PanaXSeries

Bringing Tomorrow to Today

MICROCOMPUTER MN101C00

MN101C00 Series LSI User Manual

Panasonic

Request for your special attention and precautions in using the technical information and semiconductors described in this book

(1) An export permit needs to be obtained from the competent authorities of the Japanese Government if any of the products or technologies described in this book and controlled under the "Foreign Exchange and Foreign Trade Control Law" is to be exported or taken out of Japan.

(2) The contents of this book are subject to change without notice in matters of improved function. When finalizing your design,therefore,ask for the most up-to-date version in advance in order to check for any changes.

(3) We are not liable for any damage arising out of the use of the contents of this book, or for any infringment of patents or any other rights owned by a third party.

(4) No part of this book may be reprinted or reproduced by any means without written permission from our company.

(5) This book deals with standard specifications. Ask for the latest individual Product Standards or Specifications in advance for more detailed information required for your design,purchasing and applications.

If you have any inquiries or questions about this book or our semiconductors,please contact one of our sales offices listed at the back of this book or Matsushita Electronics Corporation's Sales Department.

Reading This Manual

■ Topics Covered by this Manual

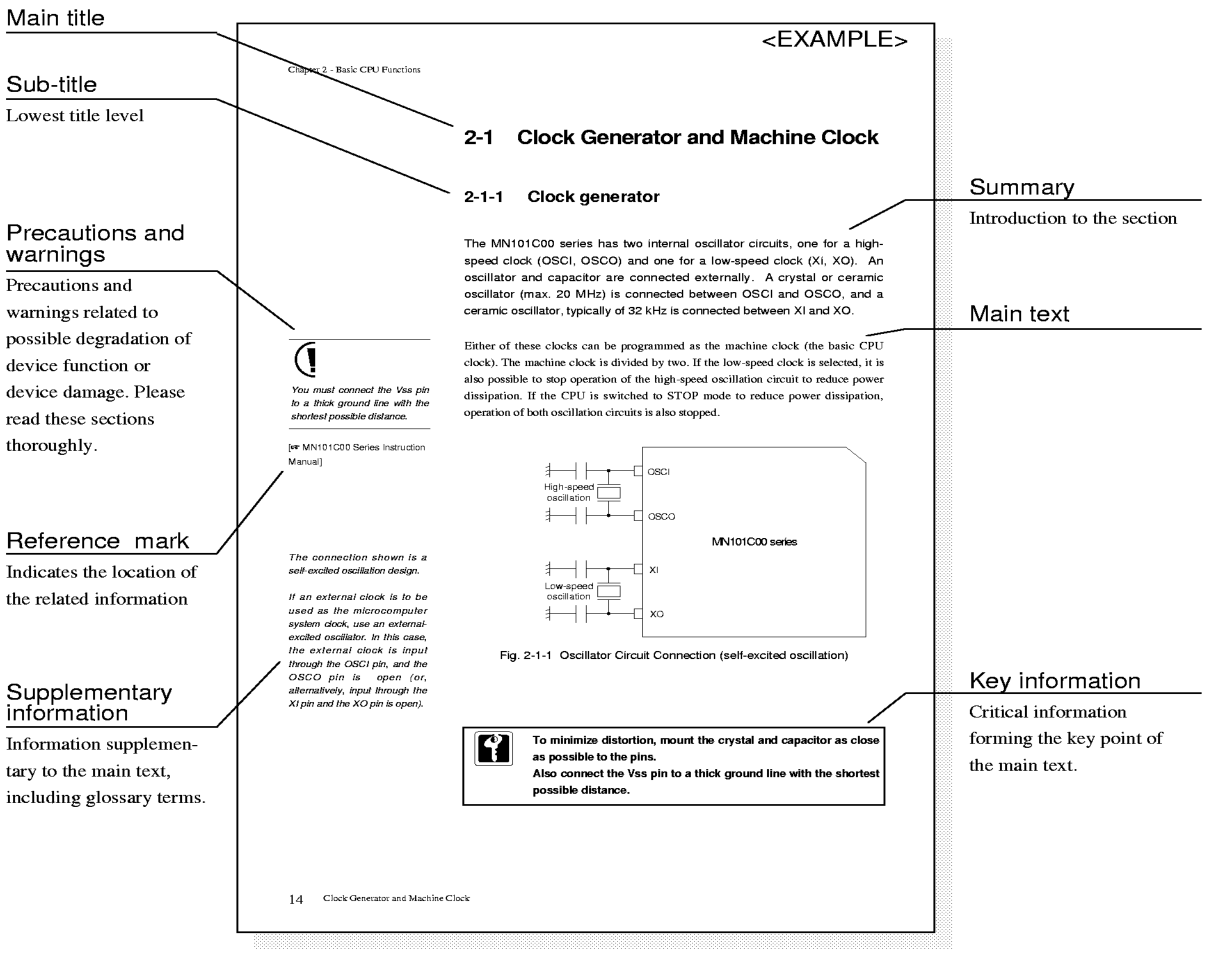
This manual describes the standard specifications common to the MN101C00 Series, but there are products to which not all of these specifications apply. Some products in this series may have peripheral functions or pins not covered in this manual. When using these Panasonic products, please be sure to verify the precise specifications, including the applicable content of this manual.

■ General Outline of the Manual

Chapter 1 introduces an outline of the hardware, the hardware configuration and basic specifications. Chapter 2 primarily covers the operation and functions of the hardware block.

■ Organization of the Manual

The various items in this manual include a title, summary, main text, supplementary information, and precautions and warnings. The layout and definitions are indicated below.



Main title

Sub-title
Lowest title level

Precautions and warnings
Precautions and warnings related to possible degradation of device function or device damage. Please read these sections thoroughly.

Reference mark
Indicates the location of the related information

Supplementary information
Information supplementary to the main text, including glossary terms.

Summary
Introduction to the section

Main text

Key information
Critical information forming the key point of the main text.

<EXAMPLE>

Chapter 2 - Basic CPU Functions

2-1 Clock Generator and Machine Clock

2-1-1 Clock generator

The MN101C00 series has two internal oscillator circuits, one for a high-speed clock (OSCI, OSCO) and one for a low-speed clock (Xi, XO). An oscillator and capacitor are connected externally. A crystal or ceramic oscillator (max. 20 MHz) is connected between OSCI and OSCO, and a ceramic oscillator, typically of 32 kHz is connected between XI and XO.

Either of these clocks can be programmed as the machine clock (the basic CPU clock). The machine clock is divided by two. If the low-speed clock is selected, it is also possible to stop operation of the high-speed oscillation circuit to reduce power dissipation. If the CPU is switched to STOP mode to reduce power dissipation, operation of both oscillation circuits is also stopped.

You must connect the Vss pin to a thick ground line with the shortest possible distance.

[☞ MN101C00 Series Instruction Manual]

The connection shown is a self-excited oscillation design.

If an external clock is to be used as the microcomputer system clock, use an external-excited oscillator. In this case, the external clock is input through the OSCI pin, and the OSCO pin is open (or, alternatively, input through the XI pin and the XO pin is open).

Fig. 2-1-1 Oscillator Circuit Connection (self-excited oscillation)

To minimize distortion, mount the crystal and capacitor as close as possible to the pins.
Also connect the Vss pin to a thick ground line with the shortest possible distance.

14 Clock Generator and Machine Clock

■ Search method

To locate the required information rapidly, there are four methods of searching in this manual.

(1) Refer to the index at the front of the manual to locate the beginning of each section.
(2) Refer to the Contents at the front of the manual to locate titles.
(3) Refer to the List of Figures and Tables at the front of the manual to locate titles of illustrations and tables.
(4) The Chapter name is given at the top of every page, and the main title at the bottom of every page. This makes it possible to scan through the manual quickly to locate a desired section.

■ Related manuals

The following manuals are available for the MN10200 Series Linear Adrressing Version:

● "MN101C00 Series Instruction Manual"
<Describes the instruction set>
● "MN101C00 Series Cross-assembler User's Manual"
<Describes the assembler syntax and notation>
● "MN101C00 Series C Compiler User's Manual: Usage Guide"
<Describes the installation, the commands, and options of the C Compiler>
● "MN101C00 Series C Compiler User's Manual: Language Description"
<Describes the syntax of the C Compiler>
● "MN101C00 Series C Compiler User's Manual: Library Reference"
<Describes the the standard library of the C Compiler>
● "MN101C00 Series C Source Code Debugger User's Manual"
<Describes the use of the C source code debugger>
● "MN101C00 Series PanaX Series Installation Manual"
<Describes the installation of the C compiler, cross-assembler and C source code debugger and the procedure for bringing up the in-circuit emulator>

■ Inquiries and comments

Please direct any comments, suggestions or inquiries to your closest semiconductor design center. A list of addresses is provided at the rear of this manual for your convenience.

Table of Contents
List of Figures and Tables

Contents

List of Figures and Tables

List of Figures

List of Tables

Chapter 1 - General Description

1

1-1 General Description

1-1-1 Introduction

The MN101C00 series features a streamlined, high-performance architecture. It optimizes the overall performance of the hardware and software while maintaining ease of use and top cost performance. It offers

- The high cost performance demanded of microcomputers for embedded systems.
- Extensive support for program development (assembler or C) and execution environments.

It adopts a programming model based on an instruction set and opcodes designed to minimize object code sizes while responding to the demand for a more efficient program development environment supporting development in the high-level programming language, C.

1-1-2 Concept

- Microcomputers that deliver high cost performance while supporting development in the C programming language.
- Source code level portability to Panasonic's 16-bit MN10200 series.
- ASSP microcomputers offering flexible peripheral expansion for use in system integration.

1-1-3 Applications

Embedded systems for use in audiovisual equipment, home appliances, home information products, and a broad range of other applications.

1-1-4 Features

The MN101C00 series brings to embedded microcomputer applications flexible, optimized hardware configurations and a simple, efficient instruction set for both economy and speed. Specific features include the following:

1. Minimized code sizes with instruction lengths based on 4-bit increments
   The series keeps code sizes down by adopting a basic instruction length of one byte and a half-byte instruction approach using a program counter capable of addressing instructions in 4-bit increments. As a result, the series minimizes code sizes in spite of its simple instruction set limiting data transfers to and from memory to load/store operations.

2. Minimum execution time of one cycle
   This cycle is 100 ns at 20-MHz oscillation.

3. Minimized register set that simplifies the architecture and supports C code development
   The instruction set is based on a thorough analysis of the code generated by C compilers and that used in assembly language programming and of the tradeoffs between hardware scale and performance. As a result, the instruction set has been minimized to that supporting development in C and is notable for its simplicity.

1-1-5 Overview

This section describes the basic configuration and functions of the series.

■ Processor Status Word (PSW)

The processor status word (PSW) holds the operation result flags and interrupt mask level.

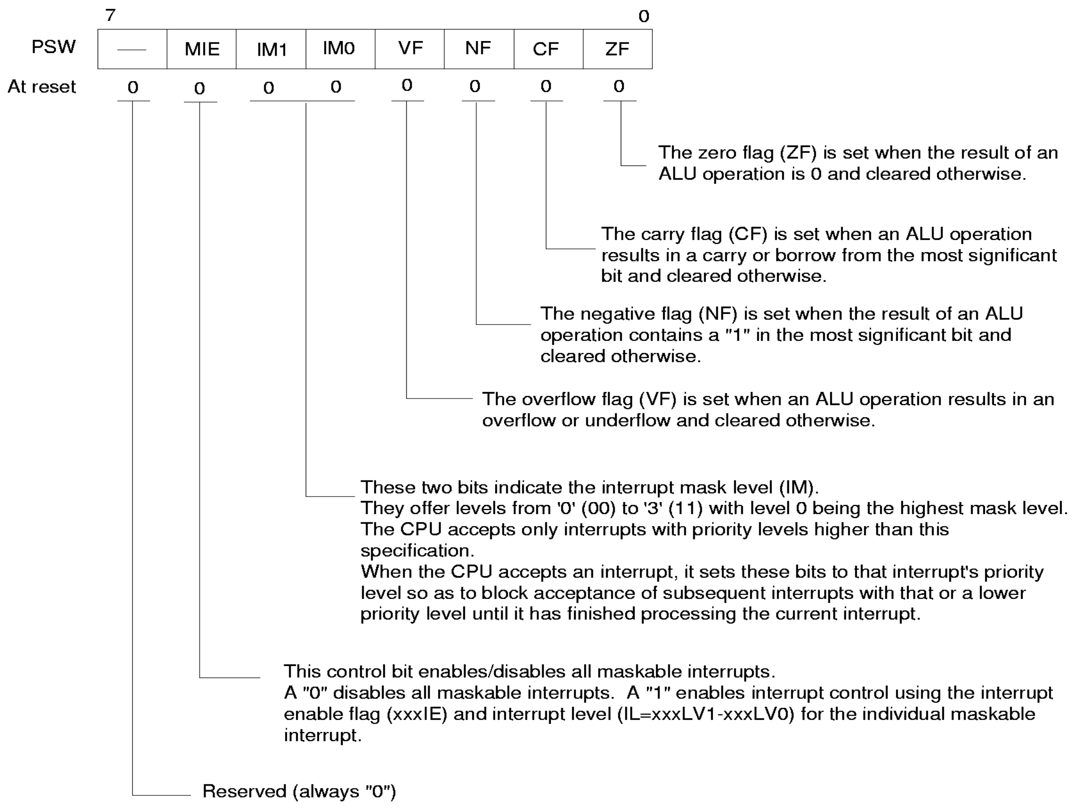

7
0
PSW
—
MIE
IM1
IM0
VF
NF
CF
ZF
At reset
0
0
0
0
0
0
0
0
The zero flag (ZF) is set when the result of an ALU operation is 0 and cleared otherwise.
The carry flag (CF) is set when an ALU operation results in a carry or borrow from the most significant bit and cleared otherwise.
The negative flag (NF) is set when the result of an ALU operation contains a "1" in the most significant bit and cleared otherwise.
The overflow flag (VF) is set when an ALU operation results in an overflow or underflow and cleared otherwise.
These two bits indicate the interrupt mask level (IM).
They offer levels from '0' (00) to '3' (11) with level 0 being the highest mask level.
The CPU accepts only interrupts with priority levels higher than this specification.
When the CPU accepts an interrupt, it sets these bits to that interrupt's priority level so as to block acceptance of subsequent interrupts with that or a lower priority level until it has finished processing the current interrupt.
This control bit enables/disables all maskable interrupts.
A "0" disables all maskable interrupts. A "1" enables interrupt control using the interrupt enable flag (xxxIE) and interrupt level (IL=xxxLV1-xxxLV0) for the individual maskable interrupt.
Reserved (always "0")

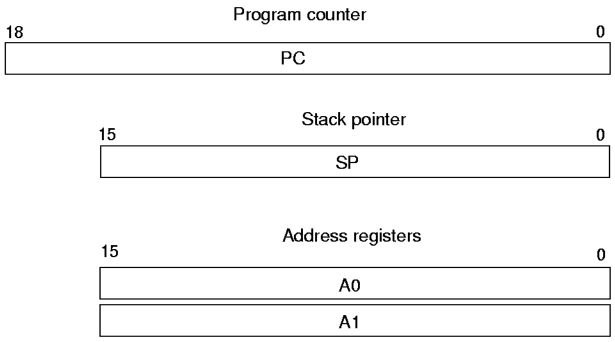
Fig. 1-1-1 Processor Status Word (PSW)

■ Internal Register, Memory and Special Function Register Configuration

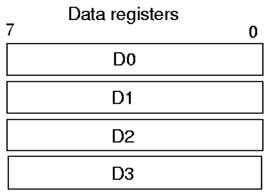
The 19-bit program counter holds the address of the program instruction being executed.

The stack pointer holds the top address of the stack area.

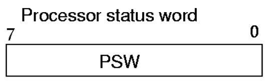
The address registers hold the locations of data in memory.



The data registers are general-purpose registers used to perform arithmetic or logic operations.



The processor status word (PSW) stores the operation result flags and the interrupt mask level.



Memory (ROM, RAM), special function register controlling peripheral functions, and I/O ports can be allocated to the same address space.

Internal control register*

Interrupt control register*

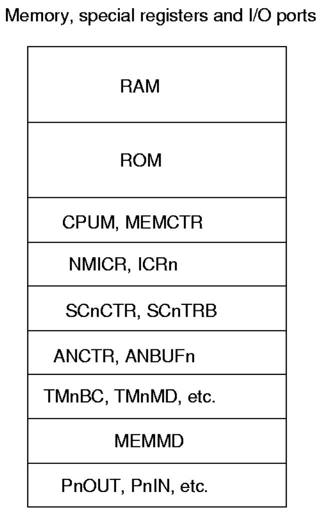
Serial interface*

A/D converter*

Timer counter*

Memory control*

I/O ports*



** This is a typical example. Actual memory, peripheral function, special register and I/O port configurations will vary with the product model.*

This is a typical example. Actual memory configuration will vary with the specific product.

■ Address Space

- The series supports an address space of 256 kilobytes.
- Instructions and data share the same address space, but data is limited to the first 64 kilobytes. Here data refers to both RAM data and ROM table data.
- The data address space includes a 256-byte area most efficiently accessed with abs8 addressing and a second 256-byte area most efficiently accessed with I/O short addressing.
- The internal RAM and special function registers are assigned to regions within the 16-kilobyte area X'00000'–X'03FFFF'.
- Up to 4 kilobytes of external RAM may be added starting at the address X'02F00'.
- There are processor modes for using external ROM and RAM together either with both the onboard ROM and RAM or with the onboard RAM alone. The MMOD pin and the EXMEM flag in the memory control register (MEMCTR) specify the mode.

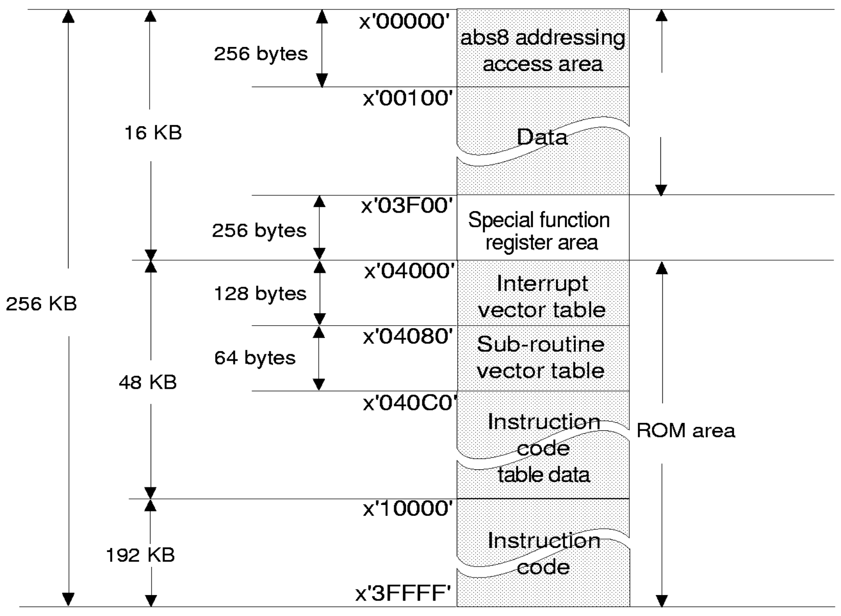


Fig. 1-1-2 Address Space

■ Interrupt Control Block

When an MN101C00 microcomputer accepts an interrupt, the hardware automatically executes a processing sequence that branches to the interrupt service routine whose starting address is specified in the interrupt vector table.

• Interrupt vectors

There are 31 interrupt vectors for specifying the starting addresses for interrupt service routines. They are initially assigned to ROM addresses X'04000'–X'0407B'.

Vector number	Address	Interrupt
0	X'04000'	Reset interrupt
1	X'04004'	Non-maskable interrupt
2 to 30	X'04008' to X'04078'	Peripheral function interrupt 1 to Peripheral function interrupt 29

Fig. 1-1-3 Interrupt Vector Table

1-2 Basic Specifications

This section introduces the basic specifications of the series.
For detailed specifications, see the manuals for each chip.

