A, PSWH	$A_L \leftarrow A_L - PSWH - C$	4	—			2	•	•	•		133
	SBCB A, PSWL	$A_L \leftarrow A_L - PSWL - C$	4	—			2	•	•	•		133
	SBCB A, off N_8	$A_L \leftarrow A_L - off\ N_8 - C$	4	7			2	•	•	•		133
	SBCB A, N_8	$A_L \leftarrow A_L - N_8 - C$	6	—			3	•	•	•		133
	SBCB A, [DP]	$A_L \leftarrow A_L - [DP] - C$	6	—			2	•	•	•		133
	SBCB A, ±N_8[USP]	$A_L \leftarrow A_L - \pm N_8[USP] - C$	7	9			3	•	•	•		133
	SBCB A, N_{16}[X1]	$A_L \leftarrow A_L - N_{16}[X1] - C$	8	10			4	•	•	•		133
	SBCB A, N_{16}[X2]	$A_L \leftarrow A_L - N_{16}[X2] - C$	8	10			4	•	•	•		133
911	SBCB rN, A	$rN \leftarrow rN - A_L - C$	5		9		2	•	•	•		134
	SBCB PSWH, A	$PSWH \leftarrow PSWH - A_L - C$	5		—		2	•	•	•		134
	SBCB PSWL, A	$PSWL \leftarrow PSWL - A_L - C$	5		—		2	•	•	•		134
	SBCB off N_8, A	$off\ N_8 \leftarrow off\ N_8 - A_L - C$	7		12		3	•	•	•		134
	SBCB N_8, A	$N_8 \leftarrow N_8 - A_L - C$	7		—		3	•	•	•		134
	SBCB [DP], A	$[DP] \leftarrow [DP] - A_L - C$	7		11		2	•	•	•		134
	SBCB ±N_8[USP], A	$\pm N_8[USP] \leftarrow \pm N_8[USP] - A_L - C$			12		3	•	•	•		134
	SBCB N_{16}[X1], A	$N_{16}[X1] \leftarrow N_{16}[X1] - A_L - C$	9		13		4	•	•	•		134
	SBCB N_{16}[X2], A	$N_{16}[X2] \leftarrow N_{16}[X2] - A_L - C$	9		13		4	•	•	•		134
920	SBCB rN, off N_8	$rN \leftarrow rN - off\ N_8 - C$	7	—	—	14	3	•	•	•		135
	SBCB PSWH, off N_8	$PSWH \leftarrow PSWH - off\ N_8 - C$	7	12	—	—	3	•	•	•		135
	SBCB PSWL, off N_8	$PSWL \leftarrow PSWL - off\ N_8 - C$	7	12	—	—	3	•	•	•		135
	SBCB off N'_8, off N_8	$off\ N'_8 \leftarrow off\ N'_8 - off\ N_8 - C$	9	—	—	17	4	•	•	•		135
	SBCB N'_8, off N_8	$N'_8 \leftarrow N'_8 - off\ N_8 - C$	9	14	—	—	4	•	•	•		135
	SBCB [DP], off N_8	$[DP] \leftarrow [DP] - off\ N_8 - C$	9	14	11	13	3	•	•	•		135
	SBCB ±N'_8[USP], off N_8	$\pm N'_8[USP] \leftarrow \pm N'_8[USP] - off\ N_8 - C$	10	15	12	17	4	•	•	•		135
	SBCB N_{16}[X1], off N_8	$N_{16}[X1] \leftarrow N_{16}[X1] - off\ N_8 - C$	11	13	13	18	5	•	•	•		135
	SBCB N_{16}[X2], off N_8	$N_{16}[X2] \leftarrow N_{16}[X2] - off\ N_8 - C$	11	13	13	18	5	•	•	•		135
929	SBCB rN, #N_8	$rN \leftarrow rN - \#N_8 - C$	6		11		3	•	•	•		136
	SBCB PSWH, #N_8	$PSWH \leftarrow PSWH - \#N_8 - C$	6		—		3	•	•	•		136
	SBCB PSWL, #N_8	$PSWL \leftarrow PSWL - \#N_8 - C$	6		—		3	•	•	•		136
	SBCB off N'_8, #N_8	$off\ N'_8 \leftarrow off\ N'_8 - \#N_8 - C$	8		14		4	•	•	•		136
	SBCB N'_8, #N_8	$N'_8 \leftarrow N'_8 - \#N_8 - C$	8		—		4	•	•	•		136
	SBCB [DP], #N_8	$[DP] \leftarrow [DP] - \#N_8 - C$	8		13		3	•	•	•		136
	SBCB ±N'_8[USP], #N_8	$\pm N'_8[USP] \leftarrow \pm N'_8[USP] - \#N_8 - C$	9		14		4	•	•	•		136
	SBCB N_{16}[X1], #N_8	$N_{16}[X1] \leftarrow N_{16}[X1] - \#N_8 - C$	10		15		5	•	•	•		136
	SBCB N_{16}[X2], #N_8	$N_{16}[X2] \leftarrow N_{16}[X2] - \#N_8 - C$	10		15		5	•	•	•		136

