NEC

Preliminary User's Manual

V30MZ™

16-Bit Microprocessor Core

Hardware

Document No. A13761EJ1V0UM00 (1st edition)
Date Published December 1998 N CP(K)

[MEMO]

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

V30MZ, V30HL, V30MX, and V Series are trademarks of NEC Corporation.

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by the customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or others.

While NEC Corporation has been making continuous effort to enhance the reliability of its semiconductor devices, the possibility of defects cannot be eliminated entirely. To minimize risks of damage or injury to persons or property arising from a defect in an NEC semiconductor device, customers must incorporate sufficient safety measures in its design, such as redundancy, fire-containment, and anti-failure features.

NEC devices are classified into the following three quality grades:
"Standard", "Special", and "Specific". The Specific quality grade applies only to devices developed based on a customer designated "quality assurance program" for a specific application. The recommended applications of a device depend on its quality grade, as indicated below. Customers must check the quality grade of each device before using it in a particular application.

Standard:	Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots
Special:	Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)
Specific:	Aircrafts, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems or medical equipment for life support, etc.

The quality grade of NEC devices is "Standard" unless otherwise specified in NEC's Data Sheets or Data Books.
If customers intend to use NEC devices for applications other than those specified for Standard quality grade, they should contact an NEC sales representative in advance.

Anti-radioactive design is not implemented in this product.

M7 96.5

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Rodovia Presidente Dutra, Km 214
07210-902-Guarulhos-SP Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

J98. 11

[MEMO]

INTRODUCTION

Readers : This manual is intended for users who have an understanding of the V30MZ hardware which is the CPU core of CBIC functions and wish to design an application system using the V30MZ functions.

Purpose : This manual is intended for users to understand the V30MZ hardware functions described in the Organization below.

Organization : This V30MZ User's Manual mainly consists of the following chapters.

- General Description
- Pin functions
- CPU functions
- Bus control functions
- Interrupt functions
- Standby functions
- Reset functions
- Test functions

How to read this manual: This manual assumes that users have a general understanding of electric circuits, logical circuits, and microcontrollers.

To understand the overall functions of the V30MZ functions
→ Read this manual in the order of the **TABLE OF CONTENTS**.

To find the differences between V30HL™ and V30MX™
→ Refer to **Section 1.3 Differences between V30MZ and V30HL, V30MX.**

To find the details of instruction functions
→ Refer to the separate volume of the **16-Bit V Series™ Instruction User's Manual**.

Conventions :

Data significance	:	Higher digits on the left and lower digits on the right
Active low representation	:	xxxB (B after pin or signal name)
Note	:	Footnote for item marked with **Note** in the text
Caution	:	Information requiring particular attention
Remark	:	Supplementary information
Numerical representation	:	Binary … xxxx or xxxxB Decimal … xxxx Hexadecimal … xxxxH

Prefixes indicating power of 2 (address space, memory capacity) :

K (kilo)	: $2^{10} = 1024$
M (mega)	: $2^{20} = 1024^{2}$
G (giga)	: $2^{30} = 1024^{3}$

Related documents : Note that the related documents may be preliminary versions, but there are not indicated as such in this document.

- 16-Bit V Series Instruction User's Manual (U11301E)
- CB-C9 Family VX/VM Type Design Manual User's Manual (A12745E)
- CB-C9 Family VX/VM Type Core Library CPU Core User's Manual (A13195E)

CONTENTS

LIST OF FIGURES

Figure No. Title Page

LIST OF TABLES

CHAPTER 1 GENERAL DESCRIPTION

The V30MZ is a CPU core that is an improved version of the V30MX, which itself enhances the bus efficiency of the μPD70116H (other name: V30HL), an original NEC microprocessor.

The V30MZ raises the bus efficiency by realizing 1 clock/bus cycle. The incorporation of an internal pipeline considerably raises the instruction execution time, enabling fast processing comparable to that of RISC microprocessors.