Table 1-2-1 Basic Specifications

Structure	Load/store architecture	
	Six registers	Data: 8-bitX4 Address: 16-bitX2
	Other	PC: 19-bit PSW: 8-bit SP: 16-bit
Instructions	Number of instructions	37
	Addressing modes	9
	Instruction length	Basic portion: 1 byte (min.) Extended portion: 0.5-byteXn(0≤n≤9)
Basic performance	Internal operating frequency (max)	10 MHz (20 MHz external oscillator)
	Instruction execution	Min. 1 cycle
	Inter-register operation	Min. 2 cycles
	Load/store	Min. 2 cycles
	Conditional branch	2 to 3 cycles
Pipeline	3-stage (instruction fetch, decode, execution)	
Address space	256 KB (max. 64 KB for data)	
	Instruction/data common space	
External bus	Address	18-bit (max.)
	Data	8-bit
	Minimum bus cycle	1 clock (100 ns at 20 MHz)
Interrupt	Vector interrupt	3 interrupt levels
Low-power dissipation mode	STOP mode	
	HALT mode	

1-3 Block Diagram

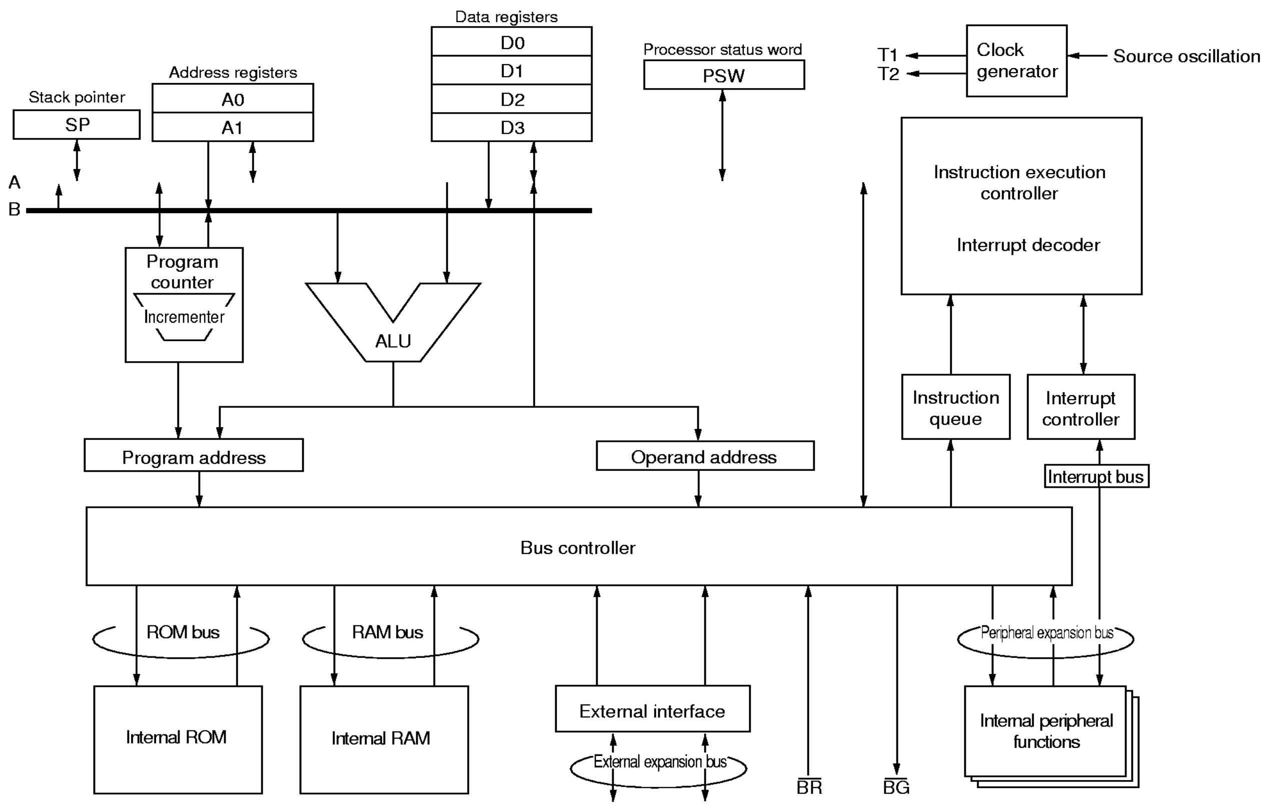


Fig. 1-3-1 Block Function Diagram

Table 1-3-1 Block Function Overview

Block	Function	See section
Clock generator	Uses a clock oscillator circuit driven by an external crystal or ceramic resonator to supply clock signals to CPU blocks.	(2-1)
Program counter	Generates addresses for the instructions to be inserted into the instruction queue. Normally incremented by sequencer indication, but may be set to branch destination address or ALU operation result when branch instructions or interrupts occur.	(2-3)
Instruction queue	Stores up to 2 bytes of pre-fetched instructions.	(2-2)
Instruction decoder	Decodes the instruction queue, sequentially generates the control signals needed for instruction execution, and executes the instruction by controlling the blocks within the chip.	
Instruction execution controller	Controls CPU block operations in response to the result decoded by the instruction decoder and interrupt requests.	
ALU	Executes arithmetic operations, logic operations, shift operations, and calculates operand addresses for register relative indirect addressing mode.	——
Internal ROM,RAM	Assigned to the execution program, data and stack region.	(2-9)
Address register (An)	Stores the addresses specifying memory for data transfer. Stores the base address for register relative indirect addressing mode.	(2-3)
Data register (Dn)	Holds data for operations,. Two 8-bit registers can be connected to form a 16-bit register.	
Interrupt controller	Detects interrupt requests from peripheral functions and requests CPU shift to interrupt processing.	(2-5)
Bus controller	Controls connection of CPU internal bus and CPU external bus. Includes bus usage arbitration function.	(2-9)
Internal peripheral functions	Includes peripheral functions (timer, serial, A/D converter, D/A converter, etc.). Peripheral functions vary with model.	——

1-4 Addressing Modes

The MN101C00 series supports the nine addressing modes shown below. Register operations offer a choice of register direct and immediate addressing modes. [☞ MN101C00 Series Instruction Manual]

Table 1-4-1 Addressing Mode Overview

Addressing mode		Effective address	Explanation
Register direct	Dn/DWn An/SP PSW	——	Directly specifies the register. Only internal registers can be specified.
Immediate	imm3/imm4 imm8/imm16	——	Direct operation on the operand or mask value appended to the instruction code.
Register indirect	(An)	15 An 0	Specifies the address using an address register.
Register relative indirect	(d8, An)	15 An+d8 0	Specifies the address using an address register with 8-bit displacement.
	(d16, An)	15 An+d16 0	Specifies the address using an address register with 8-bit displacement.
	(d4, PC) (branch instructions only)	17 PC+d4 0 H *1	Specifies the address using the program counter with 4-bit displacement and H bit.
	(d7, PC) (branch instructions only)	17 PC+d7 0 H *1	Specifies the address using the program counter with 7-bit displacement and H bit.
	(d11, PC) (branch instructions only)	17 PC+d11 0 H *1	Specifies the address using the program counter with 11-bit displacement and H bit.
	(d12, PC) (branch instructions only)	17 PC+d12 0 H *1	Specifies the address using the program counter with 12-bit displacement and H bit.
	(d16, PC) (branch instructions only)	17 PC+d16 0 H *1	Specifies the address using the program counter with 16-bit displacement and H bit.
Stack relative indirect	(d4, SP)	15 SP+d4 0	Specifies the address using the stack pointer with 4-bit displacement.
	(d8, SP)	15 SP+d8 0	Specifies the address using the stack pointer with 8-bit displacement.
	(d16, SP)	15 SP+d16 0	Specifies the address using the stack pointer with 16-bit displacement.
Absolute	(abs8)	7 abs8 0	Specifies the address using the operand value appended to the instruction code. Optimum operand length can be used to specify the address.
	(abs12)	11 abs12 0	
	(abs16)	15 abs16 0	
	(abs18) (branch instructions only)	17 abs18 0 H *1	
RAM short	(abs8)	7 abs8 0	Specifies an 8-bit offset from the address x'00000'.
I/O short	(io8)	15 IOTOP+io8 0	Specifies an 8-bit offset from the top address of the special function register area.
Handy	(HA)	——	Reuses the last memory address accessed and is only available with the MOV and MOVW instructions. Combined use with absolute addressing reduces code size.

*1 H: half-byte bit

1-5 List of Instructions

The MN101C00 series provides 37 assembler instructions, shown below.

Table 1-5-1 List of Instructions

Type	Mnemonic	Description
Data transfer	MOV	Transfer 8-bit data between register and memory
	MOVW	Transfer 16-bit data between register and memory
	PUSH	Save register contents onto stack
	POP	Restore register contents from stack
	EXT	Sign-extend data
Arithmetic operations	ADD	Add (8-bit)
	ADDC	Add with carry
	ADDW	Add (16-bit)
	ADDUW	Add with zero extension (16-bit)
	ADDSW	Add with sign extension (16-bit)
	SUB	Subtract (8-bit)
	SUBC	Subtract with borrow
	SUBW	Subtract (16-bit)
	MULU	Unsigned multiplication
	DIVU	Unsigned division
	CMP	Compare (8-bit)
	CMPW	Compare (16-bit)
Logical operations	AND	Logical AND
	OR	Logical OR
	XOR	Exclusive logical OR
	NOT	Not (one's complement)
	ASR	Arithmetic shift right
	LSR	Logical shift right
	ROR	Right rotate
Bit manipulation	BSET	Bit test and set (byte processing)
	BCLR	Bit test and clear (byte processing)
	BTST	Bit test
Program branching	Bcc	Conditional branch (PC relative)
	CBcc	Compare and conditional branch (PC relative)
	TBcc	Bit test and conditional branch (PC relative)
	JMP	Unconditional branch (absolute, register indirect)
	JSR	Branch to subroutine (absolute, register indirect)
	JSRV	Branch to subroutine (vector table indirect)
	NOP	No operation
	RTS	Return from subroutine
	RTI	Return from maskable interrupt
Control operations	REP	Repeat

Chapter 2 - Basic CPU Functions

2

2-1 Clock Generator and Machine Clock

2-1-1 Clock generator

The MN101C00 series has two internal oscillator circuits, one for a high-speed clock (OSCI, OSCO) and one for a low-speed clock (Xi, XO). An oscillator and capacitor are connected externally. A crystal or ceramic oscillator (max. 20 MHz) is connected between OSCI and OSCO, and a ceramic oscillator, typically of 32 kHz is connected between XI and XO.

Either of these clocks can be programmed as the machine clock (the basic CPU clock). The machine clock is divided by two. If the low-speed clock is selected, it is also possible to stop operation of the high-speed oscillation circuit to reduce power dissipation. If the CPU is switched to STOP mode to reduce power dissipation, operation of both oscillation circuits is also stopped.

The connection shown is a self-excited oscillation design.

If an external clock is to be used as the microcomputer system clock, use an external-excited oscillator. In this case, the external clock is input through the OSCI pin, and the OSCO pin is open (or, alternatively, input through the XI pin and the XO pin is open).

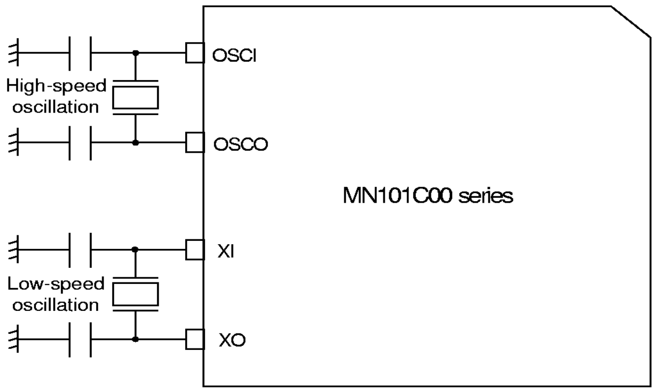


Fig. 2-1-1 Oscillator Circuit Connection (self-excited oscillation)

To minimize distortion, mount the crystal and capacitor as close as possible to the pins.
Also connect the Vss pin to a thick ground line with the shortest possible distance.

2-1-2 Machine clock

Source oscillation is divided by two to form system clock. Machine clock is generated by the system clock. CPU control and execution uses this machine clock for basic timing.

The machine clock consists of a two-phase (T1 and T2) clock.

■ During external memory access (no wait)

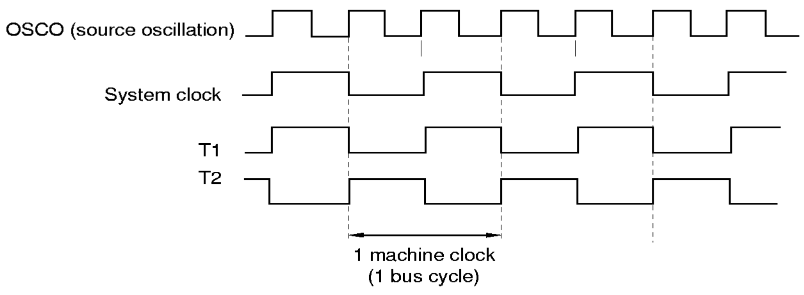


Fig. 2-1-2 Machine Clock with No Wait Cycle

■ During external memory access (0, 1, 2, or 3 waits)

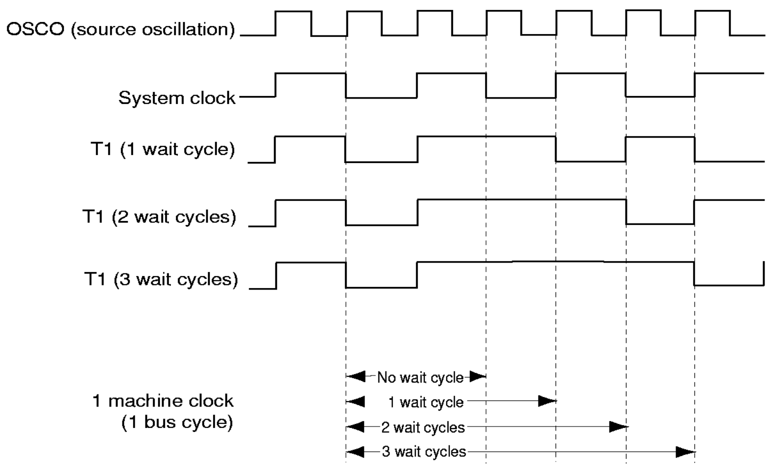


Fig. 2-1-3 Machine Clock with Memory Wait Cycles

At reset start, memory access is set to fixed wait mode with three wait cycles.

2-2 Instruction Execution Controller

2-2-1 Configuration

The instruction execution controller consists of four blocks: memory, instruction queue, instruction registers, and instruction decoder.

Instructions are fetched in 1-byte units, and temporarily stored in the 2-byte instruction queue. Transfer is made in 1-byte or half-byte units from the instruction queue to the instruction register to be decoded by the instruction decoder.

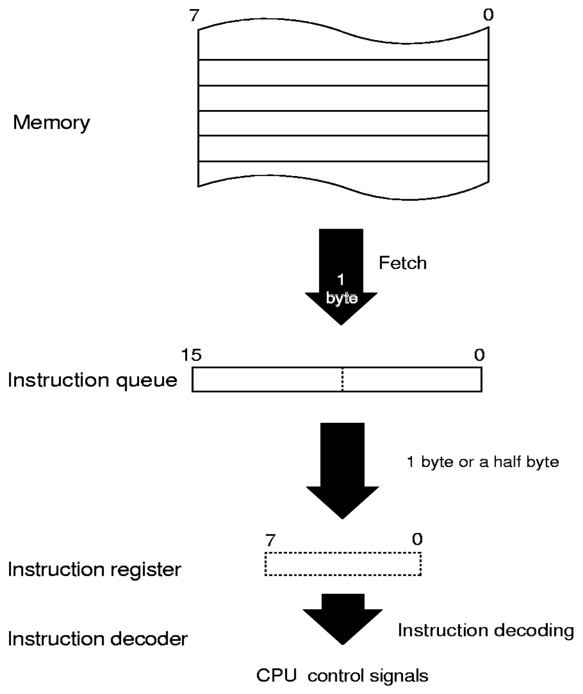


Fig. 2-2-1 Instruction Execution Controller Configuration

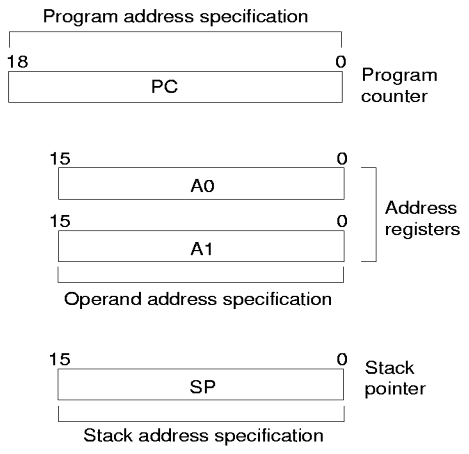
2-3 Internal Registers

2-3-1 Address registers

MN101C00 device registers are divided into CPU core internal registers and special registers for control. Internal registers include address registers, operation registers and processor status word (PSW).

Address registers include the 19-bit program counter (PC), address registers (An), and stack pointer (SP).

The contents of An and SP are undefined at reset start. Please initialize them with the initialization program.



Program address specification
18
0
PC
Program counter
15
0
A0
15
0
A1
Address registers
Operand address specification
15
0
SP
Stack pointer
Stack address specification

Fig. 2-3-1 Address Register Configuration

■ Program Counter (PC)

This register gives the address of the currently executing instruction. It is 19 bits wide to provide access to a 256-kilobyte address space in 4-bit increments. The program's instruction code can use the full linear 256-kilobyte address space, but data must be located within the first 64 kilobytes. A jump subroutine instruction (JSR) pushes these 19 bits into three bytes on the stack for use as the return address.

■ Address Registers (A0, A1)

These registers are used as address pointers specifying data locations in memory. They support only the operations involved in address calculations (i.e., addition, subtraction, and comparison). Transfers between these registers and memory are always in 16-bit units. These transfers do not require that the memory address be aligned on an even-numbered boundary.

■ Stack Pointer (SP)

This register gives the address of the byte at the top of the stack. It is decremented during push operations and incremented during pop operations.

2-3-2 Operation registers

Operation registers include four 8-bit data registers (Dn).

Dn is undefined at reset start. You must initialize these values.

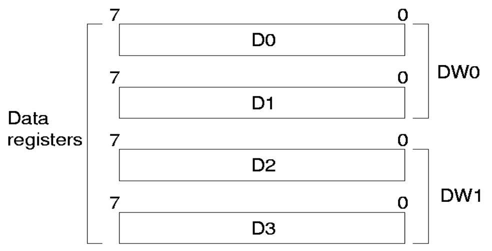


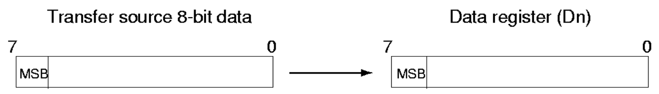
Fig. 2-3-2 Operation Register Configuration

■ Data Registers (D0 to D3)

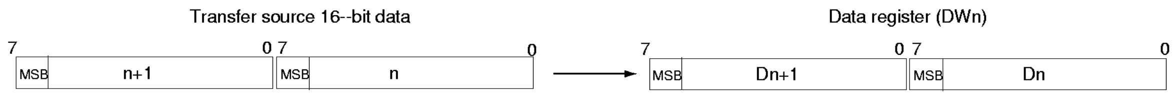
Data registers D0 to D3 are 8-bit general-purpose registers that support all arithmetic, logical and shift operations. All registers can be used for data transfers with memory.

The four data registers may be paired to form the 16-bit data registers DW0 (D0+D1) and DW1 (D2+D3).