TABLE 3-22. Logical Calculation Instructions (1)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
938	AND A, #N_{16}	A ← A ∧ N_{16}	6	—			3	•				20
	AND A, erN	A ← A ∧ erN	3	7			1	•				20
	AND A, DP	A ← A ∧ DP	4	—			2	•				20
	AND A, X1	A ← A ∧ X1	4	—			2	•				20
	AND A, X2	A ← A ∧ X2	4	—			2	•				20
	AND A, USP	A ← A ∧ USP	4	—			2	•				20
	AND A, SSP	A ← A ∧ SSP	4	—			2	•				20
	AND A, LRB	A ← A ∧ LRB	4	—			2	•				20
	AND A, off N_8	A ← A ∧ off N_8	4	9			2	•				20
	AND A, N_8	A ← A ∧ N_8	6	—			3	•				20
	AND A, [DP]	A ← A ∧ [DP]	6	10			2	•				20
	AND A, ±N_8[USP]	A ← A ∧ ±N_8[USP]	7	11			3	•				20
	AND A, N_{16}[X1]	A ← A ∧ N_{16}[X1]	8	12			4	•				20
	AND A, N_{16}[X2]	A ← A ∧ N_{16}[X2]	8	12			4	•				20
952	AND erN, A	erN ← erN ∧ A	5		13		2	•				21
	AND DP, A	DP ← DP ∧ A	5		—		2	•				21
	AND X1, A	X1 ← X1 ∧ A	5		—		2	•				21
	AND X2, A	X2 ← X2 ∧ A	5		—		2	•				21
	AND USP, A	USP ← USP ∧ A	5		—		2	•				21
	AND SSP, A	SSP ← SSP ∧ A	5		—		2	•				21
	AND LRB, A	LRB ← LRB ∧ A	5		—		2	•				21
	AND off N_8, A	off N_8 ← off N_8 ∧ A	7		13		3	•				21
	AND N_8, A	N_8 ← N_8 ∧ A	7		—		3	•				21
	AND [DP], A	[DP] ← [DP] ∧ A	7		15		2	•				21
	AND ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] ∧ A	8		13		3	•				21
	AND N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] ∧ A	9		17		4	•				21
	AND N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] ∧ A	9		17		4	•				21
965	AND erN, off N_8	erN ← erN ∧ off N_8	7	—	—	20	3	•				22
	AND DP, off N_8	DP ← DP ∧ off N_8	7	13	—	—	3	•				22
	AND X1, off N_8	X1 ← X1 ∧ off N_8	7	13	—	—	3	•				22
	AND X2, off N_8	X2 ← X2 ∧ off N_8	7	13	—	—	3	•				22
	AND USP, off N_8	USP ← USP ∧ off N_8	7	13	—	—	3	•				22
	AND SSP, off N_8	SSP ← SSP ∧ off N_8	7	13	—	—	3	•				22
	AND LRB, off N_8	LRB ← LRB ∧ off N_8	7	13	—	—	3	•				22
	AND off N'$_8$, off N_8	off N'$_8$ ← off N'$_8$ ∧ off N_8	9	—	—	23	4	•				22
	AND N'$_8$, off N_8	N'$_8$ ← N'$_8$ ∧ off N_8	9	18	—	—	4	•				22
	AND [DP], off N_8	[DP] ← [DP] ∧ off N_8	9	18	13	22	3	•				22
	AND ±N'$_8$[USP], off N_8	±N'$_8$[USP] ← ±N'$_8$[USP] ∧ off N_8	10	19	14	23	4	•				22
	AND N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] ∧ off N_8	11	20	15	24	5	•				22
	AND N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] ∧ off N_8	11	20	15	24	5	•				22
978	AND erN, #N_{16}	erN ← erN ∧ #N_{16}	8		17		4	•				23
	AND DP, #N_{16}	DP ← DP ∧ #N_{16}	8		—		4	•				23
	AND X1, #N_{16}	X1 ← X1 ∧ #N_{16}	8		—		4	•				23
	AND X2, #N_{16}	X2 ← X2 ∧ #N_{16}	8		—		4	•				23
	AND USP, #N_{16}	USP ← USP ∧ #N_{16}	8		—		4	•				23
	AND SSP, #N_{16}	SSP ← SSP ∧ #N_{16}	8		—		4	•				23
	AND LRB, #N_{16}	LRB ← LRB ∧ #N_{16}	8		—		4	•				23
	AND off N_8, #N_{16}	off N_8 ← off N_8 ∧ #N_{16}	10		20		5	•				23
	AND N_8, #N_{16}	N_8 ← N_8 ∧ #N_{16}	10		—		5	•				23
	AND [DP], #N_{16}	[DP] ← [DP] ∧ #N_{16}	10		19		4	•				23
	AND ±N_8[USP], #N_{16}	±N_8[USP] ← ±N_8[USP] ∧ #N_{16}	11		20		5	•				23

TABLE 3-23. Logical Calculation Instructions (2)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
989	AND N'_{16}[X1], #N_{16}	N'_{16}[X1] ← N'_{16}[X1] ∧#N_{16}	12		21		6	•				23
	AND N'_{16}[X2], #N_{16}	N'_{16}[X2] ← N'_{16}[X2] ∧#N_{16}	12		21		6	•				23
991	ANDB A, #N_8	A_L ← A_L ∧#N_8	4	—			2	•				24
	ANDB A, rN	A_L ← A_L ∧rN	3	5			1	•				24
	ANDB A, PSWH	A_L ← A_L ∧PSWH	4	—			2	•				24
	ANDB A, PSWL	A_L ← A_L ∧PSWL	4	—			2	•				24
	ANDB A, off N_8	A_L ← A_L ∧off N_8	4	7			2	•				24
	ANDB A, N_8	A_L ← A_L ∧N_8	6	—			3	•				24
	ANDB A, [DP]	A_L ← A_L ∧[DP]	6	—			2	•				24
	ANDB A, ±N_8[USP]	A_L ← A_L ∧±N_8[USP]	7	9			3	•				24
	ANDB A, N_{16}[X1]	A_L ← A_L ∧N_{16}[X1]	8	10			4	•				24
	ANDB A, N_{16}[X2]	A_L ← A_L ∧N_{16}[X2]	8	10			4	•				24
1001	ANDB rN, A	rN ← rN ∧A_L	5		9		2	•				25
	ANDB PSWH, A	PSWH ← PSWH ∧A_L	5		—		2	•				25
	ANDB PSWL, A	PSWL ← PSWL ∧A_L	5		—		2	•				25
	ANDB off N_8, A	off N_8 ← off N_8 ∧A_L	7		12		3	•				25
	ANDB N_8, A	N_8 ← N_8 ∧A_L	7		—		3	•				25
	ANDB [DP], A	[DP] ← [DP] ∧A_L	7		11		2	•				25
	ANDB ±N_8[USP], A	±N_8[USP]← ±N_8[USP]∧A_L	8		12		3	•				25
	ANDB N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1]∧A_L	9		13		4	•				25
	ANDB N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2]∧A_L	9		13		4	•				25
1010	ANDB rN, off N_8	rN ← rN ∧off N_8	7	—	—	14	3	•				32
	ANDB PSWH, off N_8	PSWH ←PSWH∧off N_8	7	12	—	—	3	•				26
	ANDB PSWL, off N_8	PSWL ←PSWL∧off N_8	7	12	—	—	3	•				26
	ANDB off N'_8, off N_8	off N'_8 ← off N'_8 ∧off N_8	9	—	—	17	4	•				26
	ANDB N'_8, off N_8	N'_8 ← N'_8 ∧off N_8	9	14	—	—	4	•				26
	ANDB [DP], off N_8	[DP] ← [DP] ∧off N_8	9	14	11	13	3	•				26
	ANDB ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP]∧off N_8	10	15	12	17	4	•				26
	ANDB N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] ∧off N_8	11	13	13	18	5	•				26
	ANDB N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] ∧off N_8	11	13	13	18	5	•				26
1019	ANDB rN, #N_8	rN ← rN ∧#N_8	6		11		3	•				27
	ANDB PSWH, #N_8	PSWH ←PSWH ∧#N_8	6		—		3	•				27
	ANDB PSWL, #N_8	PSWL ← PSWL ∧#N_8	6		—		3	•				27
	ANDB off N'_8, #N_8	off N'_8 ← off N'_8 ∧#N_8	8		14		4	•				27
	ANDB N'_8, #N_8	N'_8 ← N'_8 ∧#N_8	8		—		4	•				27
	ANDB [DP], #N_8	[DP] ← [DP] ∧#N_8	8		13		3	•				27
	ANDB ±N'_8[USP], #N_8	±N'_8[USP] ← ±N'_8[USP]∧#N_8	9		14		4	•				27
	ANDB N_{16}[X1], #N_8	N_{16}[X1] ← N_{16}[X1] ∧#N_8	10		15		5	•				27
	ANDB N_{16}[X2], #N_8	N_{16}[X2] ← N_{16}[X2] ∧#N_8	10		15		5	•				27