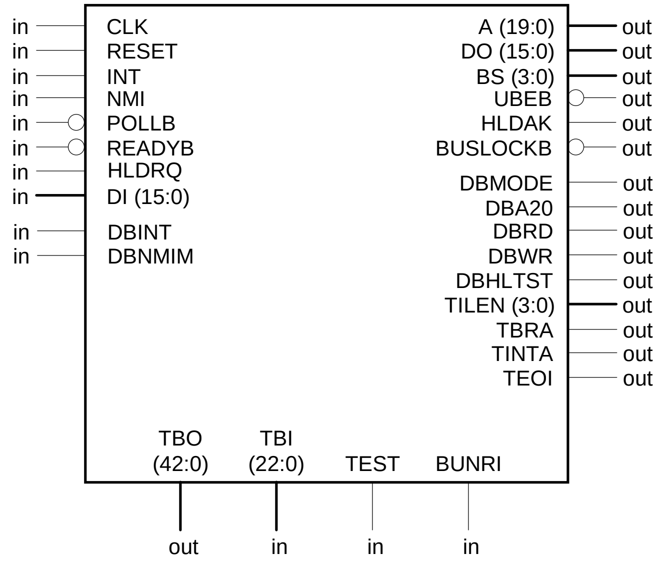
Compared to the V30MX's 4.3 MIPS (33-MHz operation, no wait), the V30MZ realizes a processing performance of 35 MIPS (66-MHz operation, no wait).

1.1 Features

(1) Processing performance: 35 MIPS (66-MHz operation, no wait)
(2) CMOS static design (internal system clock can be fully stopped)
(3) 1 bus cycle: 1 clock
(4) External bus interface
- Address bus: 20 bits
- Data bus: 16 bits (separate input/output buses)

(5) Bus hold function
(6) Standby function (HALT mode)

1.2 Symbol Diagram



1.3 Differences between V30MZ and V30HL, V30MX

(1/3)

Item	V30MZ	V30HL	V30MX
Address/data bus	A19 to A0, DI15 to DI0, DO15 to DO0	A19 to A16, AD15 to AD0	A23 to A0, D15 to D0
Large-scale mode/small-scale mode	Not provided	Provided	Not provided
Pin functions	Following pins of V30HL are removed[Note]: ASTB, PS3 to PS0, $\overline{\text{BUFEN}}$, QS1, QS0, BUF$\overline{\text{R}}$/W, $\overline{\text{RD}}$, IC, $\overline{\text{RQ}}$/$\overline{\text{AK1}}$, $\overline{\text{RQ}}$/AK0, $\overline{\text{INTAK}}$, S/$\overline{\text{LG}}$, LBS0, $\overline{\text{WR}}$, NC		—
μPD8080AF emulation function	Not provided	Provided	Not provided
Connection to numerical operation co-processor	Not possible	Possible	
LIM EMS4.0 function	Not provided	Not provided	Provided
Test function as CBIC core	Provided (TBI22 to TBI0, TBO42 to TBO0, BUNRI, TEST)	Not provided	Provided
BUSLOCKB pin status in case of BUSLOCK instruction execution prior to HALT instruction	High-level output	Low-level output	
Status of UBEB pin during interrupt acknowledge cycle	High-level output	Low-level output	
Status of output pins during bus hold	See section **2.2. Pin Statuses**	High impedance	
Relationship between BUSLOCK instruction and bus hold request	Bus hold request is acknowledged even if BUSLOCK instruction is executed immediately before an instruction that does not perform access to memory or I/O. The BUSLOCKB output remains high.	BUSLOCK instruction effective for all instructions	
Bus hold request acknowledged between first and second bus cycle when odd address word data is accessed	Not possible	Possible	
Bus status output at recovery from bus hold status to standby mode	No (remains in idle status)	Provided	
Instruction execution time	The number of instruction clocks for each instruction and the CPU operating frequency of the V30MZ have been improved, so that the instruction execution time is considerably reduced. Note that programs that depend on the number of instruction execution clocks, such as consecutive I/O accesses, may not function normally.		
Interrupt response time	The V30MZ performs pipeline processing internally, executing multiple instruction in parallel. Therefore, in cases such as when a hardware interrupt synchronized with a given bus cycle is requested, the V30MZ may acknowledge the interrupt request after performing a larger number of instructions than the V30HL and V30MX. However, this does not apply with regard to I/O accesses		
Undefined flag change	If an arithmetic operation defined as an indefinite flag change is executed, the contents of the flag immediately after the execution may differ from the V30HL and V30MX. This is especially likely to occur in the case of multiply and divide instruction.		
Interrupt request acknowledge disable timing	The timing at which interrupt requests are not acknowledged differs. (Refer to **Section 5.3 Timing at which Interrupt is Not Acknowledged**.)		