• 8-bit transfer



• 16-bit transfer

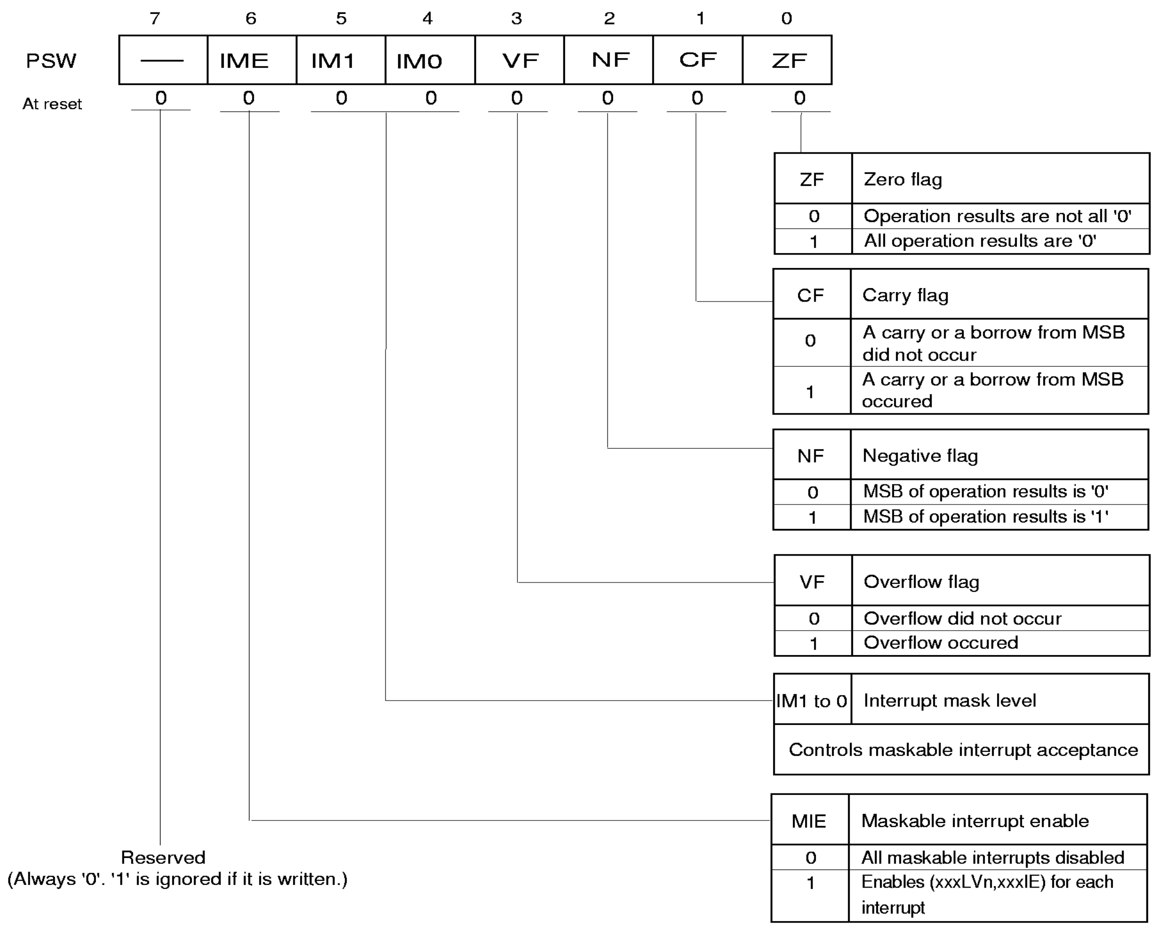


2-3-3 Processor status word

The processor status word (PSW) is an 8-bit register that stores flags for the state of the CPU interrupt control circuit, operation results and other data.

Four flags (VF, NF, CF, and ZF) reflect an operation's result and are reset to '0' upon reset. They can be used with the Bcc command in programs. Two kinds of flags (IM1, IM0, and MIE) are used for controlling interrupts. IM1 and IM0 are reset to '00' and MIE is reset to '0' upon reset.

The PSW is automatically pushed onto the stack when an interrupt occurs and is automatically popped when the interrupt service routine returns.



7 6 5 4 3 2 1 0
PSW
—— IME IM1 IM0 VF NF CF ZF
At reset
0 0 0 0 0 0 0 0
Reserved
(Always '0'. '1' is ignored if it is written.)

ZF	Zero flag
0	Operation results are not all '0'
1	All operation results are '0'

CF	Carry flag
0	A carry or a borrow from MSB did not occur
1	A carry or a borrow from MSB occured

NF	Negative flag
0	MSB of operation results is '0'
1	MSB of operation results is '1'

VF	Overflow flag
0	Overflow did not occur
1	Overflow occured

IM1 to 0	Interrupt mask level
Controls maskable interrupt acceptance	

MIE	Maskable interrupt enable
0	All maskable interrupts disabled
1	Enables (xxxLVn,xxxIE) for each interrupt

Fig. 2-3-3 Processor Status Word Configuration

■ Zero Flag (ZF)

The zero flag (ZF) is set when all the bits in the operation result are '0'. Otherwise, the zero flag is cleared.

■ Carry Flag (CF)

The carry flag (CF) is set when a carry from or a borrow to the MSB occurs. The carry flag is cleared when no carry or borrow occurs.

■ Negative Flag (NF)

The negative flag (NF) is set when the MSB is '1' and reset when the MSB is '0'. The negative flag is used to handle a signed value.

■ Overflow Flag (VF)

The overflow flag (VF) is set when the arithmetic operation results as a signed value. Otherwise, the overflow flag is cleared.
The overflow flag is used to handle a signed value.

■ Interrupt Mask Level (IM1 and IM0)

The interrupt mask level (IM1 and IM0) controls the maskable interrupt acceptance in accordance with the interrupt factor interrupt priority for the interrupt control circuit in the CPU. The two-bit control flag defines levels '0' (00) to '3' (11). Level 0 is the highest mask level. The interrupt will be accepted only when the level set in the interrupt level flag (xxxLVn) of the interrupt control register (ICRn) is higher than the interrupt mask level. When the interrupt is accepted, the level is reset to IM1–IM0, and interrupts whose mask levels are the same or lower are rejected during the accepted interrupt processing.

Table 2-3-1 Interrupt Mask Level and Interrupt Acceptance

Interrupt mask level		Acceptable interrupt levels	Priority
IM1	IM0		
0	0	Non-maskable interrupt (NMI) only	High
0	1	Level 0, NMI	↑
1	0	Levels 0 to 1, NMI	
1	1	Levels 0 to 2, NMI	Low

■ Maskable interrupt enable (MIE)

The maskable interrupt enable flag (MIE) enables/disables acceptance of maskable interrupts by the CPU's internal interrupt acceptance circuit. A '1' enables maskable interrupts; a '0' disables all maskable interrupts regardless of the interrupt mask level (IM1–IM0) setting in the PSW.
This flag is not changed by interrupts.

2-4 Special Function Registers

The MN101C00 series locates the peripheral circuit registers in memory space (X'03F00' to X'03FFF') with memory-mapped I/O. Special function registers control these peripheral circuits and the CPU.

Table 2-4-1 List of Special Function Registers

Address	Symbol	R/W	Name
X3F00'	CPUM	R/W *	CPU mode control registery(☞2-6-2)
X3F01'	MEMCTR	R/W	Memory control registery(☞ 2-10-2)
X'3F02' to X'3FDF'	Depends on specific chip.		
X'3FE0'	Reserved (for debugger)		
X'3FE1'	NMICR	R/W	Non-maskable interrupt register (☞2-5-2)
X'3FE2' to X'3FFE'	xxxICR	R/W	Maskable interrupt register (☞2-5-2)
x'3FFF	Reserved (used by hardware to read the interrupt vector data)		

** Some bits are read only.*

2-5 Interrupt Controller

2-5-1 Outline

The MN101C00 series speeds up interrupt response by adopting the interrupt vector approach of branching to an interrupt service routine. The interrupt vector table can have up to 32 entries. In addition to the reset and non-maskable interrupts, there can be up to 29 peripheral interrupts. Vector 31 is reserved for use by an in-circuit emulator.

Table 2-5-1 Interrupt Controller Overview

Interrupt type	Vector number	Table address	Interrupt level	Interrupt factor	Operation generated
Reset (interrupt) (☞2-7 Reset Function)	0	X'04000'	———	- External $\overline{\text{RST}}$ pin input	Direct input to CPU core
Non-maskable interrupt	1	X'04004'	———	- External $\overline{\text{NMIRQ}}$ pin input - CPU run-away detection interrupt	Input to CPU core from non-maskable interrupt control register (NMICR)
Maskable interupt	2 to 30	X'04008' to X'0407F'	Can be set to levels 0 to 2 by software	- External pin input - Internal peripheral function (timer, serial, etc.)	Input to CPU core of interrupt request level set in interrupt level flag (xxxL Vn) of maskable interrupt control register (xxxICR)

The IVBM flag in the MEMCTR register permits changing the starting address for the interrupt vector table to X'00100' in the internal RAM region.

Accept operation	Starting address	Machine cycles until accepted	PSW status after acceptance
Always accepts	Address specified by vector address	12	All flags are cleared to "0"
Always accepts		12	Interrupt mask level of the PSW is cleared to "0"
Acceptance determined by interrupt control of interrupt mask level (IMn) in the processor status word (PSW) and interrupt control register (xxxICRn)		12	The interrupt level (IL=xxxLV1-xxxLV0) for the interrupt is copied to the interrupt mask (IM=IM1-IM0) in the processor status word (PSW). (☞2-5-6 Multiplex Interrupt Enabled)

Note: The maskable interrupt enable flag (MIE) is not changed by interrupts.

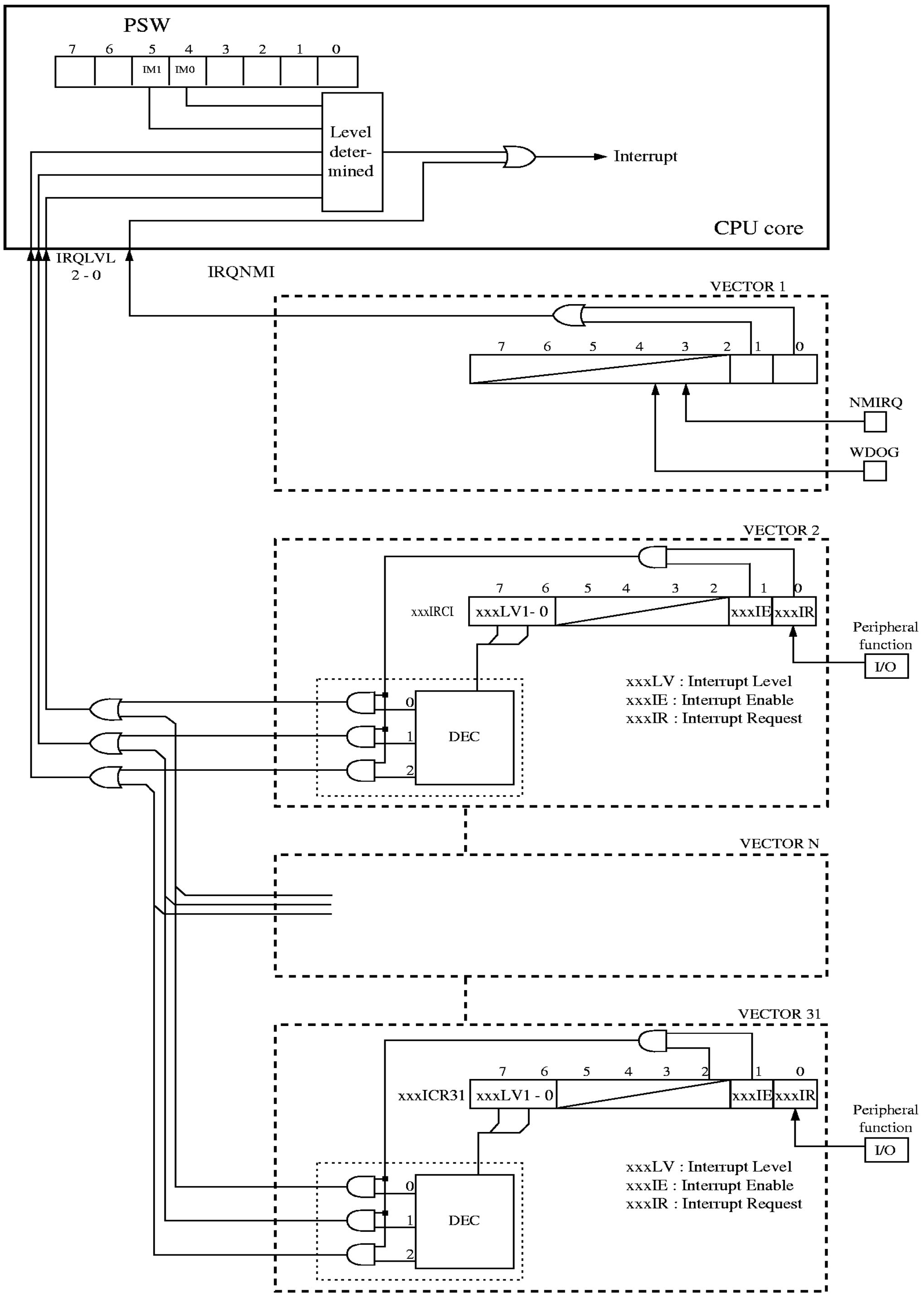
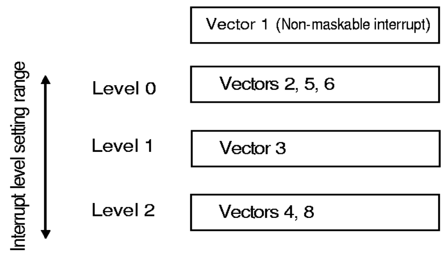


Fig. 2-5-1 Interrupt Controller Configuration

■ Setting Interrupt Groups and Masking Levels

The MN10C00 series permits up to four interrupt factors per interrupt group. While the number of interrupt groups and details of allocation of interrupt factors to each group varies with the product used, the user can program the interrupt level for all groups except vector 0 (reset) and vector 1 (reserved for non-maskable interrupt). There are three hierarchical interrupt levels. If multiple interrupts have the same priority, the one with the lowest vector number takes priority. For example, if a vector 3 set to level 1 and a vector 4 set to level 2 request interrupts simultaneously, vector 3 will be accepted.

With the exception of vectors 0 and 1, each interrupt has a maskable interrupt control register (xxxICR) which controls the interrupts.


Vector 1 (Non-maskable interrupt)
Interrupt level setting range
Level 0
Vectors 2, 5, 6
Level 1
Vector 3
Level 2
Vectors 4, 8

Priority	
1	Vector 1
2	Vector 2
3	Vector 5
4	Vector 6
5	Vector 3
6	Vector 4
7	Vector 8

If multiple interrupts have the same priority, the one with the lowest vector number takes precedence.

Fig. 2-5-2 Interrupt Priority Outline

■ Determination of Interrupt Acceptance

A maskable interrupt is accepted if the maskable interrupt enable flag (MIE) in the processor status word (PSW) is '1', the interrupt enable flag (xxxIE) in the corresponding interrupt control register (xxxICR) is '1', and the interrupt level (xxxLV1–xxxLV0) in the interrupt control register (xxxICR) for the interrupt is lower than the interrupt mask level (IM1–IM0) in the processor status word (PSW). The interrupt mask level (IM0–IM1) is updated to the interrupt level (xxxLV1–xxxLV0), and the interrupt reset flag (xxxICR) is reset to '0'. Reset input and non-maskable interrupts are always accepted, regardless of mask level or the MIE flag setting.

■ Interrupt Processing Sequence

For interrupts other than reset, the interrupt processing sequence consists of interrupt request, interrupt acceptance, and hardware processing. After the interrupt is accepted, the program counter (PC) and processor status word (PSW) are saved onto the stack, the interrupt mask (IM1–IM0) and interrupt request (xxxIR) flags are updated, and execution branches to the address specified by the corresponding interrupt vector.

When the interrupt service routine returns, the hardware restores saved register value.

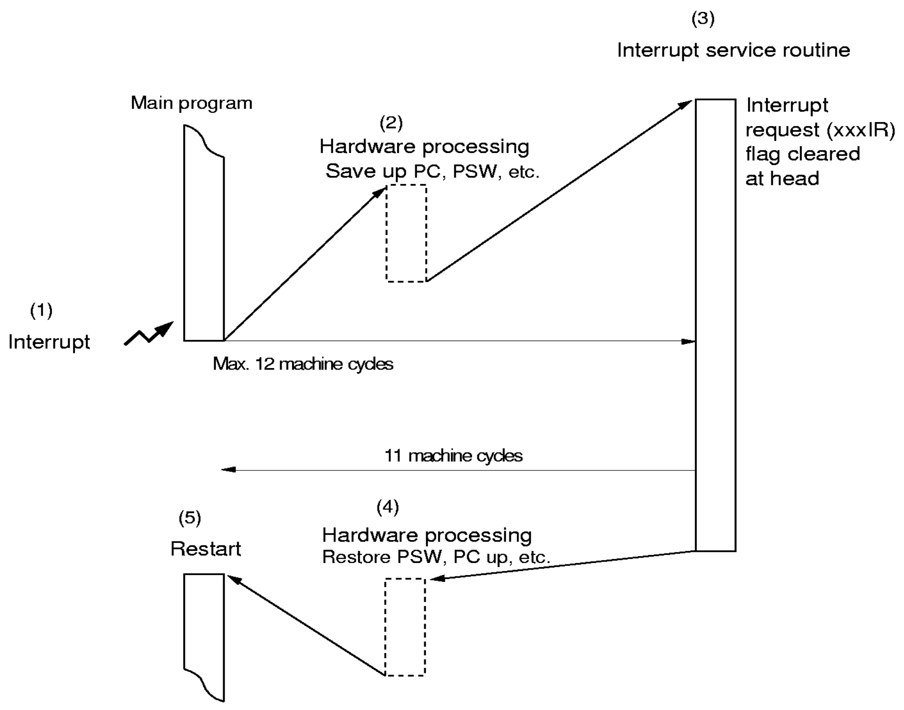


Fig. 2-5-3 Interrupt Processing Sequence (maskable interrupts)

2-5-2 Interrupt control registers

The interrupt control registers include the maskable interrupt control registers (xxxICR) and the non-maskable interrupt control register (NMICR).

xxxICR can be labelled (for instance as TMICR for a time function) to clarify its relation to peripheral functions.

■ Maskable Interrupt Control Register (xxxICR)

A maskable interrupt control register (xxxICR) controls the interrupts for each maskable interrupt group (but not group 0). The register consists of the interrupt level field (xxxLV1–xxxLV0), interrupt enable flag (xxxIE), and interrupt request flag (xxxIIR).

An interrupt level (xxxLV0–xxxLV1) of '11' (level 3) disables interrupts for the group regardless of interrupt enable and interrupt request flags.

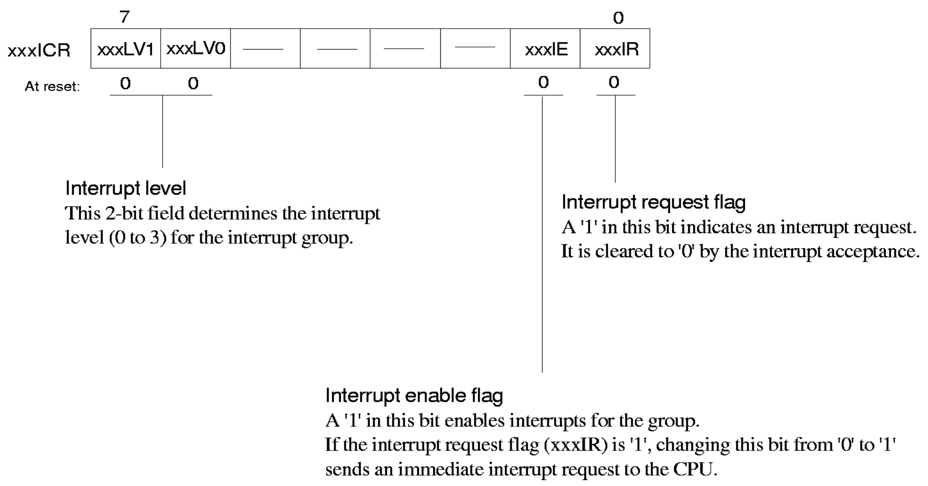

7
0
xxxICR
xxxLV1
xxxLV0
xxxIE
xxxIR
At reset:
0
0
0
0
Interrupt level
This 2-bit field determines the interrupt level (0 to 3) for the interrupt group.
Interrupt request flag
A '1' in this bit indicates an interrupt request. It is cleared to '0' by the interrupt acceptance.
Interrupt enable flag
A '1' in this bit enables interrupts for the group.
If the interrupt request flag (xxxIR) is '1', changing this bit from '0' to '1' sends an immediate interrupt request to the CPU.