TABLE 3-24. Logical Calculation Instructions (3)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
1028	OR A, #N_{16}	A ← A ∨ N_{16}	6	—			3	•				104
	OR A, erN	A ← A ∨ erN	3	7			1	•				104
	OR A, DP	A ← A ∨ DP	4	—			2	•				104
	OR A, X1	A ← A ∨ X1	4	—			2	•				104
	OR A, X2	A ← A ∨ X2	4	—			2	•				104
	OR A, USP	A ← A ∨ USP	4	—			2	•				104
	OR A, SSP	A ← A ∨ SSP	4	—			2	•				104
	OR A, LRB	A ← A ∨ LRB	4	—			2	•				104
	OR A, off N_8	A ← A ∨ off N_8	4	9			2	•				104
	OR A, N_8	A ← A ∨ N_8	6	—			3	•				104
	OR A, [DP]	A ← A ∨ [DP]	6	10			2	•				104
	OR A, ±N_8[USP]	A ← A ∨ ±N_8[USP]	7	11			3	•				104
	OR A, N_{16}[X1]	A ← A ∨ N_{16}[X1]	8	12			4	•				104
	OR A, N_{16}[X2]	A ← A ∨ N_{16}[X2]	8	12			4	•				104
1042	OR erN, A	erN ← erN ∨ A	5		13		2	•				105
	OR DP, A	DP ← DP ∨ A	5		—		2	•				105
	OR X1, A	X1 ← X1 ∨ A	5		—		2	•				105
	OR X2, A	X2 ← X2 ∨ A	5		—		2	•				105
	OR USP, A	USP ← USP ∨ A	5		—		2	•				105
	OR SSP, A	SSP ← SSP ∨ A	5		—		2	•				105
	OR LRB, A	LRB ← LRB ∨ A	5		—		2	•				105
	OR off N_8, A	off N_8 ← off N_8 ∨ A	7		13		3	•				105
	OR N_8, A	N_8 ← N_8 ∨ A	7		—		3	•				105
	OR [DP], A	[DP] ← [DP] ∨ A	7		15		2	•				105
	OR ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] ∨ A	8		13		3	•				105
	OR N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] ∨ A	9		17		4	•				105
	OR N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] ∨ A	9		17		4	•				105
1055	OR erN, off N_8	erN ← erN ∨ off N_8	7	—	—	20	3	•				106
	OR DP, off N_8	DP ← DP ∨ off N_8	7	13	—	—	3	•				106
	OR X1, off N_8	X1 ← X1 ∨ off N_8	7	13	—	—	3	•				106
	OR X2, off N_8	X2 ← X2 ∨ off N_8	7	13	—	—	3	•				106
	OR USP, off N_8	USP ← USP ∨ off N_8	7	13	—	—	3	•				106
	OR SSP, off N_8	SSP ← SSP ∨ off N_8	7	13	—	—	3	•				106
	OR LRB, off N_8	LRB ← LRB ∨ off N_8	7	13	—	—	3	•				106
	OR off N'_8, off N_8	off N'_8 ← off N'_8 ∨ off N_8	9	—	—	23	4	•				106
	OR N'_8, off N_8	N'_8 ← N'_8 ∨ off N_8	9	18	—	—	4	•				106
	OR [DP], off N_8	[DP] ← [DP] ∨ off N_8	9	18	13	22	3	•				106
	OR ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP] ∨ off N_8	10	19	14	23	4	•				106
	OR N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] ∨ off N_8	11	20	15	24	5	•				106
	OR N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] ∨ off N_8	11	20	15	24	5	•				106
1068	OR erN, #N_{16}	erN ← erN ∨ #N_{16}	8		17		4	•				107
	OR DP, #N_{16}	DP ← DP ∨ #N_{16}	8		—		4	•				107
	OR X1, #N_{16}	X1 ← X1 ∨ #N_{16}	8		—		4	•				107
	OR X2, #N_{16}	X2 ← X2 ∨ #N_{16}	8		—		4	•				107
	OR USP, #N_{16}	USP ← USP ∨ #N_{16}	8		—		4	•				107
	OR SSP, #N_{16}	SSP ← SSP ∨ #N_{16}	8		—		4	•				107
	OR LRB, #N_{16}	LRB ← LRB ∨ #N_{16}	8		—		4	•				107
	OR off N_8, #N_{16}	off N_8 ← off N_8 ∨ #N_{16}	10		20		5	•				107
	OR N_8, #N_{16}	N_8 ← N_8 ∨ #N_{16}	10		—		5	•				107
	OR [DP], #N_{16}	[DP] ← [DP] ∨ #N_{16}	10		19		4	•				107
	OR ±N_8[USP], #N_{16}	±N_8[USP] ← ±N_8[USP] ∨ #N_{16}	11		20		5	•				107