Note The BS3 pin of the V30MZ has the same functions as the $\overline{\text{IO}}$/M pin of the V30HL, except for the output timing.

Remark Active low pins are indicated with $\overline{\text{xxx}}$ (overscore added) in the case of the V30HL, whereas they are indicated with xxxB (B added) in the case of the V30MZ.

(2/3)

Item	V30MZ	V30HL	V30MX
Supported instructions	The V30MZ does not support the following instructions supported by the V30HL and V30MX. An undefined result is obtained by executing these instructions. ADD4S, BRKEM, CALLN, CLR1[Note 1], CMP4S, EXT, FPO2, INS, NOT1[Note 2], REPC, REPNC, RETEM, ROL4, ROR4, SET1[Note 3], SUB4S, TEST1 Moreover, the FPO1 instruction is handled as an NOP instruction.		
Number of instruction prefixes	Up to 7 instruction prefixes can be used (for all instructions). Even if instruction prefixes are used redundantly, normal processing is performed as long as their total number doesn't exceed 7. If there are more than 7 prefixes for one instruction, the execution result of the instruction (to which prefixes have been attached) is not guaranteed. Furthermore, normal recovery from interrupt processing is not possible.	For repeat string instructions (REP, MOVBK, etc.), 3 types of prefixes MAX. can be used (REP is also counted as 1 type). If there are redundant instruction prefixes, repeat string instructions cannot be performed normally after the end of interrupt processing. In the case of instructions other than string instructions, the number of instruction prefixes is not limited.	
Decimal correction instruction	Performs a correction operation for the second byte of CVTDB and DVTBD instructions.	Decimal correction operation is performed regardless of the value of the second byte of the CVTDB and CVTBD instructions.	
Multiple bit shift and rotate instructions	Only the lower 5 bits of the number of shifts are valid.	All 8 bits of the number of shifts (immediate, or specification by CL register) are valid.	
PREPARE instruction	Only the lower 5 bits of the second operand are valid.	All 8 bits of the second operand are valid.	
POP R instruction	Executes memory read cycle 8 times. However, data corresponding to SP is not used.	Except for SP, 7 memory read cycles are performed.	
Repeat prefixed CMPBK, CMPBKB, and CMPBKW instructions	Memory read is performed in the order IX → IY	Memory read is performed in the order IY → IX.	
CALL memptr32 instruction	Reads new PC, PS values after saving current PC, PS values to the stack.	Current PC and PS values are saved on to the stack after the new PC and PS values are read.	
When number of shifts = 0 for shift, and rotate instructions	Executes also write cycle of memory operand. Z flag, P flag, and S flag change for SHL, SHR, and SARA instructions. These flags are set/cleared depending on the execution result of shift instruction.	If the operand is memory, only the read cycle is performed, and the write cycle of the shift result is not performed. For the SHL, SHR, and SARA instructions, the Z flag, P flag, and S flag do not change. These flags retain the status prior to instruction execution.	

Notes **1.** Excluding CLR1 CY and CLR1 DIR.