Fig. 2-5-4 Maskable Interrupt Control Register (xxxICR)

For interrupt factors which are critical or have short permissible processing times, set the interrupt level (xxxLV1–xxxLV0) to a high level (small value).

Always reset the maskable interrupt enable flag (MIE) to "0" before manipulating the contents of a maskable interrupt control register.

The initialization routine or other software manipulating the interrupt request flag (xxxIR) must first set the IR write enable flag (IRWE) in the memory control register (MEMCTR) to "1" and reset the flag to "0" after the operation.

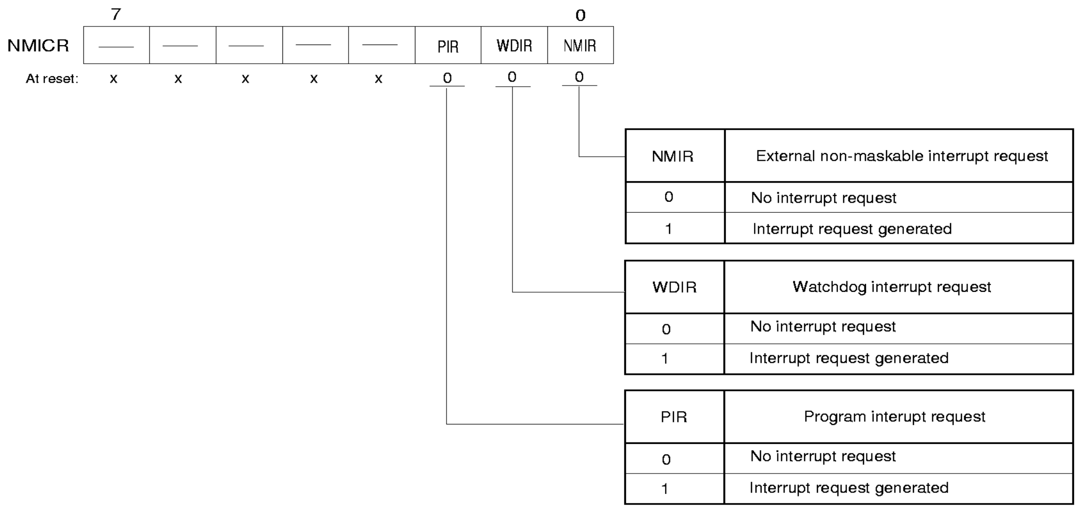
Table 2-5-2 Interrupt Mask Levels and Interrupt Levels

Interrupt mask level (PSW)	Acceptable interrupt level	Priority
0	Non-maskable interrupt (NMI) only	High
1	Level 0, NMI	↑
2	Levels 0 or 1, NMI	
3	Levels 0 to 2, NMI	Low

The non-maskable interrupt specifications vary with the specific product. See the specifications for the particular product.

■ Non-Maskable Interrupt Control Register (NMICR address: X'3FE1')

The non-maskable interrupt control register (NMICR) is assigned to vector 1. Bits in this register indicate the interrupt factors for non-maskable interrupt requests. Such interrupt requests are accepted regardless of the interrupt mask level (IM1–IM0) setting in the PSW. The hardware then branches to the address stored at location X'4004' in the interrupt vector table.



NMIR	External non-maskable interrupt request
0	No interrupt request
1	Interrupt request generated

WDIR	Watchdog interrupt request
0	No interrupt request
1	Interrupt request generated

PIR	Program interupt request
0	No interrupt request
1	Interrupt request generated

Fig. 2-5-5 Non-Maskable Interrupt Control Register (NMICR)

The external non-maskable interrupt request (NMIR) and watchhdog timer overflow interrupt request (WDIR) flags are preserved after the interrupts are accepted. Use the non-maskable interrupt processing program to clear these flags.

■ External Non-Maskable Interrupt Request Flag (NMIR)

The external non-maskable interrupt request flag is set to '1' when a negative edge (minimum pulsewidth 4 cycles) is detected by the external NMIRQ pin.

■ Watchdog Timer Overflow Interrupt Request Flag (WDIR)

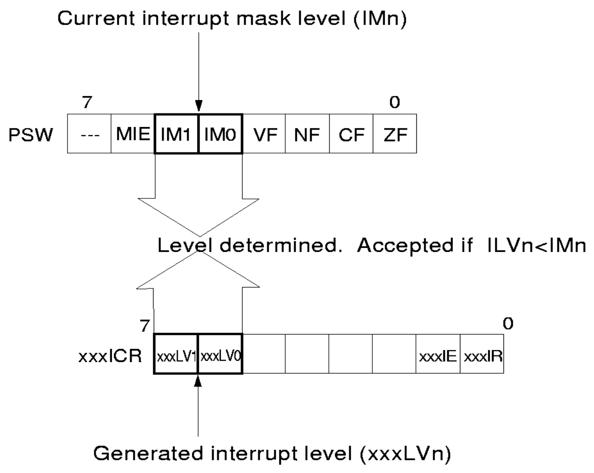
The watchdog timer overflow interrupt request flag is set to '1' when the watchdog timer overflows.

■ Program Interrupt Request Flag (PIR)

The program interrupt request flag is set to '1' by software.

2-5-3 Interrupt level

The interrupt level can be set for each interrupt group. Maskable interrupt requests may be accepted or not depending on the states of the maskable interrupt enable flag (MIE) and the interrupt mask level (IM1, IM0) in the processor status word (PSW) and the interrupt enable flag (xxxIE) in the maskable interrupt control register (xxxICR).


Current interrupt mask level (IMn)
7
0
PSW

MIE
IM1
IM0
VF
NF
CF
ZF
Level determined. Accepted if ILVn<IMn
7
0
xxxICR
xxxLV1
xxxLV0
xxxIE
xxxIR
Generated interrupt level (xxxLVn)

Fig. 2-5-6 Interrupt Acceptance Determination

The sequence from interrupt request generation to acceptance is described below.

(1) When an interrupt request is generated, the interrupt request flag (xxxICR) in the corresponding maskable interrupt control register (xxxICR) is set to '1'.

(2) If the interrupt enable flag (xxxIE) in the same register is '1', the interrupt request is output to the CPU.

(3) The level set in the interrupt level field (xxxLV1–xxxLV0) for the group is output to the CPU.

(4) If the output interrupt request signal has a level lower than the processor status word (PSW) interrupt mask level (IM1, 0), and the PSW maskable interrupt enable flag (MIE) is '1' (enabled), the interrupt is accepted.

(5) When the hardware accepts the interrupt, it resets the corresponding interrupt request flag (xxxIR) to "0."

Acceptance of an interrupt does not reset the corresponding interrupt enable flag (xxxIE) to "0."

If interrupts of the same level are generated at the same time, priority is given to the interrupt with the lower vector number.

MIE='0' and interrupts are disabled when:

- MIE in the PSW is reset to '0' by a program
- Reset is detected

MIE='1' and interrupts are enabled when:

- MIE in the PSW is set to '1' by a program

The value of the interrupt mask level changes when:

- The program writes a new value to IM1, 0 in the PSW
- Reset is detected. In this case, IM1=IM0='00'
- A maskable interrupt is accepted, and its interrupt level becomes the interrupt mask level
- The RTI instruction is executed at the end of an interrupt service routine, and the mask level from before interrupt acceptance is restored

The MN101C00 series does not reset the maskable interrupt enable flag (MIE) flag in the processor status word (PSW) to "0" when accepting interrupts. Note that this behavior is different from that of the MN10200, MN1860, MN1870, and MN1880 series.

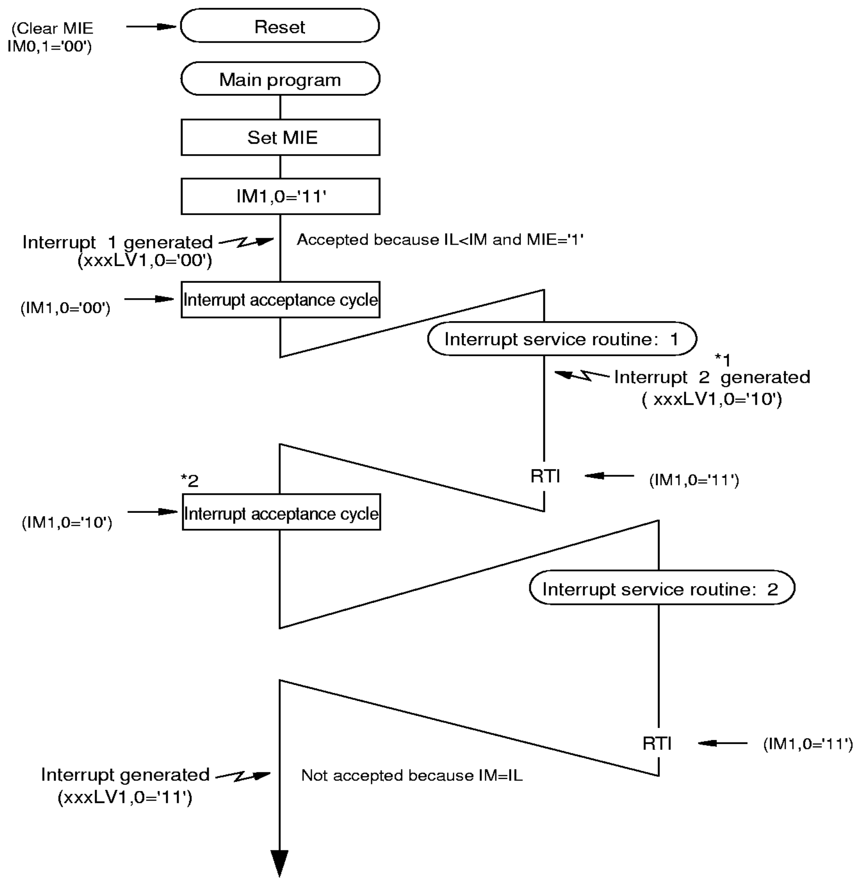
If maskable and non-maskable interrupts are generated simultaneously, the non-maskable interrupt has priority.

Figure 2-5-7 shows the processing flow when a second interrupt with a lower priority level (xxxLV1–xxxLV0='10') arrives during the processing of one with a higher priority level (xxxLV1–xxxLV0='00').

Parentheses ( ) indicate hardware processing.

(1)
If, during the processing of the first interrupt, an interrupt request with an interrupt level (IL) numerically lower than the interrupt mask (IM) arrives, it is accepted as a nested interrupt. If ILÅÜIM, however, the interrupt is not accepted.

(2)
The second interrupt, postponed because its interrupt level (IL) was numerically greater than the interrupt mask (IM) for the first interrupt service routine, is accepted when the first interrupt handler returns.



(Clear MIE IM0,1='00')
Reset
Main program
Set MIE
IM1,0='11'
Interrupt 1 generated (xxxLV1,0='00')
Accepted because IL<IM and MIE='1'
(IM1,0='00')
Interrupt acceptance cycle
Interrupt service routine: 1
*1
Interrupt 2 generated (xxxLV1,0='10')
RTI
(IM1,0='11')
*2
(IM1,0='10')
Interrupt acceptance cycle
Interrupt service routine: 2
RTI
(IM1,0='11')
Interrupt generated (xxxLV1,0='11')
Not accepted because IM=IL

Fig. 2-5-7 Processing Sequence for Maskable Interrupts

2-5-4 Interrupt acceptance operation

When accepting an interrupt, the MN101C00 hardware saves the handy address register, the return address from the program counter, and the processor status word (PSW) to the stack and branches to the interrupt handler using the starting address in the vector table.

The following is the hardware processing sequence invoked by interrupt acceptance.

1. The stack pointer (SP) is updated.
   (SP-6 → SP)

The handy address register is an internal register used by the handy addressing function. The hardware saves its contents to the stack to prevent the interrupt from interfering with operation of the function.

2. The contents of the handy address register (HA) are saved to the stack.
   Upper half of HA → (SP+5)
   Lower half of HA → (SP+4)
3. The contents of the program counter (PC)—i.e., the return address—are saved to the stack.
   PC bits 18, 17, and 0 → (SP+3)
   PC bits 16-9 → (SP+2)
   PC bits 8-1 → (SP+1)
4. The contents of the PSW are saved to the stack.
   PSW → (SP)
5. The interrupt level (xxxLVn) for the interrupt is copied to the interrupt mask (IM) in the PSW.
   Interrupt level (xxxLVn) → IM
6. The hardware branches to the address in the vector table.

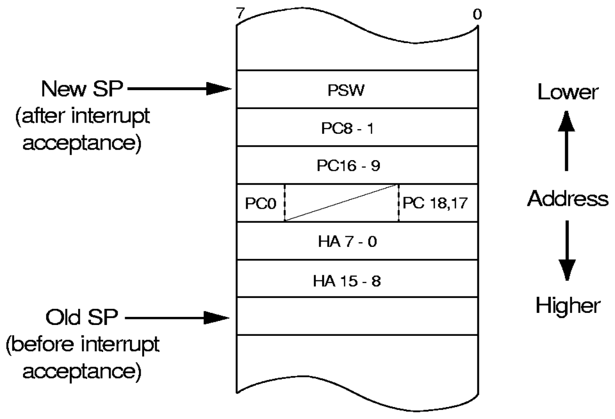


Fig. 2-5-8 Stack Operation during Interrupt Acceptance

2-5-5 Interrupt return operation

An interrupt handler ends by restoring, using the POP instruction and other means, the contents of any registers used during processing and then executing the return from interrupt (RTI) instruction to return to the point at which execution was interrupted.

The following is the processing sequence invoked by the RTI instruction.

1. The contents of the PSW are restored from the stack. (SP)
2. The contents of the program counter (PC)—i.e., the return address—are restored from the stack. (SP+1 to SP+3)
3. The contents of the handy address register (HA) are restored from the stack. (SP+4, SP+5)
4. The stack pointer is updated. (SP+6 →SP)
5. Execution branches to the address in the program counter.

2-5-6 Multiplex interrupt enable

When an MN101C00 series device accepts an interrupt, it automatically disables acceptance of subsequent interrupts with the same or lower priority level.

It is possible to forcibly rewrite IM to accept an interrupt with a priority lower than the interrupt being processed, but be careful of stack overflow.

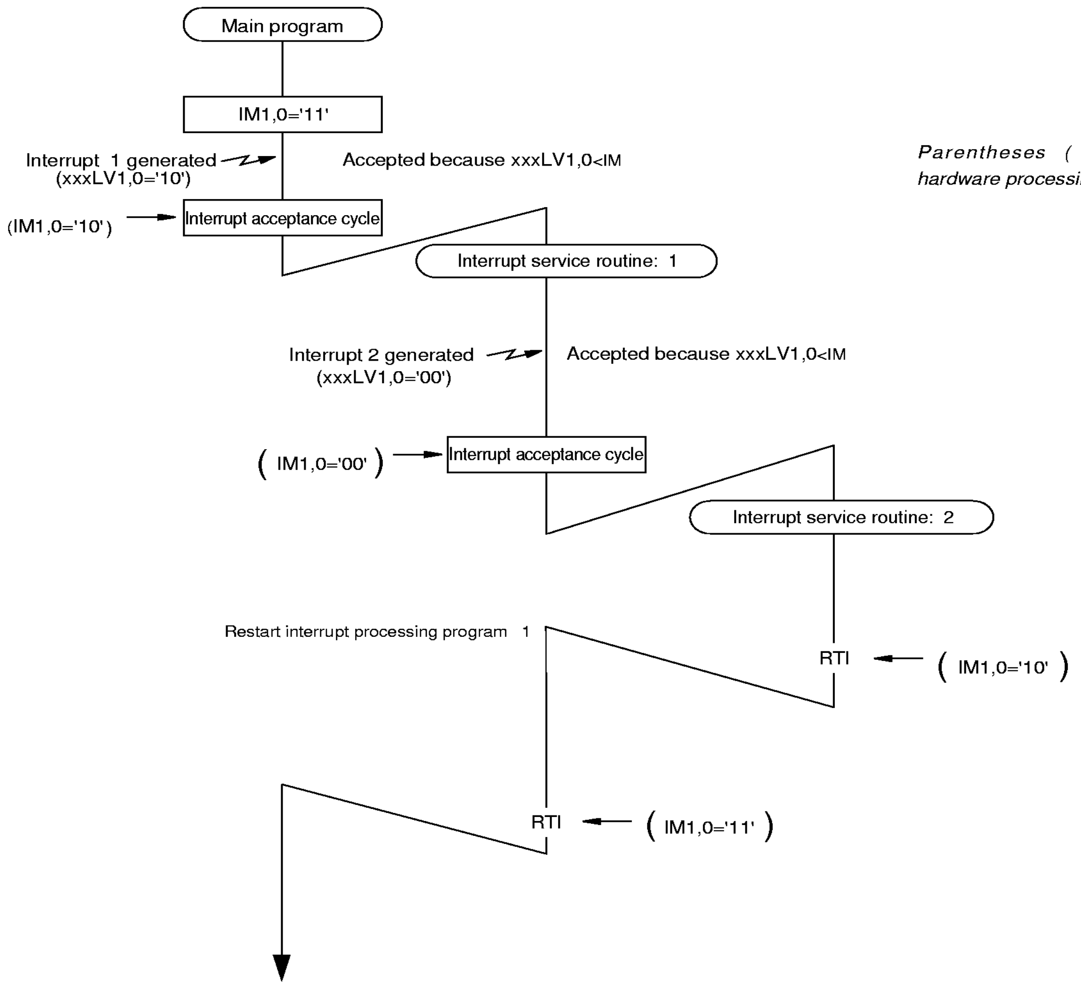
When the hardware accepts an interrupt, it copies the interrupt level (xxxLVn) for the interrupt to the interrupt mask (IM) in the PSW. As a result, subsequent interrupts with the same or lower priority levels are automatically masked. Only interrupts with higher priority levels are accepted. The net result is that interrupts are normally processed in decreasing order of priority. It is, however, possible to alter this arrangement.

1. To disable interrupt nesting
   - Reset the MIE bit in the PSW to "0."
   - Raise the priority level of the interrupt mask (IM) in the PSW.
2. To enable interrupts with lower priority than the currently accepted interrupt
   - Lower the priority level of the interrupt mask (IM) in the PSW.

Multiplex interrupts are only enabled for interrupts with levels higher than the PSW interrupt mask level (IM).

Do not operate the maskable interrupt control register (xxxICR) when multiple interrupts are enabled. If operation is necessary, first clear the PSW MIE flag.

Figure 2-5-10 shows the processing flow for multiple interrupts (interrupt 1: xxxLV1–xxxLV0='10', and interrupt 2: xxxLV1–xxxLV0='00').


Main program
IM1,0='11'
Interrupt 1 generated (xxxLV1,0='10')
Accepted because xxxLV1,0<IM
(IM1,0='10')
Interrupt acceptance cycle
Interrupt service routine: 1
Interrupt 2 generated (xxxLV1,0='00')
Accepted because xxxLV1,0<IM
(IM1,0='00')
Interrupt acceptance cycle
Interrupt service routine: 2
Restart interrupt processing program 1
RTI
(IM1,0='10')
RTI
(IM1,0='11')

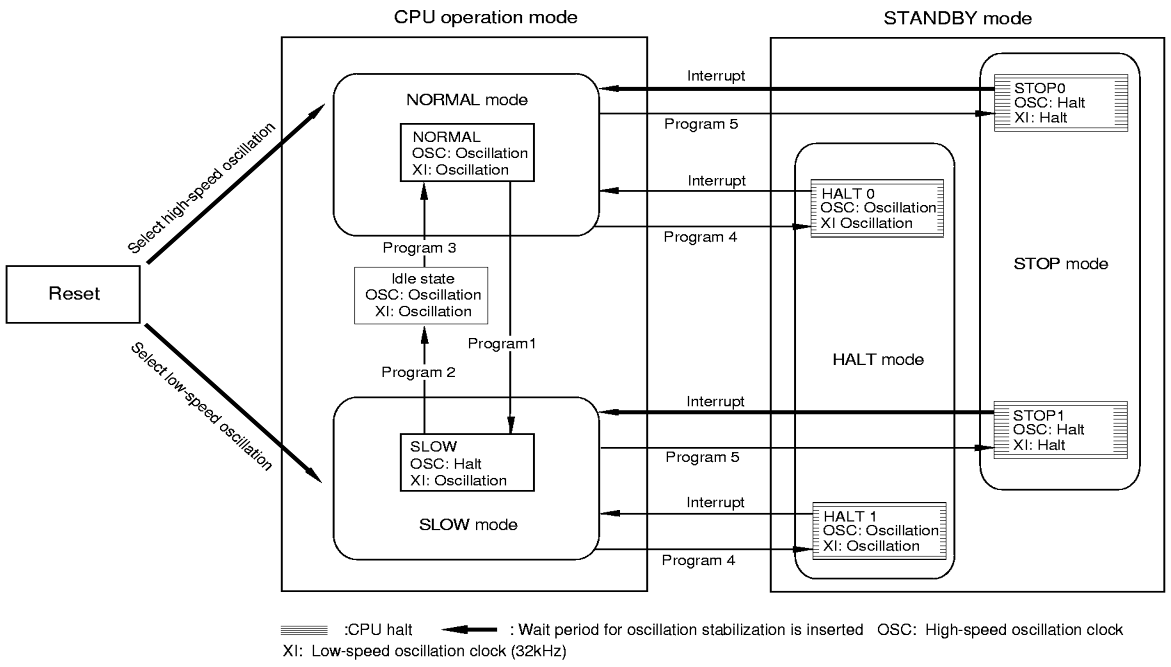
Parentheses ( ) indicate hardware processing.