TABLE 3-25. Logical Calculation Instructions (4)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
1079	OR N'_{16}[X1], #N_{16}	N'_{16}[X1] ← N'_{16}[X1] ∨#N_{16}	12		21		6	•				107
	OR N'_{16}[X2], #N_{16}	N'_{16}[X2] ← N'_{16}[X2] ∨#N_{16}	12		21		6	•				107
1081	ORB A, #N_8	A_L ← A_L ∨#N_8	4	—				•				108
	ORB A, rN	A_L ← A_L ∨rN	3	5				•				108
	ORB A, PSWH	A_L ← A_L ∨PSWH	4	—				•				108
	ORB A, PSWL	A_L ← A_L ∨PSWL	4	—				•				108
	ORB A, off N_8	A_L ← A_L ∨off N_8	4	7				•				108
	ORB A, N_8	A_L ← A_L ∨N_8	6	—				•				108
	ORB A, [DP]	A_L ← A_L ∨[DP]	6	—				•				108
	ORB A, ±N_8[USP]	A_L ← A_L ∨±N_8[USP]	7	9				•				108
	ORB A, N_{16}[X1]	A_L ← A_L ∨N_{16}[X1]	8	10				•				108
	ORB A, N_{16}[X2]	A_L ← A_L ∨N_{16}[X2]	8	10				•				108
1091	ORB rN, A	rN ← rN ∨A_L	5		9		2	•				109
	ORB PSWH, A	PSWH ← PSWH ∨A_L	5		—		2	•				109
	ORB PSWL, A	PSWL ← PSWL ∨A_L	5		—		2	•				109
	ORB off N_8, A	off N_8 ← off N_8 ∨A_L	7		12		3	•				109
	ORB N_8, A	N_8 ← N_8 ∨A_L	7		—		3	•				109
	ORB [DP], A	[DP] ← [DP] ∨A_L	7		11		2	•				109
	ORB ±N_8[USP], A	±N_8[USP]← ±N_8[USP]∨A_L	8		12		3	•				109
	ORB N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1]∨A_L	9		13		4	•				109
	ORB N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2]∨A_L	9		13		4	•				109
1100	ORB rN, off N_8	rN ← rN ∨off N_8	7	—	—	14	3	•				110
	ORB PSWH, off N_8	PSWH ←PSWH∨off N_8	7	12	—	—	3	•				110
	ORB PSWL, off N_8	PSWL ←PSWL∨off N_8	7	12	—	—	3	•				110
	ORB off N'_8, off N_8	off N'_8 ← off N'_8 ∨off N_8	9	—	—	17	4	•				110
	ORB N'_8, off N_8	N'_8 ← N'_8 ∨off N_8	9	14	—	—	4	•				110
	ORB [DP], off N_8	[DP] ← [DP] ∨off N_8	9	14	11	13	3	•				110
	ORB ±N'_8[USP], offN_8	±N'_8[USP] ← ±N'_8[USP]∨off N_8	10	15	12	17	4	•				110
	ORB N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] ∨off N_8	11	13	13	18	5	•				110
	ORB N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] ∨off N_8	11	13	13	18	5	•				110
1109	ORB rN, #N_8	rN ← rN ∨#N_8	5		9		2	•				111
	ORB PSWH, #N_8	PSWH ←PSWH ∨#N_8	5		—		2	•				111
	ORB PSWL, #N_8	PSWL ←PSWL ∨#N_8	5		—		2	•				111
	ORB off N'_8, #N_8	off N'_8 ← off N'_8 ∨#N_8	7		12		3	•				111
	ORB N'_8, #N_8	N'_8 ← N'_8 ∨#N_8	7		—		3	•				111
	ORB [DP], #N_8	[DP] ← [DP] ∨#N_8	7		11		2	•				111
	ORB ±N'_8[USP], #N_8	±N'_8[USP] ← ±N'_8[USP]∨#N_8	8		13		3	•				111
	ORB N_{16}[X1], #N_8	N_{16}[X1] ← N_{16}[X1] ∨#N_8	9		13		4	•				111
	ORB N_{16}[X2], #N_8	N_{16}[X2] ← N_{16}[X2] ∨#N_8	9		13		4	•				111

TABLE 3-26. Logical Calculation Instructions (5)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
1116	XOR A, #N_{16}	A ← A ∨ N_{16}	6	—			3	•				172
	XOR A, erN	A ← A ∨ erN	4	8			2	•				172
	XOR A, DP	A ← A ∨ DP	4	—			2	•				172
	XOR A, X1	A ← A ∨ X1	4	—			2	•				172
	XOR A, X2	A ← A ∨ X2	4	—			2	•				172
	XOR A, USP	A ← A ∨ USP	4	—			2	•				172
	XOR A, SSP	A ← A ∨ SSP	4	—			2	•				172
	XOR A, LRB	A ← A ∨ LRB	4	—			2	•				172
	XOR A, off N_8	A ← A ∨ off N_8	4	9			2	•				172
	XOR A, N_8	A ← A ∨ N_8	6	—			3	•				172
	XOR A, [DP]	A ← A ∨ [DP]	6	10			2	•				172
	XOR A, ±N_8[USP]	A ← A ∨ ±N_8[USP]	7	11			3	•				172
	XOR A, N_{16}[X1]	A ← A ∨ N_{16}[X1]	8	12			4	•				172
	XOR A, N_{16}[X2]	A ← A ∨ N_{16}[X2]	8	12			4	•				172
1132	XOR erN, A	erN ← erN ∨ A	5		13		2	•				173
	XOR DP, A	DP ← DP ∨ A	5		—		2	•				173
	XOR X1, A	X1 ← X1 ∨ A	5		—		2	•				173
	XOR X2, A	X2 ← X2 ∨ A	5		—		2	•				173
	XOR USP, A	USP ← USP ∨ A	5		—		2	•				173
	XOR SSP, A	SSP ← SSP ∨ A	5		—		2	•				173
	XOR LRB, A	LRB ← LRB ∨ A	5		—		2	•				173
	XOR off N_8, A	off N_8 ← off N_8 ∨ A	7		13		3	•				173
	XOR N_8, A	N_8 ← N_8 ∨ A	7		—		3	•				173
	XOR [DP], A	[DP] ← [DP] ∨ A	7		15		2	•				173
	XOR ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] ∨ A	8		13		3	•				173
	XOR N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] ∨ A	9		17		4	•				173
	XOR N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] ∨ A	9		17		4	•				173
1145	XOR erN, off N_8	erN ← erN ∨ off N_8	7	—	—	20	3	•				174
	XOR DP, off N_8	DP ← DP ∨ off N_8	7	13	—	—	3	•				174
	XOR X1, off N_8	X1 ← X1 ∨ off N_8	7	13	—	—	3	•				174
	XOR X2, off N_8	X2 ← X2 ∨ off N_8	7	13	—	—	3	•				174
	XOR USP, off N_8	USP ← USP ∨ off N_8	7	13	—	—	3	•				174
	XOR SSP, off N_8	SSP ← SSP ∨ off N_8	7	13	—	—	3	•				174
	XOR LRB, off N_8	LRB ← LRB ∨ off N_8	7	13	—	—	3	•				174
	XOR off N'_8, off N_8	off N'_8 ← off N'_8 ∨ off N_8	9	—	—	23	4	•				174
	XOR N'_8, off N_8	N'_8 ← N'_8 ∨ off N_8	9	18	—	—	4	•				174
	XOR [DP], off N_8	[DP] ← [DP] ∨ off N_8	9	18	13	22	3	•				174
	XOR ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP] ∨ off N_8	10	19	14	23	4	•				174
	XOR N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] ∨ off N_8	11	20	15	24	5	•				174
	XOR N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] ∨ off N_8	11	20	15	24	5	•				174
1158	XOR erN, #N_{16}	erN ← erN ∨ #N_{16}	6		17		4	•				175
	XOR DP, #N_{16}	DP ← DP ∨ #N_{16}	8		—		4	•				175
	XOR X1, #N_{16}	X1 ← X1 ∨ #N_{16}	8		—		4	•				175
	XOR X2, #N_{16}	X2 ← X2 ∨ #N_{16}	8		—		4	•				175
	XOR USP, #N_{16}	USP ← USP ∨ #N_{16}	8		—		4	•				175
	XOR SSP, #N_{16}	SSP ← SSP ∨ #N_{16}	8		—		4	•				175
	XOR LRB, #N_{16}	LRB ← LRB ∨ #N_{16}	6		—		4	•				175
	XOR off N_8, #N_{16}	off N_8 ← off N_8 ∨ #N_{16}	10		20		5	•				175
	XOR N_8, #N_{16}	N_8 ← N_8 ∨ #N_{16}	10		—		5	•				175
	XOR [DP], #N_{16}	[DP] ← [DP] ∨ #N_{16}	10		19		4	•				175
	XOR ±N_8[USP], #N_{16}	±N_8[USP] ← ±N_8[USP] ∨ #N_{16}	11		20		5	•				175