2. Excluding NOT1 CY.

3. Excluding SET1 CY and SET1 DIR.

(3/3)

Item	V30MZ	V30HL	V30MX
BUSLOCK instruction	Only valid for instruction performing memory or I/O access, and not valid for other instructions. Moreover, during bus lock period, code fetch bus cycle is not performed.	Effective for all instructions. During execution of the instructions following the BUSLOCK instruction, the $\overline{\text{BUSLOCK}}$ output is low level, and during this period, bus hold requests are not accepted. Moreover, during the bus lock period, a code fetch bus cycle may be performed.	

CHAPTER 2 PIN FUNCTIONS

2.1 Pin List

Pin	Input/Output	Function
A19 to A0	Output	Address signal output
DI15 to DI0	Input	Data signal input
DO15 to DO0	Output	Data signal output
UBEB	Output	Data bus upper byte enable signal output
BS3 to BS0	Output	Bus status signal output
READYB	Input	Wait state generation signal input
BUSLOCKB	Output	Bus lock signal output
POLLB	Input	External system period sense signal input
RESET	Input	System reset signal input
HLDRQ	Input	Bus hold request signal input
HLDAK	Output	Bus hold acknowledge signal output
NMI	Input	Non-maskable interrupt request signal input
INT	Input	Maskable interrupt request signal input
CLK	Input	System clock input
BUNRI	Input	Pin for performing test using test bus
TEST	Input	
TBI22 to TBI0	Input	
TBO42 to TBO0	Output	
DBINT	Input	Reserved for NEC
DBMODE	Output	
DBA20	Output	
DBRD	Output	
DBWR	Output	
DBNMIM	Input	
DBHLTST	Output	
TEOI	Output	
TILEN3 to TILEN0	Output	
TBRA	Output	
TINTA	Output	

2.2 Pin Statuses

The status of each output pin in the different operation modes is listed in the table below.

Pin	Pin Status				
	Normal Mode			Test Mode	
	Bus Hold	Standby (HALT) mode	Reset	Standby test mode	Unit test mode
A19 to A0	H	H	H	Undefined	Undefined
DO15 to DO0	Undefined	Undefined	Undefined	Undefined	Undefined
UBEB	H	H	H	Undefined	Undefined
BS3 to BS0	H	H	H	Undefined	Undefined
BUSLOCKB	H	H	H	Undefined	Undefined
HLDAK	H	L	L	Undefined	Undefined
TBO42 to TBO0	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Operating

Remark		
	H	: High-level output
	L	: Low-level output
	Hi-Z	: High impedance
	Operating	: Outputs valid signal

2.3 Description of Pin Statuses

2.3.1 Normal pins

(1) A19 to A0 (Address)...Output

Bus for outputting 20-bit address.

None of the pins ever go into high impedance.

(2) DI15 to DI0 (Data input)...Input

Dedicated input bus for inputting 16-bit data.

Always input high-level or low-level signal (do not make signal high impedance).

(3) DO15 to DO0 (Data output)...Output

Dedicated output bus for outputting 16-bit data.

None of the pins ever go into high impedance.

(4) UBEB (Upper byte enable)...Output

Outputs low-active signal indicating that higher 8 bits of 16-bit data bus are to be used with memory or I/O access cycle. This pin does not go into high impedance.

The bus cycles for which this signal becomes active are as follows.

- Bus cycle through byte access of odd address
- Bus cycle through first byte access of odd address for word data
- Bus cycle through access of even address for word data

Combined with the A0 signal, the bus cycle can be identified as follows.

Table 2-1. Relationship among Operand and UBEB, A0, and Bus Cycles

Operand		UBEB pin output level	A0 pin output level	Number of bus cycles
Even address word		L	L	1
Odd address word	1st bus cycle	L	H	2
	2nd bus cycle	H	L	
Even address byte		H	L	1
Odd address byte		L	H	1

Remark L: Low level
H: High level

(5) BS3 to BS0 (Bus status)...Output

Outputs status signal to external to notify state of the bus cycle. During reset and bus hold acknowledge, go into idle state (high-level output).

This pin does not go into high-impedance.