Fig. 2-5-10 Processing Sequence with Multiple Interrupts Enabled

2-6 Standby Function

2-6-1 Outline

The MN101C00 series has two sets of system clock oscillator pins (high-speed and low-speed oscillation), two CPU operation modes (NORMAL and SLOW), and two standby modes (HALT and STOP). Effective use of these modes can reduce power dissipation.



Sample programs for programs 1 to 5 are shown later in this chapter.

- *Some products have only one system clock oscillation circuit, so do not support the HALT1, SLOW, and STOP1 modes.*
- *Some products have a pin that permits selection of the initial mode after a reset. Others do not offer this choice.*

Fig. 2-6-1 Transition Between Operation Modes

■ HALT Modes (HALT0, HALT1)

- The CPU stops operating, but the oscillators remain operational. An interrupt produces an immediate return to the CPU's operational state.
- In the HALT0 mode, both the high- and low-speed oscillators remain operational. An interrupt produces a return to the NORMAL mode.

■ STOP Modes (STOP0, STOP1)

- The CPU and the oscillators stop operating. An interrupt restarts the oscillators. The CPU restarts after allowing time for the oscillators to stabilize.
- From the STOP0 mode, an interrupt returns the CPU to the NORMAL mode.
- From the STOP1 mode, an interrupt returns the CPU to the SLOW mode.

■ SLOW Mode

- This mode executes the software using the low-speed clock. Since the high-speed oscillator is turned off, the device consumes less power while executing the software.

■ IDLE Mode

- This mode allows time for the high-speed oscillator to stabilize when the software is changing from SLOW to NORMAL mode.

To reduce power dissipation in STOP and HALT modes, it is necessary to check both the output current from pins and port level of input pins. For output pins, the output level should match the external level or direction control should be changed to input mode. For input pins, the external level should be fixed.

The MN101C00 series has two system clock oscillation circuits. OSCI is for high-speed operation (NORMAL mode) and XI is for low-speed operation (SLOW mode). Transition between NORMAL and SLOW modes or to standby mode is controlled by the CPU mode control register (CPUM). Reset and interrupts are the return factors from standby mode. A wait period is inserted for oscillation stabilization at reset and when returning from STOP mode, but not when returning from HALT mode. High/low-speed oscillation mode is automatically returned to the same state as existed before entering standby mode.

CPUM : X'3F00' R/W
[☞2-7 Reset Function]

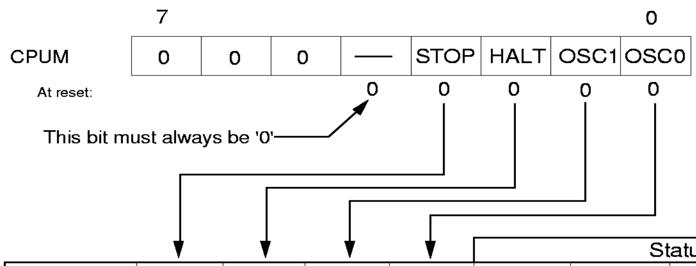
To stabilize the synchronization at the moment of switching clock speed between high-speed and low-speed, high-speed oscillation frequency (fosc) should be set to 2.5 times or higher frequency than the low-speed oscillation frequency (fx).

2-6-2 CPU mode control register

CPUM: X'3F00' R/ W

Transition to other modes is controlled by operating the related flags in the CPU mode control register (CPUM).

Use MOV instruction to write data in the CPUM register.



					Status			
Operation mode	STOP	HALT	OSC1	OSC0	OSCI /OSCO	XI/XO	System clock	CPU
NORMAL	0	0	0	0	Oscillation	Oscillation	OSCI	Operating
IDLE	0	0	0	1	Oscillation	Oscillation	XI	Operating
SLOW	0	0	1	1	Halt	Oscillation	XI	Operating
HALT0	0	1	0	0	Oscillation	Oscillation	OSCI	Halt
HALT1	0	1	1	1	Halt	Oscillation	XI	Halt
STOP0	1	0	0	0	Halt	Halt	Halt	Halt
STOP1	1	0	1	1	Halt	Halt	Halt	Halt

Fig. 2-6-2 Operation Mode Control and Clock Oscillation On/Off

The procedure for transition from NORMAL to HALT or STOP mode is given below.

(1) If the return factor is a maskable interrupt, set the MIE flag in the PSW to "1" and set the interrupt mask (IM) to a level permitting acceptance of the interrupt.

(2) Clear the interrupt request flag (xxxIR) in the maskable interrupt control register (xxxICR) , set the interrupt enable flag (xxxIE) for the return factor, and set the IE flag in the PSW.

(3) Set CPUM to HALT or STOP mode.

2-6-3 Transition between SLOW and NORMAL

The MN101C00 series has two CPU operating modes, NORMAL and SLOW. Transition from SLOW to NORMAL requires passing through an idle state.

A sample program for transition from NORMAL to SLOW mode is given below.

Transition from NORMAL to SLOW mode, when the low-speed clock has fully

```
Program 1     ex.1)
                MOV    x'3', D0          ; set SLOW mode
                MOV    D0, (CPUM)

              ex.2)
                BSET   (CPUM)0
                BSET   (CPUM)1
```

stabilized, can be done by writing to the CPU mode control register. In this case, transition through the idle state is not needed.

For transition from the SLOW to NORMAL mode, the program must maintain the idle state until high-speed clock oscillation is fully stable.

In the idle state, the CPU operates on the low-speed clock.

Even though the same length of wait time is needed to stabilize oscillation at reset, timing count must be controlled by the program in this case.
We recommend selecting the oscillation stabilization interval after consulting with the oscillator manufacturer.

Sample program for transition from SLOW to NORMAL mode is given below.

```
Program 2    ex.1)
                  MOV     x'01', D0          ; set IDLE mode
                  MOV     D0, (CPUM)

             ex.2)
                  BCLR    (CPUM)1
```

```
Program 3    ex.1)
                  MOV     x'0B', D0          ; A loop to keep low-speed clock
LOOP              ADD     -1, D0             ; (32 kHz) operation for apprx. 6.7ms when
                  BNE     LOOP               ; changed to high-speed clock (20 MHz).
                  SUB     D0, D0             ; set NORMAL mode
                  MOV     D0, (CPUM)

             ex.2)
                  MOV     x'0B', D0
LOOP              SUB     1, D0
                  BNE     LOOP
                  BCLR    (CPUM)0
```

2-6-4 Transition to STANDBY mode

The program controls transition from CPU operation mode to STANDBY mode, and the interrupt initiates the return process.

Before the transition to STANDBY mode, the following settings are necessary:

(1) Clear interrupt enable flag (MIE) in the processor status word (PSW) and interrupt enable flag (xxxIE) in the maskable interrupt control register (xxxICR) to disable all interrupts temporarily.

(2) Determine the interrupt for the return factor to transfer control from STANDBY mode to CPU operation mode, and set the appropriate xxxIE only. Set MIE flag in PSW as well.

If the interrupt is enabled and interrupt priority level of the interrupt to be used is not equal to or higher than the mask level in PSW before transition to HALT or STOP mode, it is impossible to return to CPU operation mode by maskable interrupt.

Processing inside parentheses () is handled by hardware.

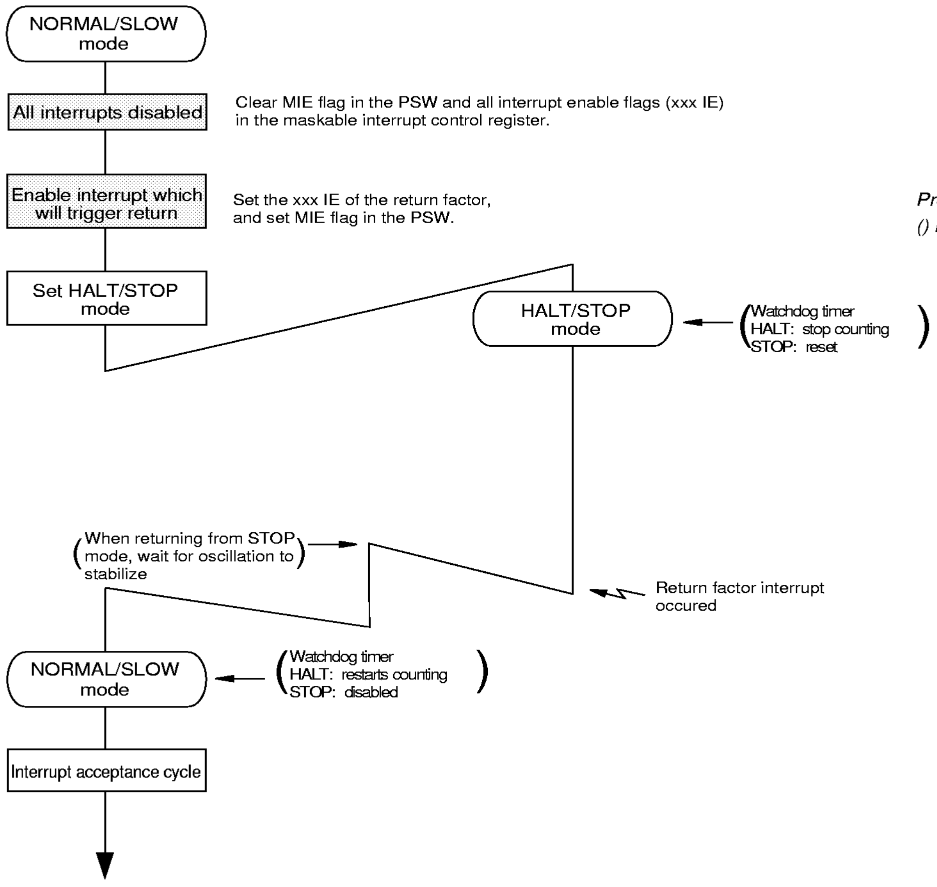

NORMAL/SLOW mode
All interrupts disabled
Clear MIE flag in the PSW and all interrupt enable flags (xxx IE) in the maskable interrupt control register.
Enable interrupt which will trigger return
Set the xxx IE of the return factor, and set MIE flag in the PSW.
Set HALT/STOP mode
HALT/STOP mode
(Watchdog timer HALT: stop counting STOP: reset)
Return factor interrupt occured
(When returning from STOP mode, wait for oscillation to stabilize)
NORMAL/SLOW mode
(Watchdog timer HALT: restarts counting STOP: disabled)
Interrupt acceptance cycle

Fig. 2-6-3 Transition to/from STANDBY Mode

■ Transition to HALT mode

The system transfers from NORMAL mode to HALT0 mode, and from SLOW mode to HALT1 mode. In both cases, oscillation is maintained and only the CPU is halted. Return from HALT mode is initiated by an interrupt or a reset. A reset is just as in normal reset operation, and the interrupt restores the system to the state it was in prior to HALT. If the watchdog timer is enabled when the system switches to HALT mode, the watchdog timer count is halted and restarts continuously when the system returns to CPU operation.

CPUM: X'3F00' R/ W

```
Program 4   ex.1)
               MOV    x'4',  D0         ; set HALT mode
               MOV    D0,  (CPUM)
               NOP                      ; After written in CPUM, some NOP
               NOP                      ; instructions (three or less) arre
               NOP                      ; executed.
            ex.2)
               BSET   (CPUM)2
               NOP
               NOP
               NOP
```

■ Transition to STOP mode

The system transfers from NORMAL mode to STOP0 mode, and from SLOW mode to STOP1 mode. In both cases, oscillation and the CPU are both halted. Return from STOP mode is initiated by an interrupt or a reset. At transition to STOP mode, the watchdog timer is reset and acts as a counter for the oscillation stabilization period, and after that it is disabled.

If the wait for oscillation stabilization ends and the system switches to CPU operation mode, the watchdog timer is automatically disabled. If watchdog timer monitoring is required, enable it specifically.

```
Program 5   ex.1)
               MOV    x'8',  D0         ; set STOP mode
               MOV    D0,  (CPUM)
               NOP                      ; After written in CPUM, some NOP
               NOP                      ; instructions (three or less) are
               NOP                      ; executed.

            ex.2)
               BSET   (CPUM)3
               NOP
               NOP
               NOP
```

2-7 Reset Function

Setting the $\overline{\text{RST}}$ pin to low level will reset the CPU internally and initialize the registers.

(1) The system shifts to the reset state* when the RST pin goes low.

(2) When the $\overline{\text{RST}}$ pin switches from low to high, the internal 15-bit binary counter begins to count the source oscillation clock. It counts for a period called the oscillation stabilization wait time and releases the internal reset when oscillations have stabilized.

The $\overline{\text{RST}}$ pin must be held low for at least four cycles of the source oscillation clock.

In some chips, the reset pin is shared with a port. In these cases, setting the port to 0 will shift to reset state.

When a low-voltage detect circuit is connected to the RST pin, be sure to use a circuit capable of generating a sufficient low pulse in the event of instantaneous drop in the level. Even if the pulsewidth is less than four source oscillation cycles, it is possible that a reset will be triggered, so be careful of noise.

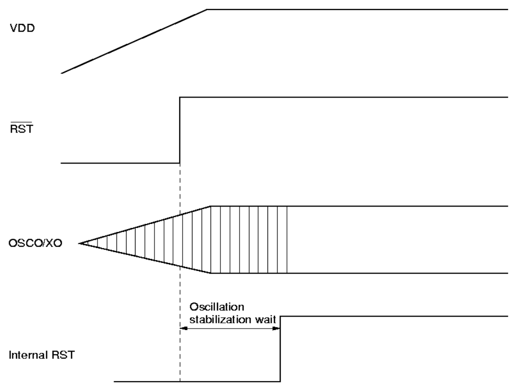


Fig. 2-7-1 Reset Release Sequence

The wait time for oscillation stabilization is determined as given below.

High-speed oscillation: Oscillation stabilization time for oscillation frequency fosci.

$tosciw=2^{15} X (1/fosci)$

For example, when fosci=20MHz, tosciw=1.6384ms

Low-speed oscillation: Oscillation stabilization time for oscillation frequency fxi.

$txiw=2^{15} X (1/fxi)$

For example, when fxi=32kHz, txiw=1.024s

(3) Hardware implemented reset processing initializes the following internal and special registers.

Table 2-7-1 Register Initialization at Reset

Register name		Register address	Initial value
Processor status word	PSW	--	X'00'
Program counter	PC	--	Address stored in X'04000'
Address register	An	--	Not fixed
Data register	Dn	--	Not fixed
CPU mode control register	CPUM	X'03F00'	X'00'
Memory control register	MEMCTR	X'03F01'	X'CB'
Interrupt control register	xxxICR(NMICR)	X'03FE2 to '03FEF'	X'00'

(4) When the oscillation stabilization wait time is over, the internal reset is released and program execution is started from the address that is written in the vector table at address X'04000'. The initialization program should be located at the beginning of the program.

(5) Immediately after reset processing, the CPU operates in three-wait cycle WAIT mode for the external memory and in fixed three-wait cycle WAIT mode for the special register space in accordance with the initial setting of the MEMCTR register.

[2-9 Bus Controller]

2-8 Memory

2-8-1 Setting memory mode

For memory, ROM is the read only area and RAM is the memory area which contains readable/writable data.In addition to these, peripheral resources such as memory-mapped special registers are allocated. The MN101C00 series supports three memory modes in its memory model.

The MN101C00 series allocates ROM, RAM, I/O, and peripheral circuit control registers to the same address space.

Mode selection range and set method vary with product.

Modes are specified with the mode set pin (MMOD) and EXMEM flag in the MEMCTR register.

Table 2-8-1 Setting The Memory Mode

MMOD pin	EXMEM flag in MEMCTR register	Memory mode
L	0	Single chip mode
L	1	Memory expansion mode
H	Don't care	Processor mode

2-8-2 Single-chip mode

In single-chip mode, the system consists of only internal memory. This is the optimized memory model and allows construction of systems with the highest performance.

The single-chip mode uses only internal ROM and internal RAM. The MN101C00 series devices offer up to 12 kilobytes of RAM and up to 240 kilobytes of ROM. For the exact amounts, check the specifications of the individual product.

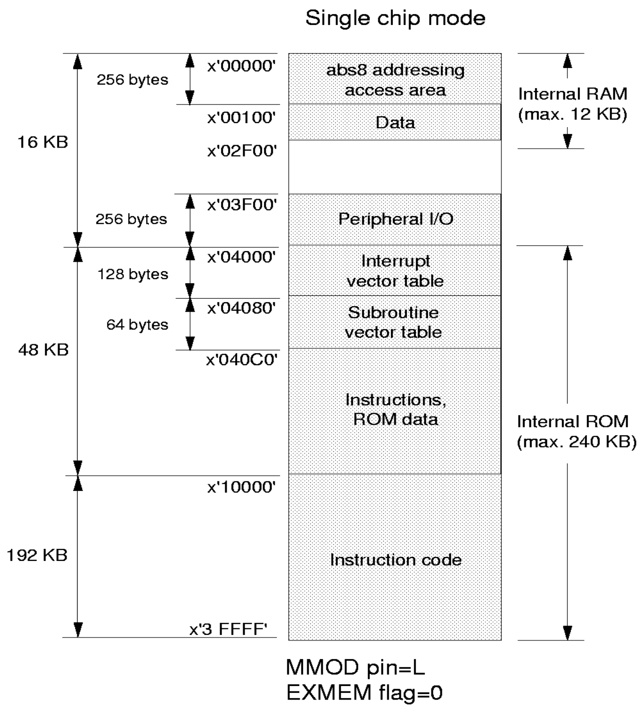


Fig. 2-8-1 Single-chip Mode Configuration

2-8-3 Memory expansion modes

The MN101C00 series can connect external ROM, RAM and I/O ports for operation in memory expansion modes. This is the mode to expand to external memory while using internal ROM and RAM.

In memory expansion modes, the start address at reset is allocated to internal ROM, so at least 8 Kbytes of internal ROM is needed.

■ Memory expansion mode

This mode permits the use of external expansion ROM and RAM to augment the internal ROM and RAM. To use this mode, set the EXMEM flag in the memory control register (MEMCTR) to "1." The following regions are available for expansion.

RAM: X'02F00'–X'03EFF' (4 kilobytes)
ROM: X'20000'–X'3FFFF' (128 kilobytes)

Note that, in this mode, the internal ROM region ends at X'1FFFF'. Any internal ROM above the address X'2000' is ignored, with only the external ROM accessed.