TABLE 3-27. Logical Calculation Instructions (6)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
1169	XOR N'_{16}[X1], #N_{16}	N'_{16}[X1] ← N'_{16}[X1] ∨ #N_{16}	12		21		6	*				175
	XOR N'_{16}[X2], #N_{16}	N'_{16}[X2] ← N'_{16}[X2] ∨ #N_{16}	12		21		6	*				175
1171	XORB A, #N_8	A_L ← A_L ∨ #N_8	4	—				*				176
	XORB A, rN	A_L ← A_L ∨ rN	4	6				*				176
	XORB A, PSWH	A_L ← A_L ∨ PSWH	4	—				*				176
	XORB A, PSWL	A_L ← A_L ∨ PSWL	4	—				*				176
	XORB A, off N_8	A_L ← A_L ∨ off N_8	4	7				*				176
	XORB A, N_8	A_L ← A_L ∨ N_8	6	—				*				176
	XORB A, [DP]	A_L ← A_L ∨ [DP]	6	—				*				176
	XORB A, ±N_8[USP]	A_L ← A_L ∨ ±N_8[USP]	7	9				*				176
	XORB A, N_{16}[X1]	A_L ← A_L ∨ N_{16}[X1]	6	10				*				176
	XORB A, N_{16}[X2]	A_L ← A_L ∨ N_{16}[X2]	6	10				*				176
1181	XORB rN, A	rN ← rN ∨ A_L	5		9		2	*				177
	XORB PSWH, A	PSWH ← PSWH ∨ A_L	5		—		2	*				177
	XORB PSWL, A	PSWL ← PSWL ∨ A_L	5		—		2	*				177
	XORB off N_8, A	off N_8 ← off N_8 ∨ A_L	7		12		3	*				177
	XORB N_8, A	N_8 ← N_8 ∨ A_L	7		—		3	*				177
	XORB [DP], A	[DP] ← [DP] ∨ A_L	7		11		2	*				177
	XORB ±N_8[USP], A	±N_8[USP] ← ±N_8[USP] ∨A_L	8		12		3	*				177
	XORB N_{16}[X1], A	N_{16}[X1] ← N_{16}[X1] ∨A_L	9		13		4	*				177
	XORB N_{16}[X2], A	N_{16}[X2] ← N_{16}[X2] ∨A_L	9		13		4	*				177
1190	XORB rN, off N_8	rN ← rN ∨ off N_8	7	—	—	14	3	*				178
	XORB PSWH, off N_8	PSWH ←PSWH∨off N_8	7	12	—	—	3	*				178
	XORB PSWL, off N_8	PSWL ←PSWL∨off N_8	7	12	—	—	3	*				178
	XORB off N'_8, off N_8	off N'_8 ← off N'_8 ∨ off N_8	9	—	—	17	4	*				178
	XORB N'_8, off N_8	N'_8 ← N'_8 ∨ off N_8	9	14	—	—	4	*				178
	XORB [DP], off N_8	[DP] ← [DP] ∨ off N_8	9	14	11	13	3	*				178
	XORB ±N'_8[USP], off N_8	±N'_8[USP] ← ±N'_8[USP]∨off N_8	10	15	12	17	4	*				178
	XORB N_{16}[X1], off N_8	N_{16}[X1] ← N_{16}[X1] ∨ off N_8	11	13	13	16	5	*				178
	XORB N_{16}[X2], off N_8	N_{16}[X2] ← N_{16}[X2] ∨ off N_8	11	13	13	16	5	*				178
1199	XORB rN, #N_8	rN ← rN ∨ #N_8	6		11		3	*				179
	XORB PSWH, #N_8	PSWH ←PSWH ∨ #N_8	6		—		3	*				179
	XORB PSWL, #N_8	PSWL ← PSWL ∨ #N_8	6		—		3	*				179
	XORB off N'_8, #N_8	off N'_8 ← off N'_8 ∨ #N_8	8		14		4	*				179
	XORB N'_8, #N_8	N'_8 ← N'_8 ∨ #N_8	8		—		4	*				179
	XORB [DP], #N_8	[DP] ← [DP] ∨ #N_8	8		13		3	*				179
	XORB ±N'_8[USP], #N_8	±N'_8[USP] ← ±N'_8[USP]∨#N_8	9		14		4	*				179
	XORB N_{16}[X1], #N_8	N_{16}[X1] ← N_{16}[X1] ∨ #N_8	10		15		5	*				179
	XORB N_{16}[X2], #N_8	N_{16}[X2] ← N_{16}[X2] ∨ #N_8	10		15		5	*				179