The BS3 pin has the same functions as the $\overline{\text{IO}}$/M pin of the V30HL, except for output timing (only names differ).

Table 2-2. Relationship between BS3 to BS0 Signal and Bus Cycle

Pin Output Level				Bus Cycle (Status)
BS3	BS2	BS1	BS0	
L	L	L	L	Interrupt acknowledge
L	H	L	H	I/O read
L	H	H	L	I/O write
H	L	L	L	Standby (HALT) mode
H	L	L	H	Memory data read
H	L	H	L	Memory data write
H	H	L	H	Code fetch
H	H	H	H	Idle status

Remarks 1. L: Low level
H: High level

2. No output with combinations other than above.

(6) READYB (Ready)...Input

Performs wait control.

When memory or I/O data read/write operation cannot be completed within the basic bus cycle (1 clock), the bus cycle can be extended by inputting an inactive level (high level) to this pin.

(7) BUSLOCKB (Bus lock)...Output

It outputs a low-active signal to other bus masters requesting that they do not use the system bus during execution of 1 instruction following the BUSLOCK instruction. It also outputs the signal during interrupt acknowledge.

It does not go into high impedance.

(8) POLLB (Poll)...Input

It is used to synchronize between program execution by the V30MZ and operation of an external device. Input to this pin are checked by the POLL instruction: if a low level is input, the next instruction is processed; if a high level is input, program execution is halted until this pin is driven low.

Input of a low level to this pin should be done for at least 9 clocks.

(9) RESET (Reset)...Input

Inputs a reset signal. Following reset release, the V30MZ starts program execution from memory address FFFF0H (segment value: FFFFH, offset value: 0000H).

(10) HLDRQ (Hold request)...Input

Inputs a signal to the V30MZ to request that the external bus master release the address bus, data bus, and control bus (bus hold).

Inputting a high level to this pin causes the bus hold acknowledge status to be entered upon completion of the currently executing bus cycle, and while the high level is input, the bus hold acknowledge status continues.

Input a high level for at least 3 clock cycles.

(11) HLDAK (Hold acknowledge)...Output

Outputs a signal indicating that the HLDRQ signal has been acknowledged and that the bus hold acknowledge status is entered.

(12) NMI (Non-maskable interrupt)...Input

Inputs a non-maskable interrupt signal by software.

The NMI signal is active at the rising edge and detected in any clock cycle, however, it starts interrupt servicing after the end of the instruction being executed.

The interrupt start address for this interrupt is determined by interrupt vector 2.

Input an active level (high level) for at least 5 cycles after a rising edge.

When inputting NMI requests consecutively, keep NMI low for at least one clock cycle.

The priority order of interrupt request signals is as follows.

INT < NMI < HLDRQ

Remark The standby mode can also be released by an NMI signal.

(13) INT (Interrupt request)...Input

Inputs an interrupt request signal that can be masked by software.

Input an active level (high level) to this until the interrupt acknowledge status is output from the BS3-BS0 pins.

(14) CLK (Clock)...Input

Inputs a clock signal. Input to this CLK pin and internal operation of the V30MZ are performed at the same frequency.

When the CLK input is stopped, the supply current enters 0A.

2.3.2 Test Pins

(1) TBI22 to TBI0 (Test bus input)...Input

Input test bus pin.

(2) TBO42 to TBO0 (Test bus output)...Output

Output test bus pin.

(3) TEST (Test bus control)...Input

Test bus control input pin.

(4) BUNRI (Test bus control)...Input

Input pin for selecting normal mode/test mode.

Remark For details on the functions of each pin, see **CHAPTER 8 TEST FUNCTIONS**.

2.3.3 Reserved pins

The following each pin is reserved for NEC.

According to **Section 2.4 Handling of Unused Pins**, connect each pin.