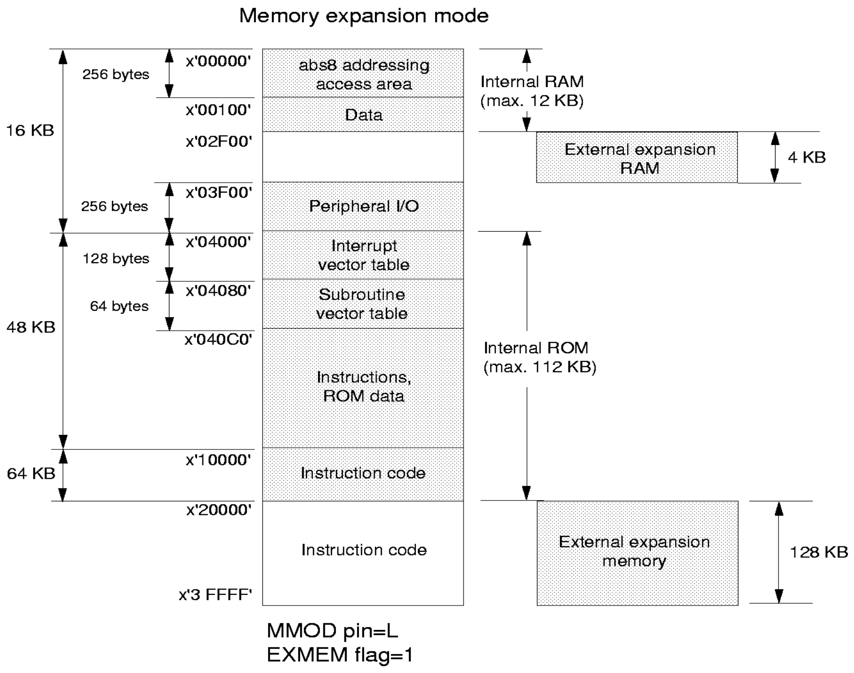


Fig. 2-8-2 Memory Expansion Mode

2-8-4 Processor mode

This mode accesses only the external expansion ROM, ignoring any internal ROM present.

This mode uses internal RAM and external ROM. To use this mode, drive the MMOD pin at "H" level. The following region is available for expansion.

RAM: X'02F00'–X'03EFF' (4 kilobytes)

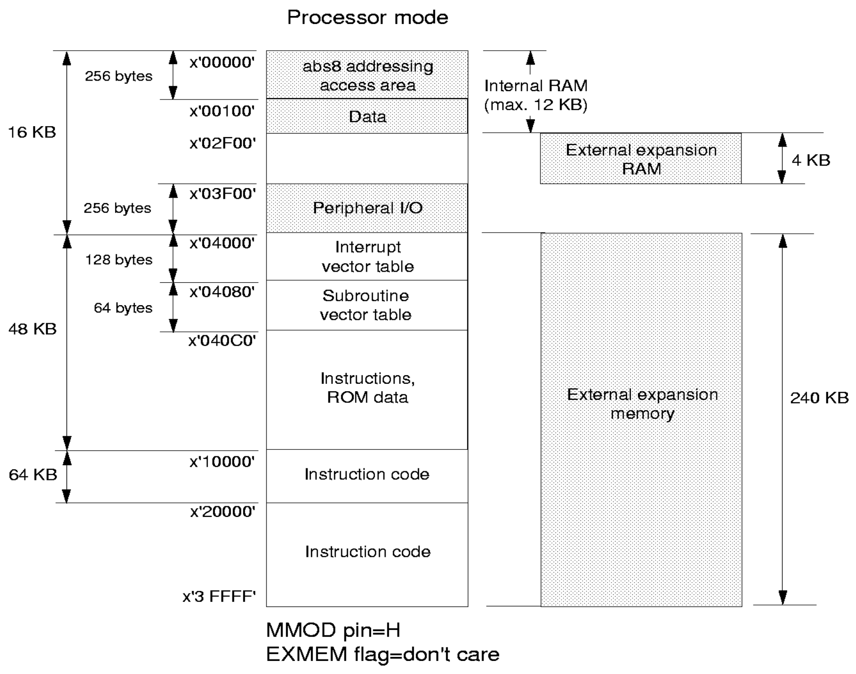


Fig. 2-8-3 Processor Mode

2-9 Bus Controller

2-9-1 Outline

The MN101C00 series has separate bus lines that are connected to internal memory and internal peripheral circuits to reduce the loading on each bus line and enhance operation speed.

There are four separate bus lines: ROM bus, RAM bus, peripheral expansion bus and external expansion bus. The bus control block controls the parallel operation of instruction read and data access, the access speed adjustment for low-speed external devices, and arbitration of bus access when using master devices on the external bus lines.

A functional block diagram of the bus controller is given below.

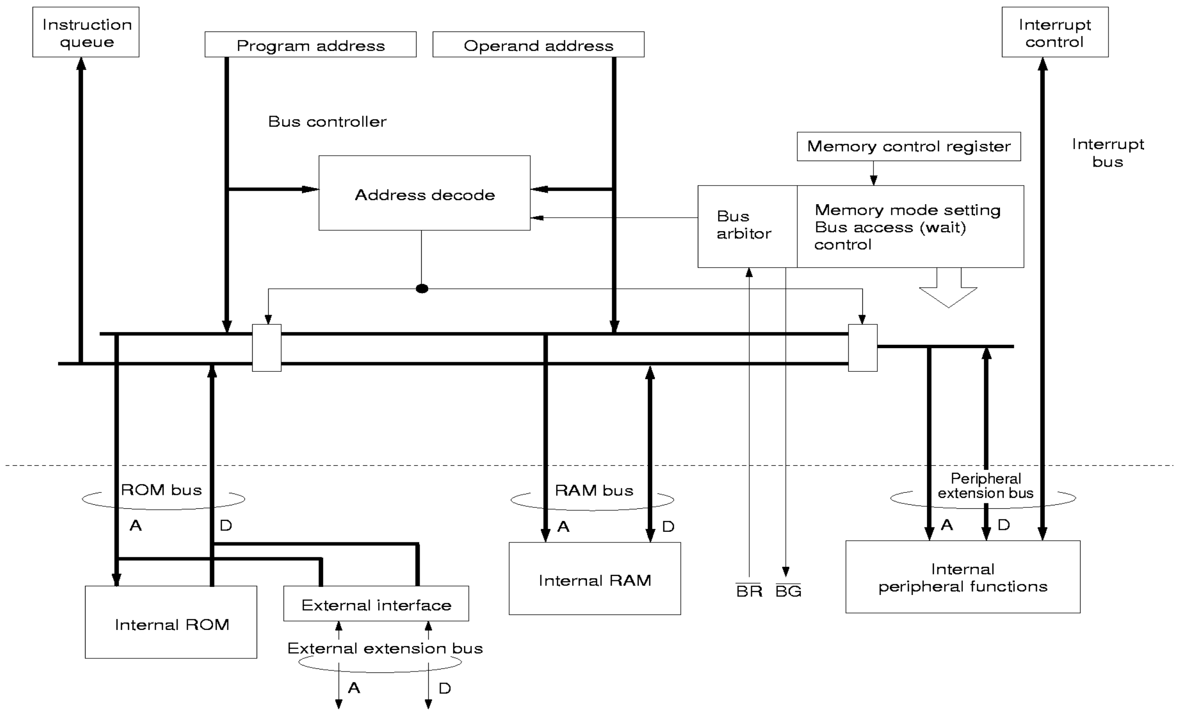


Fig. 2-9-1 Bus Controller Block Diagram

When the external expansion bus line is set to memory expansion mode or processor mode, data transfer will occur between the external expansion bus line and the external device. The external expansion bus line access sequence has two modes, fixed wait mode and handshake mode. The mode is selected using the memory control register (MEMCTR).

2-9-2 Fixed wait cycle mode

The fixed wait state mode (inserts 3 wait cycles) is selected at RESET. When handshake mode is selected or when manually reconfiguring the wait state, it is possible to change the wait cycle by setting the value to the MCR.

The MN101C00 series can connect low-speed devices (ROM, RAM, I/O expander, etc.) to the external expansion bus by inserting multiple wait cycles. The number of wait cycles can be selected by the memory control register (MEMCTR).

Fixed wait cycle mode is used to automatically insert the number of wait cycles specified by the fixed wait counter (WAITn, WAITIOn) in the MEMCTR.

	7							0
MEMCTR	IOW1	IOW0	IVBM	EXMEM	FEXW	IRWE	EXW1	EXW0
At reset:	1	1	0	0	1	0	1	1

EXW1 to 0	Number of fixed wait cycles	When bus cycle (20 MHz) oscillation
00	No wait cycles	100 ns
01	1 wait cycle	150 ns
10	2 wait cycles	200 ns
11	3 wait cycles	250 ns

IRWE	Interrupt request software write enable flag
0	Software writes disabled Writes to an interrupt control register (xxxICR) do not modify the state of the interrupt request flag (xxxIR).
1	Software writes enabled

FEXW	Fixed wait mode/handshake mode
0	Handshake mode
1	Fixed wait mode

EXMEM	External memory expansion mode
0	External memory not expanded
1	External memory expanded

IVBM	Base address of the interrupt vector table
0	Base address=x'04000'
1	Base address=x'00100'

IOW1 to 0	Number of I/O bus wait cycles	When bus cycle (20 MHz) oscillation
00	No wait cycles	100 ns
01	1 wait cycle	150 ns
10	2 wait cycles	200 ns
11	3 wait cycles	250 ns

Fig. 2-9-2 Memory Control Register (MEMCTR)

The wait cycle value written to the MEMCTR will be valid immediately after the wwrite instruction.

- The IOW1–IOW0 wait settings affect accesses to the special registers located at the addresses X'3F00'–X'3FFF'.
- The EXW1–EXW0 wait settings affect accesses to external devices in the processor and memory expansion modes.

Limit the setting of IRWE to "1" to initialization routines. After setting it to "1" to permit manipulation of interrupt control registers, always reset it to "0" when such manipulations are complete. Leaving it at "1" can lead to the loss of interrupt requests.

2-9-3 Handshake mode

Handshake mode uses the interlock control method in the data transfer sequence, with a transfer enable signals ($\overline{RE}$, $\overline{WE}$) and a data acknowledge signal ($\overline{DK}$).

The method is described below:

1) The CPU sets $\overline{RE}$ or $\overline{WE}$ to 'L'.
   $\overline{RE}$ is set when read, and $\overline{WE}$ is set when write.
2) The external device detects $\overline{RE}$='L' or $\overline{WE}$='L' and reacts as follows:
   When $\overline{RE}$='L' (Data Read: External Device > CPU):
   If the external device can send the data on the data bus line, it sets $\overline{DK}$='L'.
   When $\overline{WE}$='L' (Data Write: CPU > External Device):
   If the external device can read data from the data bus line, it sets $\overline{DK}$='L'.
3) The CPU detects $\overline{DK}$='L', sets $\overline{RE}$ or $\overline{WE}$ to 'H', and ends the bus cycle.
4) The external device detects $\overline{RE}$='L' or $\overline{WE}$='H' and immediately sets $\overline{DK}$='H'.

It is possible to use the Watch Dog Timer (WDT) to detect a malfunction (no $\overline{DK}$ signal response form the external device). The CPU waits for the $\overline{DK}$ signal until WDT overflows and then generates a non-maskable interrupt.

Handshake mode adjusts the wait cycle for each external device that has a different access speed when the $\overline{DK}$ generation circuit is provided for each device.

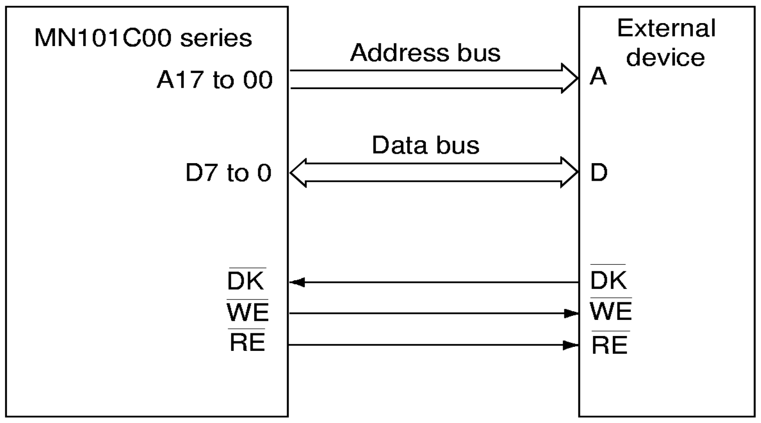


Fig. 2-9-3 Handshake Mode Pin Connection Example

2-10 DMA Support Function

2-10-1 Bus arbitration function*

The bus arbitration function handles bus usage rights between the MN101C00 series and external devices. The bus master request signal ($\overline{BR}$) and the bus master permission signal ($\overline{BG}$) are used. The CPU releases the line, and the external device which has received permission then executes DMA or other memory access.

**Not supported by all products.*

■ Bus Master Request Signal ($\overline{BR}$)

The external device generates this signal. The MN101C00 series samples the $\overline{BR}$ signal on the falling edge of the main oscillator while the system clock (PSYSCLK) is high. When the CPU detects $\overline{BR}$='L', it sets A23 to A00 and D15 to D00 to high impedance after completing the current bus cycle. It then sets $\overline{BG}$='L' and releases the bus line to the external device. When the CPU detects $\overline{BR}$='H', it sets $\overline{BG}$='H' on the next rising edge of the main oscillator and receives the rights back.

■ Bus Master Permission Signal ($\overline{BG}$)

The CPU generates this signal. The external device recognizes that the CPU has released the bus-line with the detection of $\overline{BG}$='L'. Always wait for $\overline{BG}$='L' before using the bus line.