TABLE 3-28. Comparison Instructions (1)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1288	CMP A, #N_{16}	A - #N_{16}	6	—			3	*	*			35
	CMP A, erN	A - erN	3	7			1	*	*			35
	CMP A, DP	A - DP	4	—			2	*	*			35
	CMP A, X1	A - X1	4	—			2	*	*			35
	CMP A, X2	A - X2	4	—			2	*	*			35
	CMP A, USP	A - USP	4	—			2	*	*			35
	CMP A, SSP	A - SSP	4	—			2	*	*			35
	CMP A, LRB	A - LRB	4	—			2	*	*			35
	CMP A, off N_8	A - off N_8	4	9			2	*	*			35
	CMP A, N_8	A - N_8	6	—			3	*	*			35
	CMP A, [DP]	A - [DP]	6	10			2	*	*			35
	CMP A, ±N_8[USP]	A - ±N_8[USP]	7	11			3	*	*			35
	CMP A, N_{16}[X1]	A - N_{16}[X1]	6	12			4	*	*			35
	CMP A, N_{16}[X2]	A - N_{16}[X2]	6	12			4	*	*			35
1222	CMP erN, A	erN - A	5		13		2	*	*			36
	CMP DP, A	DP - A	5		—		2	*	*			36
	CMP X1, A	X1 - A	5		—		2	*	*			36
	CMP X2, A	X2 - A	5		—		2	*	*			36
	CMP USP, A	USP - A	5		—		2	*	*			36
	CMP SSP, A	SSP - A	5		—		2	*	*			36
	CMP LRB, A	LRB - A	5		—		2	*	*			36
	CMP off N_8, A	off N_8 - A	7		13		3	*	*			36
	CMP N_8, A	N_8 - A	7		—		3	*	*			36
	CMP [DP], A	[DP] - A	7		15		2	*	*			36
	CMP ±N_8[USP], A	±N_8[USP]- A	8		13		3	*	*			36
	CMP N_{16}[X1], A	N_{16}[X1]- A	9		17		4	*	*			36
	CMP N_{16}[X2], A	N_{16}[X2]- A	9		17		4	*	*			36
1235	CMP erN, off N_8	erN - off N_8	7	—	—	20	3	*	*			37
	CMP DP, off N_8	DP - off N_8	7	13	—	—	3	*	*			37
	CMP X1, off N_8	X1 - off N_8	7	13	—	—	3	*	*			37
	CMP X2, off N_8	X2 - off N_8	7	13	—	—	3	*	*			37
	CMP USP, off N_8	USP - off N_8	7	13	—	—	3	*	*			37
	CMP SSP, off N_8	SSP - off N_8	7	13	—	—	3	*	*			37
	CMP LRB, off N_8	LRB - off N_8	7	13	—	—	3	*	*			37
	CMP off N'_8, off N_8	off N'_8 - off N_8	9	—	—	23	4	*	*			37
	CMP N'_8, off N_8	N'_8 - off N_8	9	16	—	—	4	*	*			37
	CMP [DP], off N_8	DP] - off N_8	9	16	13	22	3	*	*			37
	CMP ±N'_8[USP], off N_8	±N'_8[USP]- off N_8	10	19	14	23	4	*	*			37
	CMP N_{16}[X1], off N_8	N_{16}[X1] - off N_8	11	20	15	24	5	*	*			37
	CMP N_{16}[X2], off N_8	N_{16}[X2] - off N_8	11	20	15	24	5	*	*			37
1248	CMP erN, #N_{16}	erN - #N_{16}	8		17		4	*	*			38
	CMP DP, #N_{16}	DP - #N_{16}	8		—		4	*	*			38
	CMP X1, #N_{16}	X1 - #N_{16}	8		—		4	*	*			38
	CMP X2, #N_{16}	X2 - #N_{16}	8		—		4	*	*			38
	CMP USP, #N_{16}	USP - #N_{16}	8		—		4	*	*			38
	CMP SSP, #N_{16}	SSP - #N_{16}	8		—		4	*	*			38
	CMP LRB, #N_{16}	LRB - #N_{16}	8		—		4	*	*			38
	CMP off N_8, #N_{16}	off N_8 - #N_{16}	10		20		5	*	*			38
	CMP N_8, #N_{16}	N_8 - #N_{16}	10		—		5	*	*			38
	CMP [DP], #N_{16}	[DP] - #N_{16}	10		19		4	*	*			38
	CMP ±N_8[USP], #N_{16}	±N_8[USP]- #N_{16}	11		20		5	*	*			38
	CMP N'_{16}[X1], #N_{16}	N'_{16}[X1]- #N_{16}	12		21		6	*	*			38
	CMP N'_{16}[X2], #N_{16}	N'_{16}[X2] - #N_{16}	12		21		6	*	*			38

TABLE 3-29. Comparison Instructions (2)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1261	CMPB A, #N_8	A_L - #N_8	4	—			2	•	•			39
	CMPB A, rN	A_L - rN	3	5			1	•	•			39
	CMPB A, PSWH	A_L - PSWH	4	—			2	•	•			39
	CMPB A, PSWL	A_L - PSWL	4	—			2	•	•			39
	CMPB A, off N_8	A_L - off N_8	4	7			2	•	•			39
	CMPB A, N_8	A_L - N_8	6	—			3	•	•			39
	CMPB A, [DP]	A_L - [DP]	6	—			2	•	•			39
	CMPB A, ±N_8[USP]	A_L - ±N_8[USP]	7	9			3	•	•			39
	CMPB A, N_{16}[X1]	A_L - N_{16}[X1]	8	10			4	•	•			39
	CMPB A, N_{16}[X2]	A_L - N_{16}[X2]	8	10			4	•	•			39
1271	CMPB rN, A	rN - A_L	5		9		2	•	•			40
	CMPB PSWH, A	PSWH - A_L	5		—		2	•	•			40
	CMPB PSWL, A	PSWL - A_L	5		—		2	•	•			40
	CMPB off N_8, A	off N_8 - A_L	7		12		3	•	•			40
	CMPB N_8, A	N_8 - A_L	7		—		3	•	•			40
	CMPB [DP], A	[DP] - A_L	7		11		2	•	•			40
	CMPB ±N_8[USP], A	±N_8[USP]- A_L	8		12		3	•	•			40
	CMPB N_{16}[X1], A	N_{16}[X1]- A_L	9		13		4	•	•			40
	CMPB N_{16}[X2], A	N_{16}[X2]- A_L	9		13		4	•	•			40
1280	CMPB rN, off N_8	rN - off N_8	7	—	—	14	3	•	•			41
	CMPB PSWH, off N_8	PSWH-off N_8	7	12	—	—	3	•	•			41
	CMPB PSWL, off N_8	SWL-off N_8	7	12	—	—	3	•	•			41
	CMPB off N'_8, off N_8	off N'_8 - off N_8	9	—	—	17	4	•	•			41
	CMPB N'_8, off N_8	N'_8 - off N_8	9	14	—	—	4	•	•			41
	CMPB [DP], off N_8	[DP] - off N_8	9	14	11	13	3	•	•			41
	CMPB ±N'_8[USP], off N_8	±N'_8[USP]- off N_8	10	15	12	17	4	•	•			41
	CMPB N_{16}[X1], off N_8	N_{16}[X1]- off N_8	11	13	13	18	5	•	•			41
	CMPB N_{16}[X2], off N_8	N_{16}[X2] - off N_8	11	13	13	18	5	•	•			41
1289	CMPB rN, #N_8	rN - #N_8	6		11		3	•	•			42
	CMPB PSWH, #N_8	PSWH - #N_8	6		—		3	•	•			42
	CMPB PSWL, #N_8	PSWL - #N_8	6		—		3	•	•			42
	CMPB off N'_8, #N_8	off N'_8 - #N_8	8		14		4	•	•			42
	CMPB N'_8, #N_8	N'_8 - #N_8	8		—		4	•	•			42
	CMPB [DP], #N_8	[DP] - #N_8	8		13		3	•	•			42
	CMPB ±N'_8[USP], #N_8	±N'_8[USP]- #N_8	9		14		4	•	•			42
	CMPB N_{16}[X1], #N_8	N_{16}[X1] - #N_8	10		15		5	•	•			42
	CMPB N_{16}[X2], #N_8	N_{16}[X2]- #N_8	10		15		5	•	•			42