- DBINT
- DBMODE
- DBA20
- DBRD
- DBWR
- DBNMIM
- DBHLTST
- TEOI
- TILEN3 to TILEN0
- TBRA
- TINTA

2.4 Handling of Unused Pins

Pin	Input/Output	Recommended Handling
A19 to A0	Output	Leave open.
DO15 to DO0	Output	
UBEB	Output	
BS3 to BS0	Output	
READYB	Input	Input low level.
BUSLOCKB	Output	Leave open.
POLLB	Input	Input low level.
HLDRQ	Input	
HLDAK	Output	Leave open.
NMI	Input	Input low level.
INT	Input	
DBINT	Input	
DBMODE	Output	Leave open.
DBA20	Output	
DBRD	Output	
DBWR	Output	
DBNMIM	Input	Input low level.
DBHLTST	Output	Leave open.
TEOI	Output	
TILEN3 to TILEN0	Output	
TBRA	Output	
TINTA	Output	

[MEMO]

CHAPTER 3 CPU FUNCTIONS

3.1 Register Configuration

3.1.1 General-purpose registers (AW, BW, CW, DW)

There are four 16-bit registers. These can be not only used as 16-bit registers, but also accessed as 8-bit registers (AH, AL, BH, BL, CH, CL, DH, DL) by dividing each register into the higher 8 bits and the lower 8 bits.

Therefore, these registers are used as 8-bit registers or 16-bit registers for a wide range of instructions such as transfer instruction, arithmetic operation instruction, logical operation instruction.

Furthermore, the following registers are used as the default registers for specific instruction processing.

- AW : Word multiplication/division, word input/output, data conversion
- AL : Byte multiplication/division, byte input/output, BCD rotate, data conversion
- AH : Byte multiplication/division
- BW : Data conversion (table reference)
- CW : Loop control branch, repeat, and prefix
- CL : Shift instruction, rotation instruction
- DW : Word multiplication/division, indirect addressing input/output

3.1.2 Segment registers (PS, SS, DS0, DS1)

The V30MZ can divide the memory space into logical segments in 64 K-byte units and control up to 4 segments simultaneously (segment system). The start address of each segment is specified by the following 4 segment registers.

- Program segment register (PS) : Specifies the base address of the segment that stores instructions.
- Stack segment register (SS) : Specifies the base address of the segment that performs stack operations.
- Data segment 0 register (DS0) : Specifies the base address of the segment that stores data.
- Data segment 1 register (DS1) : Specifies the base address of the segment that is used as a data destination by data transfer instructions.

For details of the segment system and segment registers, refer to **Section 3.4 Logical Address and Physical Address**.

3.1.3 Pointer (SP, BP)

The pointer consists of two 16-bit registers (stack pointer (SP) and base pointer (BP)).

Each register is used as a pointer to specify a memory address and can be referenced in an instruction and is also used as an index register during a memory data reference.

The SP indicates the address in the stack segment at which the latest data is stored and is used as the default register during stack operation.

The BP is used to fetch the data stored on the stack.

3.1.4 Program counter (PC)

The PC is a 16-bit binary counter that holds the offset information of the memory address of the program that the execution unit (EXU) is about to execute.

The PC value is automatically incremented (+1) every time the microprogram fetches an instruction code from an instruction queue.

Furthermore, in execution of a branch instruction with branch or condition, subroutine control instruction, and interrupt instruction, a new location is loaded and the PC value becomes the same as that of the prefetch pointer (PFP).

3.1.5 Program status word (PSW)

The PSW consists of 6 kinds of status flag and 4 kinds of control flag.

(1) Status flag

- Overflow flag (V)
- Sign flag (S)
- Zero flag (Z)
- Auxiliary carry flag (AC)
- Parity flag (P)
- Carry flag (CY)

(2) Control flag

- Mode flag (MD)
- Direction flag (DIR)
- Interrupt enable flag (IE)
- Break flag (BRK)

The status flag is automatically set (1) and cleared (0) according to the execution result (data value) of each instruction. The CY flag can directly be set/ cleared or inverted by an instruction.

The control flag is set/cleared by an instruction and controls the operation of the V30MZ. The IE flag and BRK flag are cleared (0) when interrupt servicing is started.