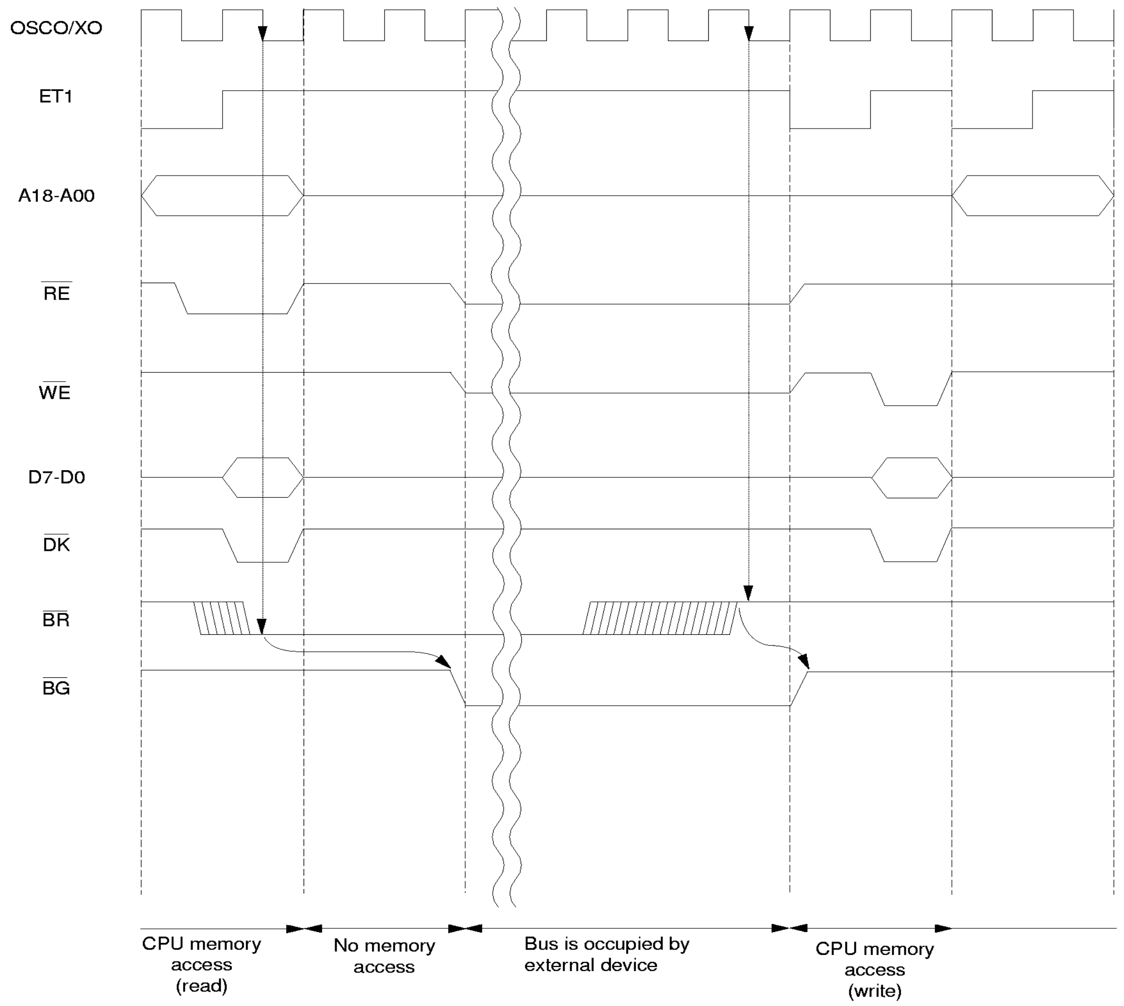


Fig 2-10-1 Bus Arbitration Timing

Appendices

3

MN101C00 SERIES INSTRUCTION SET

Group	Mnemonic	Operation	Flag				Code Size	Cycle	Re-peat	Machine Code												Notes	Page
			VF	NF	CF	ZF				Ext.	1	2	3	4	5	6	7	8	9	10	11		
Data Transfer Instructions																							
MOV	MOV Dn,Dm	Dn→Dm	—	—	—	—	2	1			1010	DnDm											25
	MOV imm8,Dm	imm8→Dm	—	—	—	—	4	2			1010	DmDm	<#8.	...>									25
	MOV Dn,PSW	Dn→PSW	●	●	●	●	3	3		0010	1001	01Dn											26
	MOV PSW,Dm	PSW→Dm	—	—	—	—	3	2		0010	0001	01Dm											26
	MOV (An),Dm	mem8(An)→Dm	—	—	—	—	2	2			0100	1ADm											27
	MOV (d8,An),Dm	mem8(d8+An)→Dm	—	—	—	—	4	2			0110	1ADm	<d8.	...>								*1	27
	MOV (d16,An),Dm	mem8(d16+An)→Dm	—	—	—	—	7	4		0010	0110	1ADm	<d16	....	....	...>							28
	MOV (d4,SP),Dm	mem8(d4+SP)→Dm	—	—	—	—	3	2			0110	01Dm	<d4>									*2	28
	MOV (d8,SP),Dm	mem8(d8+SP)→Dm	—	—	—	—	5	3		0010	0110	01Dm	<d8.	...>								*3	29
	MOV (d16,SP),Dm	mem8(d16+SP)→Dm	—	—	—	—	7	4		0010	0110	00Dm	<d16	....	....	...>							29
	MOV (io8),Dm	mem8(IOTOP+io8)→Dm	—	—	—	—	4	2			0110	00Dm	<io8	...>									30
	MOV (abs8),Dm	mem8(abs8)→Dm	—	—	—	—	4	2			0100	01Dm	<abs	8..>									30
	MOV (abs12),Dm	mem8(abs12)→Dm	—	—	—	—	5	2			0100	00Dm	<abs	12..	...>								31
	MOV (abs16),Dm	mem8(abs16)→Dm	—	—	—	—	7	4		0010	1100	00Dm	<abs	16..	....	...>							31
	MOV Dn,(Am)	Dn→mem8(Am)	—	—	—	—	2	2			0101	1aDn											32
	MOV Dn,(d8,Am)	Dn→mem8(d8+Am)	—	—	—	—	4	2			0111	1aDn	<d8.	...>								*1	32
	MOV Dn,(d16,Am)	Dn→mem8(d16+Am)	—	—	—	—	7	4		0010	0111	1aDn	<d16	....	....	...>							33
	MOV Dn,(d4,SP)	Dn→mem8(d4+SP)	—	—	—	—	3	2			0111	01Dn	<d4>									*2	33
	MOV Dn,(d8,SP)	Dn→mem8(d8+SP)	—	—	—	—	5	3		0010	0111	01Dn	<d8.	...>								*3	34
	MOV Dn,(d16,SP)	Dn→mem8(d16+SP)	—	—	—	—	7	4		0010	0111	00Dn	<d16	....	....	...>							34
	MOV Dn,(io8)	Dn→mem8(IOTOP+io8)	—	—	—	—	4	2			0111	00Dn	<io8	...>									35
	MOV Dn,(abs8)	Dn→mem8(abs8)	—	—	—	—	4	2			0101	01Dn	<abs	8..>									35
	MOV Dn,(abs12)	Dn→mem8(abs12)	—	—	—	—	5	2			0101	00Dn	<abs	12..	...>								36
	MOV Dn,(abs16)	Dn→mem8(abs16)	—	—	—	—	7	4		0010	1101	00Dn	<abs	16..	....	...>							36
	MOV imm8,(io8)	imm8→mem8(IOTOP+io8)	—	—	—	—	6	3			0000	0010	<io8	...>	<#8.	...>							37
	MOV imm8,(abs8)	imm8→mem8(abs8)	—	—	—	—	6	3			0001	0100	<abs	8..>	<#8.	...>							37
	MOV imm8,(abs12)	imm8→mem8(abs12)	—	—	—	—	7	3			0001	0101	<abs	12..	...>	<#8.	...>						38
	MOV imm8,(abs16)	imm8→mem8(abs16)	—	—	—	—	9	5		0011	1101	1001	<abs	16..	....	...>	<#8.	...>					38
	MOV Dn,(HA)	Dn→mem8(HA)	—	—	—	—	2	2			1101	00Dn											39
MOVW	MOVW (An),DWm	mem16(An)→DWm	—	—	—	—	2	3			1110	00Ad											40
	MOVW (An),Am	mem16(An)→Am	—	—	—	—	3	4		0010	1110	10Aa										*4	40
	MOVW (d4,SP),DWm	mem16(d4+SP)→DWm	—	—	—	—	3	3			1110	011d	<d4>									*2	41
	MOVW (d4,SP),Am	mem16(d4+SP)→Am	—	—	—	—	3	3			1110	010a	<d4>									*2	41
	MOVW (d8,SP),DWm	mem16(d8+SP)→DWm	—	—	—	—	5	4		0010	1110	011d	<d8.	...>								*3	42
	MOVW (d8,SP),Am	mem16(d8+SP)→Am	—	—	—	—	5	4		0010	1110	010a	<d8.	...>								*3	42
	MOVW (d16,SP),DWm	mem16(d16+SP)→DWm	—	—	—	—	7	5		0010	1110	001d	<d16	....	....	...>							43
	MOVW (d16,SP),Am	mem16(d16+SP)→Am	—	—	—	—	7	5		0010	1110	000a	<d16	....	....	...>							43
	MOVW (abs8),DWm	mem16(abs8)→DWm	—	—	—	—	4	3			1100	011d	<abs	8..>									44
	MOVW (abs8),Am	mem16(abs8)→Am	—	—	—	—	4	3			1100	010a	<abs	8..>									44
	MOVW (abs16),DWm	mem16(abs16)→DWm	—	—	—	—	7	5		0010	1100	011d	<abs	16..	....	...>							45
	MOVW (abs16),Am	mem16(abs16)→Am	—	—	—	—	7	5		0010	1100	010a	<abs	16..	....	...>							45
	MOVW DWn,(Am)	DWn→mem16(Am)	—	—	—	—	2	3			1111	00aD											46
	MOVW An,(Am)	An→mem16(Am)	—	—	—	—	3	4		0010	1111	10aA										*4	46
	MOVW DWn,(d4,SP)	DWn→mem16(d4+SP)	—	—	—	—	3	3			1111	011D	<d4>									*2	47
	MOVW An,(d4,SP)	An→mem16(d4+SP)	—	—	—	—	3	3			1111	010A	<d4>									*2	47
	MOVW DWn,(d8,SP)	DWn→mem16(d8+SP)	—	—	—	—	5	4		0010	1111	011D	<d8.	...>								*3	48
	MOVW An,(d8,SP)	An→mem16(d8+SP)	—	—	—	—	5	4		0010	1111	010A	<d8.	...>								*3	48
	MOVW DWn,(d16,SP)	DWn→mem16(d16+SP)	—	—	—	—	7	5		0010	1111	001D	<d16	....	....	...>							49
	MOVW An,(d16,SP)	An→mem16(d16+SP)	—	—	—	—	7	5		0010	1111	000A	<d16	....	....	...>							49
	MOVW DWn,(abs8)	DWn→mem16(abs8)	—	—	—	—	4	3			1101	011D	<abs	8..>									50
	MOVW An,(abs8)	An→mem16(abs8)	—	—	—	—	4	3			1101	010A	<abs	8..>									50
	MOVW DWn,(abs16)	DWn→mem16(abs16)	—	—	—	—	7	5		0010	1101	011D	<abs	16..	....	...>							51
	MOVW An,(abs16)	An→mem16(abs16)	—	—	—	—	7	5		0010	1101	010A	<abs	16..	....	...>							51
	MOVW DWn,(HA)	DWn→mem16(HA)	—	—	—	—	2	3			1001	010D											52
	MOVW An,(HA)	An→mem16(HA)	—	—	—	—	2	3			1001	011A											52
	MOVW imm8,DWm	sign(imm8)→DWm	—	—	—	—	4	2			0000	110d	<#8.	...>								*5	53
	MOVW imm8,Am	zero(imm8)→Am	—	—	—	—	4	2			0000	111a	<#8.	...>								*6	53
	MOVW imm16,DWm	imm16→DWm	—	—	—	—	6	3			1100	111d	<#16	....	....	...>							54

Note: Page refers to the corresponding page in the Instruction Manual.

*1 d8 sign extended
*2 d4 zero extended
*3 d8 zero extended
*4 A=An, a=Am
*5 #8 sign extended
*6 #8 zero extended

MN101C00 SERIES INSTRUCTION SET

Group	Mnemonic	Operation	Flag				Code Size	Cycle	Re-peat	Machine Code												Notes	Page
			VF	NF	CF	ZF				Ext.	1	2	3	4	5	6	7	8	9	10	11		
	MOVW imm16,Am	imm16→Am	—	—	—	—	6	3			1101	111a	<#16	....	....	...>							54
	MOVW SP,Am	SP→Am	—	—	—	—	3	3		0010	0000	100a											55
	MOVW An,SP	An→SP	—	—	—	—	3	3		0010	0000	101A											55
	MOVW DWn,DWm	DWn→DWm	—	—	—	—	3	3		0010	1000	00Dd										*1	56
	MOVW DWn,Am	DWn→Am	—	—	—	—	3	3		0010	0100	11Da											56
	MOVW An,DWm	An→DWm	—	—	—	—	3	3		0010	1100	11Ad											57
	MOVW An,Am	An→Am	—	—	—	—	3	3		0010	0000	00Aa										*2	57
PUSH	PUSH Dn	SP-1→SP,Dn→mem8(SP)	—	—	—	—	2	3			1111	10Dn											58
	PUSH An	SP-2→SP,An→mem16(SP)	—	—	—	—	2	5			0001	011A											58
POP	POP Dn	mem8(SP)→Dn,SP+1→SP	—	—	—	—	2	3			1110	10Dn											59
	POP An	mem16(SP)→An,SP+2→SP	—	—	—	—	2	4			0000	011A											59
EXT	EXT Dn,DWm	sign(Dn)→DWm	—	—	—	—	3	3		0010	1001	000d										*3	60

Arithmetic Operation Instructions

Group	Mnemonic	Operation	VF	NF	CF	ZF	Code Size	Cycle	Re-peat	Ext.	1	2	3	4	5	6	7	8	9	10	11	Notes	Page
ADD	ADD Dn,Dm	Dm+Dn→Dm	●	●	●	●	3	2	✓	0011	0011	DnDm											61
	ADD imm4,Dm	Dm+sign(imm4)→Dm	●	●	●	●	3	2			1000	00Dm	<#4>									*6	61
	ADD imm8,Dm	Dm+imm8→Dm	●	●	●	●	4	2			0000	10Dm	<#8.	...>									62
ADDC	ADDC Dn,Dm	Dm+Dn+CF→Dm	●	●	●	●	3	2	✓	0011	1011	DnDm											63
ADDW	ADDW DWn,DWm	DWm+DWn→DWm	●	●	●	●	3	3	✓	0010	0101	00Dd										*1	64
	ADDW DWn,Am	Am+DWn→Am	●	●	●	●	3	3	✓	0010	0101	10Da											64
	ADDW imm4,Am	Am+sign(imm4)→Am	●	●	●	●	3	2			1110	110a	<#4>									*6	65
	ADDW imm8,Am	Am+sign(imm8)→Am	●	●	●	●	5	3		0010	1110	110a	<#8.	...>								*7	65
	ADDW imm16,Am	Am+imm16→Am	●	●	●	●	7	4		0010	0101	011a	<#16	....	....	...>							66
	ADDW imm4,SP	SP+sign(imm4)→SP	—	—	—	—	3	2			1111	1101	<#4>									*6	66
	ADDW imm8,SP	SP+sign(imm8)→SP	—	—	—	—	4	2			1111	1100	<#8.	...>								*7	67
	ADDW imm16,SP	SP+imm16→SP	—	—	—	—	7	4		0010	1111	1100	<#16	....	....	...>							67
	ADDW imm16,DWm	DWm+imm16→DWm	●	●	●	●	7	4		0010	0101	010d	<#16	....	....	...>							68
ADDUW	ADDUW Dn,Am	Am+zero(Dn)→Am	●	●	●	●	3	3	✓	0010	1000	1aDn										*8	69
ADDSW	ADDSW Dn,Am	Am+sign(Dn)→Am	●	●	●	●	3	3	✓	0010	1001	1aDn											70
SUB	SUB Dn,Dm(when Dn≠Dm)	Dm-Dn→Dm	●	●	●	●	3	2	✓	0010	1010	DnDm											71
	SUB Dn,Dn	Dn-Dn→Dn	0	0	0	1	2	1			1000	01Dn											71
	SUB imm8,Dm	Dm-imm8→Dm	●	●	●	●	5	3		0010	1010	DmDm	<#8.	...>									72
SUBC	SUBC Dn,Dm	Dm-Dn-CF→Dm	●	●	●	●	3	2	✓	0010	1011	DnDm											73
SUBW	SUBW DWn,DWm	DWm-DWn→DWm	●	●	●	●	3	3		0010	0100	00Dd										*1	74
	SUBW DWn,Am	Am-DWn→Am	●	●	●	●	3	3		0010	0100	10Da											74
	SUBW imm16,DWm	DWm-imm16→DWm	●	●	●	●	7	4		0010	0100	010d	<#16	....	....	...>							75
	SUBW imm16,Am	Am-imm16→Am	●	●	●	●	7	4		0010	0100	011a	<#16	....	....	...>							75
MULU	MULU Dn,Dm	Dm*Dn→DWk	0	●	●	●	3	8		0010	1111	111D										*4	76
DIVU	DIVU Dn,DWm	DWm/Dn→DWm-l...DWm-h	●	●	●	●	3	9		0010	1110	111d										*5	77
CMP	CMP Dn,Dm	Dm-Dn...PSW	●	●	●	●	3	2		0011	0010	DnDm											78
	CMP imm8,Dm	Dm-imm8...PSW	●	●	●	●	4	2			1100	00Dm	<#8.	...>									78
	CMP imm8,(abs8)	mem8(abs8)-imm8...PSW	●	●	●	●	6	3			0000	0100	<abs	8..>	<#8.	...>							79
	CMP imm8,(abs12)	mem8(abs12)-imm8...PSW	●	●	●	●	7	3			0000	0101	<abs	12..	...>	<#8.	...>						79
	CMP imm8,(abs16)	mem8(abs16)-imm8...PSW	●	●	●	●	9	5		0011	1101	1000	<abs	16..	....	...>	<#8.	...>					80
CMPW	CMPW DWn,DWm	DWm-DWn...PSW	●	●	●	●	3	3		0010	1000	01Dd										*1	81
	CMPW DWn,Am	Am-DWn...PSW	●	●	●	●	3	3		0010	0101	11Da											81
	CMPW An,Am	Am-An...PSW	●	●	●	●	3	3		0010	0000	01Aa										*2	82
	CMPW imm16,DWm	DWm-imm16...PSW	●	●	●	●	6	3			1100	110d	<#16	....	....	...>							82
	CMPW imm16,Am	Am-imm16...PSW	●	●	●	●	6	3			1101	110a	<#16	....	....	...>							83

Logical Operation Instructions

Group	Mnemonic	Operation	VF	NF	CF	ZF	Code Size	Cycle	Re-peat	Ext.	1	2	3	4	5	6	7	8	9	10	11	Notes	Page
AND	AND Dn,Dm	Dm&Dn→Dm	0	●	0	●	3	2		0011	0111	DnDm											84
	AND imm8,Dm	Dm&imm8→Dm	0	●	0	●	4	2			0001	11Dm	<#8.	...>									84
	AND imm8,PSW	PSW&imm8→PSW	●	●	●	●	5	3		0010	1001	0010	<#8.	...>									85
OR	OR Dn,Dm	DmIDn→Dm	0	●	0	●	3	2		0011	0110	DnDm											86
	OR imm8,Dm	DmIimm8→Dm	0	●	0	●	4	2			0001	10Dm	<#8.	...>									86
	OR imm8,PSW	PSWIimm8→PSW	●	●	●	●	5	3		0010	1001	0011	<#8.	...>									87
XOR	XOR Dn,Dm	Dm^Dn→Dm	0	●	0	●	3	2		0011	1010	DnDm										*9	88
	XOR imm8,Dm	Dm^imm8→Dm	0	●	0	●	5	3		0011	1010	DmDm	<#8.	...>									88

Note: Page refers to the corresponding page in the Instruction Manual.

*1 D=DWn, d=DWm	*5 D=DWm	*9 m≠n
*2 A=An, a=Am	*6 #4 sign extended	
*3 d=DWm	*7 #8 sign extended	
*4 D=DWk	*8 Dn zero extended	

MN101C00 SERIES INSTRUCTION SET

Group	Mnemonic	Operation	Flag				Code Size	Cycle	Re-peat	Machine Code												Notes	Page
			VF	NF	CF	ZF				Ext.	1	2	3	4	5	6	7	8	9	10	11		
NOT	NOT Dn	$\overline{Dn}$→Dn	0	●	0	●	3	2		0010	0010	10Dn											89
ASR	ASR Dn	Dn.msb→temp,Dn.lsb→CF Dn>>1→Dn,temp→Dn.msb	0	—	●	●	3	2	✔	0010	0011	10Dn											90
LSR	LSR Dn	Dn.lsb→CF,Dn>>1→Dn 0→Dn.msb	0	0	●	●	3	2	✔	0010	0011	11Dn											91
ROR	ROR Dn	Dn.lsb→temp,Dn>>1→Dn CF→Dn.msb,temp→CF	0	●	●	●	3	2	✔	0010	0010	11Dn											92

Bit Manipulation Instructions

Group	Mnemonic	Operation	VF	NF	CF	ZF	Code Size	Cycle	Re-peat	Ext.	1	2	3	4	5	6	7	8	9	10	11	Notes	Page
BSET	BSET (io8)bp	mem8(IOTOP+io8)&bpdata...PSW 1→mem8(IOTOP+io8)bp	0	●	0	●	5	5		0011	1000	0bp.	<io8	...>									93
	BSET (abs8)bp	mem8(abs8)&bpdata...PSW 1→mem8(abs8)bp	0	●	0	●	4	4			1011	0bp.	<abs	8..>									93
	BSET (abs16)bp	mem8(abs16)&bpdata...PSW 1→mem8(abs16)bp	0	●	0	●	7	6		0011	1100	0bp.	<abs	16..	....	...>							94
BCLR	BCLR (io8)bp	mem8(IOTOP+io8)&bpdata...PSW 0→mem8(IOTOP+io8)bp	0	●	0	●	5	5		0011	1000	1bp.	<io8	...>									95
	BCLR (abs8)bp	mem8(abs8)&bpdata...PSW 0→mem8(abs8)bp	0	●	0	●	4	4			1011	1bp.	<abs	8..>									95
	BCLR (abs16)bp	mem8(abs16)&bpdata...PSW 0→mem8(abs16)bp	0	●	0	●	7	6		0011	1100	1bp.	<abs	16..	....	...>							96
BTST	BTST imm8,Dm	Dm&imm8...PSW	0	●	0	●	5	3		0010	0000	11Dm	<#8.	...>									97
	BTST (abs16)bp	mem8(abs16)&bpdata...PSW	0	●	0	●	7	5		0011	1101	0bp.	<abs	16..	....	...>							97

Branch Instructions

Group	Mnemonic	Operation	VF	NF	CF	ZF	Code Size	Cycle	Re-peat	Ext.	1	2	3	4	5	6	7	8	9	10	11	Notes	Page
Bcc	BEQ label	if(ZF=1), PC+3+d4(label)+H→PC if(ZF=0), PC+3→PC	—	—	—	—	3	2/3			1001	000H	<d4>									*1	98
	BEQ label	if(ZF=1), PC+4+d7(label)+H→PC if(ZF=0), PC+4→PC	—	—	—	—	4	2/3			1000	1010	<d7.	...H								*2	98
	BEQ label	if(ZF=1), PC+5+d11(label)+H→PC if(ZF=0), PC+5→PC	—	—	—	—	5	2/3			1001	1010	<d11	....	...H							*3	99
	BNE label	if(ZF=0), PC+3+d4(label)+H→PC if(ZF=1), PC+3→PC	—	—	—	—	3	2/3			1001	001H	<d4>									*1	100
	BNE label	if(ZF=0), PC+4+d7(label)+H→PC if(ZF=1), PC+4→PC	—	—	—	—	4	2/3			1000	1011	<d7.	...H								*2	100
	BNE label	if(ZF=0), PC+5+d11(label)+H→PC if(ZF=1), PC+5→PC	—	—	—	—	5	2/3			1001	1011	<d11	....	...H							*3	101
	BGE label	if((VF^NF)=0),PC+4+d7(label)+H→PC if((VF^NF)=1),PC+4→PC	—	—	—	—	4	2/3			1000	1000	<d7.	...H								*2	102
	BGE label	if((VF^NF)=0),PC+5+d11(label)+H→PC if((VF^NF)=1),PC+5→PC	—	—	—	—	5	2/3			1001	1000	<d11	....	...H							*3	102
	BCC label	if(CF=0),PC+4+d7(label)+H→PC if(CF=1), PC+4→PC	—	—	—	—	4	2/3			1000	1100	<d7.	...H								*2	103
	BCC label	if(CF=0), PC+5+d11(label)+H→PC if(CF=1), PC+5→PC	—	—	—	—	5	2/3			1001	1100	<d11	....	...H							*3	103
	BCS label	if(CF=1),PC+4+d7(label)+H→PC if(CF=0), PC+4→PC	—	—	—	—	4	2/3			1000	1101	<d7.	...H								*2	104
	BCS label	if(CF=1), PC+5+d11(label)+H→PC if(CF=0), PC+5→PC	—	—	—	—	5	2/3			1001	1101	<d11	....	...H							*3	104
	BLT label	if((VF^NF)=1),PC+4+d7(label)+H→PC if((VF^NF)=0),PC+4→PC	—	—	—	—	4	2/3			1000	1110	<d7.	...H								*2	105
	BLT label	if((VF^NF)=1),PC+5+d11(label)+H→PC if((VF^NF)=0),PC+5→PC	—	—	—	—	5	2/3			1001	1110	<d11	....	...H							*3	105
	BLE label	if((VF^NF)|ZF=1),PC+4+d7(label)+H→PC if((VF^NF)|ZF=0),PC+4→PC	—	—	—	—	4	2/3			1000	1111	<d7.	...H								*2	106
	BLE label	if((VF^NF)|ZF=1),PC+5+d11(label)+H→PC if((VF^NF)|ZF=0),PC+5→PC	—	—	—	—	5	2/3			1001	1111	<d11	....	...H							*3	106
	BGT label	if((VF^NF)|ZF=0),PC+5+d7(label)+H→PC if((VF^NF)|ZF=1),PC+5→PC	—	—	—	—	5	3/4		0010	0010	0001	<d7.	...H								*2	107

Note: Page refers to the corresponding page in the Instruction Manual.