TABLE 3-30. Decimal Adjust Instructions

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1298	DAA	IF ($A_{L3-0} \geq 10$) OR (HC = 1) THEN $A_L \leftarrow A_L + 6$ IF ($A_{L7-4} \geq 10$) OR (C = 1) THEN $A_{L7-4} \leftarrow A_{L7-4} + 6$	6				1		*	*		49
1299	DAS	IF ($A_{L3-0} \geq 10$) OR (HC = 1) THEN $A_L \leftarrow A_L - 6$ IF ($A_{L7-4} \geq 10$) OR (C = 1) THEN $A_{L7-4} \leftarrow A_{L7-4} - 6$ HC ← 0	6				1		*	*		52

TABLE 3-31. Sign Extend Instructions

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1300	EXTND	$A_{15-8} \leftarrow A_7$ DD ← 1	2				1				1	59

TABLE 3-32. Bit Operation Instructions (1)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
1301	SBR rN	IF obj.bit(A_{2-0}) = 0	7		13		2	•				137
	SBR PSWH	THEN Z ←1	7		—		2	•				137
	SBR PSWL	ELSE Z ←0	7		—		2	•				137
	SBR off N_8		9		16		3	•				137
	SBR N_8	obj.bit(A_{2-0}) ←1	9		—		3	•				137
	SBR [DP]		9		15		2	•				137
	SBR ±N_8[USP]		10		16		3	•				137
	SBR N_{16}[X1]		11		17		4	•				137
	SBR N_{16}[X2]		11		17		4	•				137
1310	RBR rN	IF obj.bit(A_{2-0}) = 0	7		13		2	•				116
	RBR PSWH	THEN Z ←1	7		—		2	•				116
	RBR PSWL	ELSE Z ←0	7		—		2	•				116
	RBR off N_8		9		16		3	•				116
	RBR N_8	obj.bit(A_{2-0}) ←0	9		—		3	•				116
	RBR [DP]		9		15		2	•				116
	RBR ±N_8[USP]		10		16		3	•				116
	RBR N_{16}[X1]		11		17		4	•				116
	RBR N_{16}[X2]		11		17		4	•				116
1319	TBR rN	IF obj.bit(A_{2-0}) = 0	4		6		2	•				167
	TBR PSWH	THEN Z ←1	4		—		2	•				167
	TBR PSWL	ELSE Z ←0	4		—		2	•				167
	TBR off N_8		6		9		3	•				167
	TBR N_8		6		—		3	•				167
	TBR [DP]		6		8		2	•				167
	TBR ±N_8[USP]		7		9		3	•				167
	TBR N_{16}[X1]		8		10		4	•				167
	TBR N_{16}[X2]		8		10		4	•				167
1328	MBR C, rN		5	7			2		•			80
	MBR C, PSWH		5	—			2		•			80
	MBR C, PSWL		5	—			2		•			80
	MBR C, off N_8		7	10			3		•			80
	MBR C, N_8	C ← obj.bit(A_{2-0})	7	—			3		•			80
	MBR C, [DP]		7	9			2		•			80
	MBR C, ±N_8[USP]		8	10			3		•			80
	MBR C, N_{16}[X1]		9	11			4		•			80
	MBR C, N_{16}[X2]		9	11			4		•			80
1337	MBR rN, C		10		16		2					81
	MBR PSWH, C		10		—		2					81
	MBR PSWL, C		10		—		2					81
	MBR off N_8, C		12		19		3					81
	MBR N_8, C	obj.bit(A_{2-0}) ← C	12		—		3					81
	MBR [DP], C		12		18		2					81
	MBR ±N_8[USP], C		13		19		3					81
	MBR N_{16}[X1], C		14		20		4					81
	MBR N_{16}[X2], C		14		20		4					81
1346	SB rN.bit	IF obj.bit = 0	7		13		2	•				128
	SB PSWH.bit	THEN Z ←1	7		—		2	•				128
	SB PSWL.bit	ELSE Z ←0	7		—		2	•				128
	SB off N_8.bit		9		16		3	•				128
	SB N_8.bit	obj.bit ←1	9		—		3	•				128
	SB [DP].bit		9		15		2	•				128
	SB ±N_8[USP].bit		10		16		3	•				128
	SB N_{16}[X1].bit		11		17		4	•				128
	SB N_{16}[X2].bit		11		17		4	•				128