RESET input clears (0) all flags (except MD flag).

The PSW is manipulated in byte units or word units by the processing shown below. Processing in byte units is only carried out on the lower 8 bits (including the status flags except the V flag).

Figure 3-1. Program Status Word (PSW)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MD	1	1	1	V	DIR	IE	BRK	S	Z	0	AC	0	P	1	CY

Bits 7 to 0 can be stored or restored in AH by a MOV instruction.

All bits of the PSW are saved to the stack when an interrupt is generated or in execution of a call instruction (CALL) and restored by a return instruction (RET, RETI).

The PSW can be saved or restored to the stack independently by a PUSH PSW instruction or POP PSW instruction.

The flags are set to the states shown below after execution of each instruction.

(a) Carry flag (CY)

<1> Binary addition/subtraction

In the case of processing in byte units, CY is set when there is a carry or borrow from operation result bit 7, and cleared otherwise.

In the case of word operation, CY is set when there is a carry or borrow from operation result bit 15, and cleared otherwise.

It is not changed by an increment or decrement instruction.

<2> Logical operation

CY is cleared without regard to the operation result.

<3> Binary multiplication

CY is cleared if AH is other than 0 as a result of an unsigned byte operation.

CY is cleared if AH is AL sign extension as a result of a signed byte operation and set otherwise.

CY is cleared if DW is 0 as a result of an unsigned word operation and set otherwise.

CY is cleared if DW is AW sign extension as a result of an unsigned word operation and set otherwise.

In the case of an 8-bit immediate operation, CY is cleared when the product is within 16 bits and set otherwise.

<4> Binary division

Undefined.

<5> Shift/rotate

In the case of a shift or rotate including the CY flag, CY is set when the bit shifted to the CY flag is 1 and cleared if 0.

(b) Parity flag (P)

<1> Binary addition/subtraction, logical operation, shift

Set when the number of “1” bits of the lower 8 bits of the operation result is even and cleared when it is odd.

Set when the result is all 0.

<2> Binary multiplication/subtraction

Undefined.

(c) Auxiliary carry flag (AC)

<1> Binary addition/subtraction

In the case of processing in byte units, it is set when there is a carry from the lower 4 bits to the higher 4 bits or a borrow from the higher 4 bits to the lower 4 bits, and cleared otherwise.

In a word operation, it performs the same operation as for a byte operation with respect to the lower bytes.

<2> Logical operation, binary multiplication/division, shift/rotate

Undefined.

(d) Zero flag (Z)

<1> Binary addition/subtraction, logical operation, shift/rotate

It is set when the 8 bits and 16 bits of the result are all 0 for a byte operation and word operation, respectively, and cleared otherwise.

<2> Binary multiplication/division

Undefined.

(e) Sign flag (S)

<1> Binary addition/subtraction, logical operation, shift/rotate

Set when bit 7 of the result is 1 and cleared when it is 0 in the case of a byte operation.

Set when bit 15 of the result is 1 and cleared when it is 0 in the case of a word operation.

<2> Binary multiplication/division

Undefined.

(f) Overflow flag (V)

<1> Binary addition/subtraction

Set when carries from bit 7 and bit 6 are different and cleared when they are the same in the case of a byte operation.

Set when carries from bit 15 and bit 14 are different and cleared when they are the same in the case of a word operation.

<2> Binary multiplication

As a result of an unsigned byte operation, cleared if AH is 0 and set otherwise.

As a result of a signed byte operation, cleared if AH is sign extension of AL and set otherwise.

As a result of an unsigned word operation, cleared if DW is 0 and set otherwise.

As a result of a signed word operation, cleared if DW is sign extension of AW and set otherwise.

In the case of an 8-bit immediate operation, cleared if the product is within 16 bits and set if the product exceeds 16 bits.

<3> Binary division

Cleared.

<4> Logical operation

Cleared.

< 5 > Shift/rotate

In the case of a left 1-bit shift/rotate, the status of the overflow flag is as follows depending on the operation result.