*1 d4 sign extended
*2 d7 sign extended
*3 d11 sign extended

MN101C00 SERIES INSTRUCTION SET

Group	Mnemonic	Operation	Flag				Code Size	Cycle	Re-peat	Machine Code												Notes	Page
			VF	NF	CF	ZF				Ext.	1	2	3	4	5	6	7	8	9	10	11		
Bcc	BGT label	if((VF^NF)|ZF=0),PC+6+d11(label)+H→PC if((VF^NF)|ZF=1),PC+6→PC	—	—	—	—	6	3/4		0010	0011	0001	<d11	....	...H							*3	107
	BHI label	if(CFIZF=0),PC+5+d7(label)+H→PC if(CFIZF=1), PC+5→PC	—	—	—	—	5	3/4		0010	0010	0010	<d7.	...H								*2	108
	BHI label	if(CFIZF=0),PC+6+d11(label)+H→PC if(CFIZF=1), PC+6→PC	—	—	—	—	6	3/4		0010	0011	0010	<d11	....	...H							*3	108
	BLS label	if(CFIZF=1),PC+5+d7(label)+H→PC if(CFIZF=0), PC+5→PC	—	—	—	—	5	3/4		0010	0010	0011	<d7.	...H								*2	109
	BLS label	if(CFIZF=1),PC+6+d11(label)+H→PC if(CFIZF=0), PC+6→PC	—	—	—	—	6	3/4		0010	0011	0011	<d11	....	...H							*3	109
	BNC label	if(NF=0),PC+5+d7(label)+H→PC if(NF=1),PC+5→PC	—	—	—	—	5	3/4		0010	0010	0100	<d7.	...H								*2	110
	BNC label	if(NF=0),PC+6+d11(label)+H→PC if(NF=1),PC+6→PC	—	—	—	—	6	3/4		0010	0011	0100	<d11	....	...H							*3	110
	BNS label	if(NF=1),PC+5+d7(label)+H→PC if(NF=0),PC+5→PC	—	—	—	—	5	3/4		0010	0010	0101	<d7.	...H								*2	111
	BNS label	if(NF=1),PC+6+d11(label)+H→PC if(NF=0),PC+6→PC	—	—	—	—	6	3/4		0010	0011	0101	<d11	....	...H							*3	111
	BVC label	if(VF=0),PC+5+d7(label)+H→PC if(VF=1),PC+5→PC	—	—	—	—	5	3/4		0010	0010	0110	<d7.	...H								*2	112
	BVC label	if(VF=0),PC+6+d11(label)+H→PC if(VF=1),PC+6→PC	—	—	—	—	6	3/4		0010	0011	0110	<d11	....	...H							*3	112
	BVS label	if(VF=1),PC+5+d7(label)+H→PC if(VF=0),PC+5→PC	—	—	—	—	5	3/4		0010	0010	0111	<d7.	...H								*2	113
	BVS label	if(VF=1),PC+6+d11(label)+H→PC if(VF=0),PC+6→PC	—	—	—	—	6	3/4		0010	0011	0111	<d11	....	...H							*3	113
	BRA label	PC+3+d4(label)+H→PC	—	—	—	—	3	3			1110	111H	<d4>									*1	114
	BRA label	PC+4+d7(label)+H→PC	—	—	—	—	4	3			1000	1001	<d7.	...H								*2	114
	BRA label	PC+5+d11(label)+H→PC	—	—	—	—	5	3			1001	1001	<d11	....	...H							*3	115
CBEQ	CBEQ imm8,Dm,label	if(Dm=imm8),PC+6+d7(label)+H→PC if(Dm≠imm8),PC+6→PC	●	●	●	●	6	3/4			1100	10Dm	<#8.	...>	<d7.	...H						*2	116
	CBEQ imm8,Dm,label	if(Dm=imm8),PC+8+d11(label)+H→PC if(Dm≠imm8),PC+8→PC	●	●	●	●	8	4/5		0010	1100	10Dm	<#8.	...>	<d11	....	...H					*3	116
	CBEQ imm8,(abs8),label	if(mem8(abs8)=imm8),PC+9+d7(label)+H→PC if(mem8(abs8)≠imm8),PC+9→PC	●	●	●	●	9	6/7		0010	1101	1100	<abs	8..>	<#8.	...>	<d7.	...H				*2	117
	CBEQ imm8,(abs8),label	if(mem8(abs8)=imm8),PC+10+d11(label)+H→PC if(mem8(abs8)≠imm8),PC+10→PC	●	●	●	●	10	6/7		0010	1101	1101	<abs	8..>	<#8.	...>	<d11	....	...H			*3	117
	CBEQ imm8,(abs16),label	if(mem8(abs16)=imm8),PC+11+d7(label)+H→PC if(mem8(abs16)≠imm8),PC+11→PC	●	●	●	●	11	7/8		0011	1101	1100	<abs	16..	....	...>	<#8.	...>	<d7.	...H		*2	118
	CBEQ imm8,(abs16),label	if(mem8(abs16)=imm8),PC+12+d11(label)+H→PC if(mem8(abs16)≠imm8),PC+12→PC	●	●	●	●	12	7/8		0011	1101	1101	<abs	16..	....	...>	<#8.	...>	<d11	....	...H	*3	118
CBNE	CBNE imm8,Dm,label	if(Dm≠imm8),PC+6+d7(label)+H→PC if(Dm=imm8),PC+6→PC	●	●	●	●	6	3/4			1101	10Dm	<#8.	...>	<d7.	..H>						*2	119
	CBNE imm8,Dm,label	if(Dm≠imm8),PC+8+d11(label)+H→PC if(Dm=imm8),PC+8→PC	●	●	●	●	8	4/5		0010	1101	10Dm	<#8.	...>	<d11	....	...H					*3	119
	CBNE imm8,(abs8),label	if(mem8(abs8)≠imm8),PC+9+d7(label)+H→PC if(mem8(abs8)=imm8),PC+9→PC	●	●	●	●	9	6/7		0010	1101	1110	<abs	8..>	<#8.	...>	<d7.	...H				*2	120
	CBNE imm8,(abs8),label	if(mem8(abs8)≠imm8),PC+10+d11(label)+H→PC if(mem8(abs8)=imm8),PC+10→PC	●	●	●	●	10	6/7		0010	1101	1111	<abs	8..>	<#8.	...>	<d11	....	...H			*3	120
	CBNE imm8,(abs16),label	if(mem8(abs16)≠imm8),PC+11+d7(label)+H→PC if(mem8(abs16)=imm8),PC+11→PC	●	●	●	●	11	7/8		0011	1101	1110	<abs	16..	....	...>	<#8.	...>	<d7.	...H		*2	121
	CBNE imm8,(abs16),label	if(mem8(abs16)≠imm8),PC+12+d11(label)+H→PC if(mem8(abs16)=imm8),PC+12→PC	●	●	●	●	12	7/8		0011	1101	1111	<abs	16..	....	...>	<#8.	...>	<d11	....	...H	*3	121
TBZ	TBZ (abs8)bp,label	if(mem8(abs8)bp=0),PC+7+d7(label)+H→PC if(mem8(abs8)bp=1),PC+7→PC	0	●	0	●	7	6/7		0011	0000	0bp.	<abs	8..>	<d7.	...H						*2	122
	TBZ (abs8)bp,label	if(mem8(abs8)bp=0),PC+8+d11(label)+H→PC if(mem8(abs8)bp=1),PC+8→PC	0	●	0	●	8	6/7		0011	0000	1bp.	<abs	8..>	<d11	....	...H					*3	122

Note: Page refers to the corresponding page in the Instruction Manual.

*1 d4 sign extended
*2 d7 sign extended
*3 d11 sign extended

MN101C00 SERIES INSTRUCTION SET

Group	Mnemonic	Operation	Flag				Code Size	Cycle	Re-peat	Machine Code												Notes	Page
			VF	NF	CF	ZF				Ext.	1	2	3	4	5	6	7	8	9	10	11		
TBZ	TBZ (io8)bp,label	if(mem8(IOTOP+io8)bp=0),PC+7+d7(label)+H→PC if(mem8(IOTOP+io8)bp=1),PC+7→PC	0	●	0	●	7	6/7		0011	0100	0bp.	<io8	...>	<d7.	...H						*1	123
	TBZ (io8)bp,label	if(mem8(IOTOP+io8)bp=0),PC+8+d11(label)+H→PC if(mem8(IOTOP+io8)bp=1),PC+8→PC	0	●	0	●	8	6/7		0011	0100	1bp.	<io8	...>	<d11	....	...H					*2	123
	TBZ (abs16)bp,label	if(mem8(abs16)bp=0),PC+9+d7(label)+H→PC if(mem8(abs16)bp=1),PC+9→PC	0	●	0	●	9	7/8		0011	1110	0bp.	<abs	16..	....	...>	<d7.	...H				*1	124
	TBZ (abs16)bp,label	if(mem8(abs16)bp=0),PC+10+d11(label)+H→PC if(mem8(abs16)bp=1),PC+10→PC	0	●	0	●	10	7/8		0011	1110	1bp.	<abs	16..	....	...>	<d11	....	...H			*2	124
TBNZ	TBNZ (abs8)bp,label	if(mem8(abs8)bp=1),PC+7+d7(label)+H→PC if(mem8(abs8)bp=0),PC+7→PC	0	●	0	●	7	6/7		0011	0001	0bp.	<abs	8..>	<d7.	...H						*1	125
	TBNZ (abs8)bp,label	if(mem8(abs8)bp=1),PC+8+d11(label)+H→PC if(mem8(abs8)bp=0),PC+8→PC	0	●	0	●	8	6/7		0011	0001	1bp.	<abs	8..>	<d11	....	...H					*2	125
	TBNZ (io8)bp,label	if(mem8(io)bp=1),PC+7+d7(label)+H→PC if(mem8(io)bp=0),PC+7→PC	0	●	0	●	7	6/7		0011	0101	0bp.	<io8	...>	<d7.	...H						*1	126
	TBNZ (io8)bp,label	if(mem8(io)bp=1),PC+8+d11(label)+H→PC if(mem8(io)bp=0),PC+8→PC	0	●	0	●	8	6/7		0011	0101	1bp.	<io8	...>	<d11	....	...H					*2	126
	TBNZ (abs16)bp,label	if(mem8(abs16)bp=1),PC+9+d7(label)+H→PC if(mem8(abs16)bp=0),PC+9→PC	0	●	0	●	9	7/8		0011	1111	0bp.	<abs	16..	....	...>	<d7.	...H				*1	127
	TBNZ (abs16)bp,label	if(mem8(abs16)bp=1),PC+10+d11(label)+H→PC if(mem8(abs16)bp=0),PC+10→PC	0	●	0	●	10	7/8		0011	1111	1bp.	<abs	16..	....	...>	<d11	....	...H			*2	127
JMP	JMP (An)	0→PC.17–16,An→PC.15–0,0→PC.H	—	—	—	—	3	4		0010	0001	00A0											128
	JMP label	abs18(label)+H→PC	—	—	—	—	7	5		0011	1001	0aaH	<abs	18.b	p15-	0..>						*5	128
JSR	JSR (An)	SP-3→SP,(PC+3).bp7–0→mem8(SP) (PC+3).bp15–8→mem8(SP+1) (PC+3).H→mem8(SP+2).bp7, 0→mem8(SP+2).bp6–2, (PC+3).bp17–16→mem8(SP+2).bp1–0 0→PC.bp17–16 An→PC.bp15–0,0→PC.H	—	—	—	—	3	7		0010	0001	00A1											129
	JSR label	SP-3→SP,(PC+5).bp7–0→mem8(SP) (PC+5).bp15–8→mem8(SP+1) (PC+5).H→mem8(SP+2).bp7, 0→mem8(SP+2).bp6–2, (PC+5).bp17–16→mem8(SP+2).bp1–0 PC+5+d12(label)+H→PC	—	—	—	—	5	6			0001	000H	<d12	....	...>							*3	129
	JSR label	SP-3→SP,(PC+6).bp7–0→mem8(SP) (PC+6).bp15–8→mem8(SP+1) (PC+6).H→mem8(SP+2).bp7, 0→mem8(SP+2).bp6–2, (PC+6).bp17–16→mem8(SP+2).bp1–0 PC+6+d16(label)+H→PC	—	—	—	—	6	7			0001	001H	<d16	....	....	...>						*4	130
	JSR label	SP-3→SP,(PC+7).bp7–0→mem8(SP) (PC+7).bp15–8→mem8(SP+1) (PC+7).H→mem8(SP+2).bp7, 0→mem8(SP+2).bp6–2, (PC+7).bp17–16→mem8(SP+2).bp1–0 abs18(label)+H→PC	—	—	—	—	7	8		0011	1001	1aaH	<abs	18.b	p15-	0..>						*5	130
	JSRV (tbl4)	SP-3→SP,(PC+3).bp7–0→mem8(SP) (PC+3).bp15–8→mem8(SP+1) (PC+3).H→mem8(SP+2).bp7 0→mem8(SP+2).bp6–2, (PC+3).bp17–16→mem8(SP+2).bp1–0 mem8(x'004080+tbl4<<2)→PC.bp7–0 mem8(x'004080+tbl4<<2+1)→PC.bp15–8 mem8(x'004080+tbl4<<2+2).bp7→PC.H mem8(x'004080+tbl4<<2+2).bp1–0→ PC.bp17-16	—	—	—	—	3	9			1111	1110	<t4>										131
NOP	NOP	PC+2→PC	—	—	—	—	2	1	✔		0000	0000											132

Note: Page refers to the corresponding page in the Instruction Manual.

*1 d7 sign extended
*2 d11 sign extended
*3 d12 sign extended
*4 d16 sign extended
*5 aa=abs18.17-16

MN101C00 SERIES INSTRUCTION SET

Group	Mnemonic	Operation	Flag				Code Size	Cycle	Re-peat	Machine Code												Notes	Page
			VF	NF	CF	ZF				Ext.	1	2	3	4	5	6	7	8	9	10	11		
RTS	RTS	mem8(SP)→(PC).bp7–0 mem8(SP+1)→(PC+3).bp15–8 mem8(SP+2).bp7→(PC+3).H mem8(SP+2).bp1–0→(PC+3).bp17–16 SP+3→SP	—	—	—	—	2	7			0000	0001											133
RTI	RTI	mem8(SP)→PSW mem8(SP+1)→(PC).bp7–0 mem8(SP+2)→(PC+3).bp15–8 mem8(SP+3).bp7→(PC+3).H mem8(SP+3).bp1–0→(PC+3).bp17–16 mem8(SP+4)→HA-l mem8(SP+5)→HA-h SP+6→SP	●	●	●	●	2	11			0000	0011											134

Control Instructions

Group	Mnemonic	Operation	VF	NF	CF	ZF	Code Size	Cycle	Re-peat	Ext.	1	2	3	4	5	6	7	8	9	10	11	Notes	Page
REP	REP imm3	imm3→RPC	—	—	—	—	3	2		0010	0001	1rep										*1	135

Note: Page refers to the corresponding page in the Instruction Manual.

*1 No repeat when imm3=0

MN101C00 SERIES INSTRUCTION MAP

1st nibble/2nd nibble

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	RTS	MOV #8,(io8)	RTI	CMP #8,(abs8)/(abs12)		POP An		ADD #8,Dm				MOVW #8,DWm		MOVW #8,Am	
1	JSR d12(label)		JSR d16(label)		MOV #8,(abs8)/(abs12)		PUSH An		OR #8,Dm				AND #8,Dm			
2	When extended code is b'0010'															
3	When extended code is b'0011'															
4	MOV (abs12),Dm				MOV (abs8),Dm				MOV (An),Dm							
5	MOV Dn,(abs12)				MOV Dn,(abs8)				MOV Dn,(Am)							
6	MOV (io8),Dm				MOV (d4,SP),Dm				MOV (d8,An),Dm							
7	MOV Dn,(io8)				MOV Dn,(d4,SP)				MOV Dn,(d8,Am)							
8	ADD #4,Dm				SUB Dn,Dn				BGE d7	BRA d7	BEQ d7	BNE d7	BCC d7	BCS d7	BLT d7	BLE d7
9	BEQ d4		BNE d4		MOVW DWn,(HA)		MOVW An,(HA)		BGE d11	BRA d11	BEQ d11	BNE d11	BCC d11	BCS d11	BLT d11	BLE d11
A	MOV Dn,Dm / MOV #8,Dm															
B	BSET (abs8)bp								BCLR (abs8)bp							
C	CMP #8,Dm				MOVW (abs8),Am		MOVW (abs8),DWm		CBEQ #8,Dm,d7				CMPW #16,DWm		MOVW #16,DWm	
D	MOV Dn,(HA)				MOVW An,(abs8)		MOVW DWn,(abs8)		CBNE #8,Dm,d7				CMPW #16,Am		MOVW #16,Am	
E	MOVW (An),DWm				MOVW (d4,SP),Am		MOVW (d4,SP),DWm		POP Dn				ADDW #4,Am		BRA d4	
F	MOVW DWn,(Am)				MOVW An,(d4,SP)		MOVW DWn,(d4,SP)		PUSH Dn				ADDW #8,SP	ADDW #4,SP	JSRV (tbl4)	▨

Extended code: b'0010'
1st nibble/2nd nibble

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	MOVW An,Am				CMPW An,Am				MOVW SP,Am		MOVW An,SP		BTST #8,Dm			
1	JMP (A0)	JSR (A0)	JMP (A1)	JSR (A1)	MOV PSW,Dm				REP #3							
2	▨	BGT d7	BHI d7	BLS d7	BNC d7	BNS d7	BVC d7	BVS d7	NOT Dn				ROR Dn			
3	▨	BGT d11	BHI d11	BLS d11	BNC d11	BNS d11	BVC d11	BVS d11	ASR Dn				LSR Dn			
4	SUBW DWn,DWm				SUBW #16,DWm		SUBW #16,Am		SUBW DWn,Am				MOVW DWn,Am			
5	ADDW DWn,DWm				ADDW #16,DWm		ADDW #16,Am		ADDW DWn,Am				CMPW DWn,Am			
6	MOV (d16,SP),Dm				MOV (d8,SP),Dm				MOV (d16,An),Dm							
7	MOV Dn,(d16,SP)				MOV Dn,(d8,SP)				MOV Dn,(d16,Am)							
8	MOVW DWn,DWm (NOPL @n=m)				CMPW DWn,DWm				ADDUW Dn,Am							
9	EXT Dn,DWm		AND #8,PSW	OR #8,PSW	MOV Dn,PSW				ADDSW Dn,Am							
A	SUB Dn,Dm / SUB #8,Dm															
B	SUBC Dn,Dm															
C	MOV (abs16),Dm				MOVW (abs16),Am		MOVW (abs16),DWm		CBEQ #8,Dm,d11				MOVW An,DWm			
D	MOV Dn,(abs16)				MOVW An,(abs16)		MOVW DWn,(abs16)		CBNE #8,Dm,d11				CBEQ #8,(abs8),d7/d11		CBNE #8,(abs8),d7/d11	
E	MOVW (d16,SP),Am		MOVW (d16,SP),DWm		MOVW (d8,SP),Am		MOVW (d8,SP),DWm		MOVW (An),Am				ADDW #8,Am		DIVU Dn,DWm	
F	MOVW An,(d16,SP)		MOVW DWn,(d16,SP)		MOVW An,(d8,SP)		MOVW DWn,(d8,SP)		MOVW An,(Am)				ADDW #16,SP	▨	MULU Dn,Dm	

Extended code: b'0011'
2nd nibble/3rd nibble

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	TBZ (abs8)bp,d7								TBZ (abs8)bp,d11							
1	TBNZ (abs8)bp,d7								TBNZ (abs8)bp,d11							
2	CMP Dn,Dm															
3	ADD Dn,Dm															
4	TBZ (io8)bp,d7								TBZ (io8)bp,d11							
5	TBNZ (io8)bp,d7								TBNZ (io8)bp,d11							
6	OR Dn,Dm															
7	AND Dn,Dm															
8	BSET (io8)bp								BCLR (io8)bp							
9	JMP abs18(label)								JSR abs18(label)							
A	XOR Dn,Dm / XOR #8,Dm															
B	ADDC Dn,Dm															
C	BSET (abs16)bp								BCLR (abs16)bp							
D	BTST (abs16)bp								cmp #8,(abs16)	mov #8,(abs16)	▨	▨	CBEQ #8,(abs16),d7/11		CBNE #8,(abs16),d7/11	
E	TBZ (abs16)bp,d7								TBZ (abs16)bp,d11							
F	TBNZ (abs16)bp,d7								TBNZ (abs16)bp,d11							

MN101C00 Series
LSI User's Manual

December, 1996 Version 0.02

Issued by Matsushita Electronics Corporation Micro Computer SE

© Matsushita Electronics Corporation