TABLE 3-33. Bit Manipulation Instructions (2)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
1355	RB rN.bit	IF obj.bit = 0	7		13		2	*				115
	RB PSWH.bit	THEN Z ← 1	7		—		2	*				115
	RB PSWL.bit	ELSE Z ← 0	7		—		2	*				115
	RB off N_8.bit		9		16		3	*				115
	RB N_8.bit	obj.bit ← 0	9		—		3	*				115
	RB [DP].bit		9		15		2	*				115
	RB ±N_8[USP].bit		10		16		3	*				115
	RB N_{16}[X1].bit		11		17		4	*				115
	RB N_{16}[X2].bit		11		17		4	*				115
1364	MB C, rN.bit		5	7			2		*			78
	MB C, PSWH.bit		5	—			2		*			78
	MB C, PSWL.bit		5	—			2		*			78
	MB C, off N_8.bit		7	10			3		*			78
	MB C, N_8.bit	C ← obj.bit	7	—			3		*			78
	MB C, [DP].bit		7	9			2		*			78
	MB C, ±N_8[USP].bit		8	10			3		*			78
	MB C, N_{16}[X1].bit		9	11			4		*			78
	MB C, N_{16}[X2].bit		9	11			4		*			78
1373	MB rN.bit, C		10		16		2					79
	MB PSWH.bit, C		10		—		2					79
	MB PSWL.bit, C		10		—		2					79
	MB off N_8.bit, C		12		19		3					79
	MB N_8.bit, C	obj.bit ← C	12		—		3					79
	MB [DP].bit, C		12		18		2					79
	MB ±N_8[USP].bit, C		13		19		3					79
	MB N_{16}[X1].bit, C		14		20		4					79
	MB N_{16}[X2].bit, C		14		20		4					79

TABLE 3-34. Jump Call Instructions (1)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	CF	HC	DD	Page
1382	SJ address	PC ← address where PC*-128 ≤ address ≤ PC*+127 PC* = Head address of the instruction after the SJ instruction = (address of first byte of SJ instruction) + 2	8				2					141
1383	J address	PC ← address (16 bits)	7				3					62
1384	J [erN]	PC ← [erN]	6		10		2					63
	J [DP]	PC ← [DP]	6		—		2					63
	J [X1]	PC ← [X1]	6		—		2					63
	J [X2]	PC ← [X2]	6		—		2					63
	J [USP]	PC ← [USP]	6		—		2					63
	J [SSP]	PC ← [SSP]	6		—		2					63
	J [LRB]	PC ← [LRB]	6		—		2					63
	J [off N_8]	PC ← [off N_8]	8		13		3					63
	J [N_8]	PC ← [N_8]	8		—		3					63
	J [[DP]]	PC ← [[DP]]	8		12		2					63

TABLE 3-35. Jump Call Instructions (2)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1395	J [±N_8[USP]]	PC ← [±N_8[USP]]	9		13		3					63
	J [N_{16}[X1]]	PC ← [N_{16}[X1]]	10		14		4					63
	J [N_{16}[X2]]	PC ← [N_{16}[X2]]	10		14		4					63
1397	JC EQ, address	IF Z = 1 THEN PC ←address	true 8	false 4			2					65
	JC NE, address	IF Z = 0 THEN PC ←address	8	4			2					65
	JC LT, address	IF C = 1 THEN PC ←address	8	4			2					65
	JC LE, address	IF Z = 1 or C = 1 THEN PC ←address	8	4			2					65
	JC GT, address	IF Z = 0 and C = 0 THEN PC ←address	8	4			2					65
	JC GE, address	IF C = 0 THEN PC ←address	8	4			2					65
		where PC*-128 ≤ address ≤ PC*+127 PC* = Head address of the instruction after the JC instruction = (address of first byte of JC instruction) + 2										
1403	JBR off N_8.bit, address	IF (off N_8.bit = 0) THEN PC ←address	true 10		false 13		3					64
		where PC*-128 ≤ address ≤ PC*+127 PC* = Head address of the instruction after the JBR instruction = (address of first byte of JBR instruction) + 3	true 6		false 9							
1404	JBS off N_8.bit, address	IF (off N_8.bit = 1) THEN PC ←address	true 10		false 13		3					65
		where PC*-128 ≤ address ≤ PC*+127 PC* = Head address of the instruction after the JBS instruction = (address of first byte of JBS instruction) + 3	true 6		false 9							
1405	JRNZ DP, address	DPL ← DP_L - 1 IF DP_L ≠ 0 THEN PC ←address	11 DP_L≠0				2					69
		where PC*-128 ≤ address ≤ PC*+127 PC* = Head address of the instruction after the JRNZ instruction = (address of first byte of JRNZ instruction) + 2	7 DP_L=0									

TABLE 3-36. Jump Call Instructions (3)

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1406	SCAL address	(SSP) ← PC + 2 SSP ← SSP - 2 SF ← 0 PC ←address where PC*-128 ≤ address ≤ PC*+127 PC* =Head address of the instruction after the SCAL instruction = (address of first byte of SCAL instruction) + 2	9		13		2					139
1407	CAL address	(SSP) ← PC + 3 SSP ← SSP - 2 SF ← 0 PC ←address (16 bits)	9		13		3					29
1408	CAL [erN] CAL [DP] CAL [X1] CAL [X2] CAL [USP] CAL [SSP] CAL [LRB] CAL [off N_8] AL [N_8] CAL [[DP] CAL [±N_8[USP]] CAL [N_{16}[X1]] CAL [N_{16}[X2]]	(SSP) ← PC + n SSP ← SSP - 2 SF ← 0 PC ←obj (16 bits) n is the number of bytes in the instruction and varies with the addressing objects.	8 8 8 8 8 8 8 10 10 10 11 12 12	12 12 12 12 12 12 12 14 14 14 15 16 16	12 — — — — — — 15 — 14 15 16 16	16 — — — — — — 19 — 18 19 20 20	2 2 2 2 2 2 2 3 3 2 3 4 4					30 30 30 30 30 30 30 30 30 30 30 30 30
1421	VCAL table-address	(SSP) ← PC + 1 SSP ← SSP - 2 SF ← 0 PC ←(table-address) where table-address must be an even-numbered address in the VCAL table area (28_H-37_H)	11		15		1					168
1422	RT	SSP ← SSP + 2 PC ← (SSP) SF ← 0	7		11		1					126
1423	RTI	SSP ← SSP + 2 PSW ← (SSP) SSP ← SSP + 2 LRB ← (SSP) SSP ← SSP + 2 A ← (SSP) SSP ← SSP + 2 PC ← (SSP)	15		31		1					127

TABLE 3-37. Other Instructions

*1 - the first operand
*2 - the second operand

No.	Mnemonics	Operation	Int *1 Int *2 Cyc	Int *1 Ext *2 Cyc	Ext *1 Int *2 Cyc	Ext *1 Ext *2 Cyc	Byte no.	Flags ZF	Flags CF	Flags HC	Flags DD	Page
1424	SC	C ← 1	2				1		1			138
1425	RC	C ← 0	2				1		0			117
1426	BRK	SYSTEM RESET PC ← (vector table 002_H)	13				1					28
1427	NOP		2				1					103