- When CY = MSB: Cleared
- When CY ≠ MSB: Set

In the case of a right 1-bit shift/rotate, its status is as follows depending on the operation result.

- When MSB = next lower bit of MSB: Cleared
- When MSB ≠ next lower bit of MSB: Set

In the case of a multi-bit shift/rotate, it is undefined.

(g) Break flag (BRK)

Only when it is saved to the stack as part of the PSW, it can be set by a memory manipulation instruction, and becomes valid when restored to the PSW after it is set.

If the BRK flag is set, executing one instruction automatically generates a software interrupt (interrupt vector 1) allowing tracing of one instruction at a time.

(h) Interrupt enable flag (IE)

IE is set by an EI instruction and the maskable interrupt (INT) is enabled. It is cleared by a DI instruction and the maskable interrupt (INT) is disabled.

(i) Direction flag (DIR)

When the DIR flag is set, processing is carried out from the higher addresses to the lower addresses in block transfer and/or I/O system instructions. When it is cleared, processing is carried out from the lower addresses to the higher addresses.

(j) Mode flag (MD)

This is a μPD8080AF emulation function related flag that conforms to the previous V30HL. Since the V30MZ is not provided with the emulation function, this flag is invalid.

3.1.6 Index register (IX, IY)

This consists of two 16-bit registers (IX, IY). In a memory data reference, it is used as an index register to generate effective addresses (each register can also be referenced in an instruction).

Furthermore, in specific instruction processing, it has the following special roles.

IX: Address register for source operand in block data manipulation instruction
Address register for source operand in BCD string operation instruction

IY: Address register for destination operand in block data manipulation
Address register for destination operand in BCD string operation instruction

3.2 Address Space

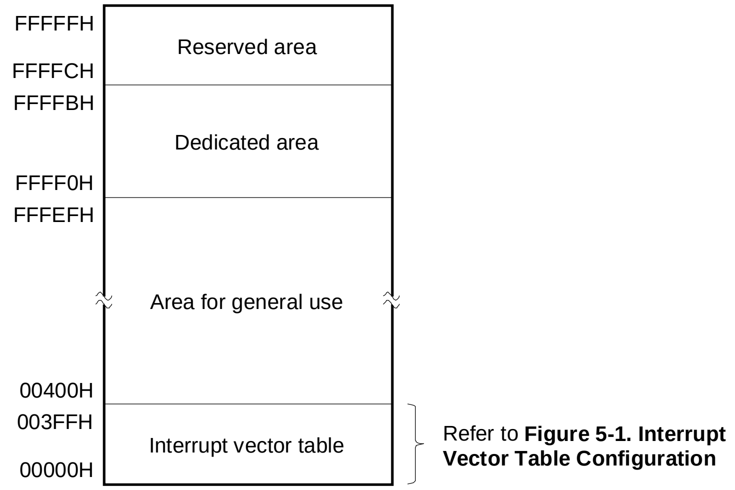
3.2.1 Memory space

The V30MZ uses 20-bit address information and can access 1 M bytes (512 K words) of memory.

Figure 3-2 shows the memory map. The 1 K byte from 00000H to 003FFH is allocated to the interrupt vector table. However, the table area that is not used by the system can be used for other purposes.

The start address after a reset is FFFF0H. The 12 bytes from FFFF0H to FFFFBH are automatically used for a reset start, etc., and cannot be used for other purposes. The 4 bytes from FFFFCH to FFFFFH are also reserved for future use and are not available for users.

Figure 3-2. Memory Map



The elements stored in the memory area include operation codes, interrupt start addresses, stack data, general variables, and consist of two kinds; byte units and word units.

Addresses generated by an instruction for these elements can be even (A0 = 0) or odd (A0 = 1). Word data in the V30MZ is designed to be accessible for both even and odd addresses. Both even and odd addresses are possible for generation of an instruction. For the access method, refer to **Section 4.1 Interface between V30MZ and Memory**.

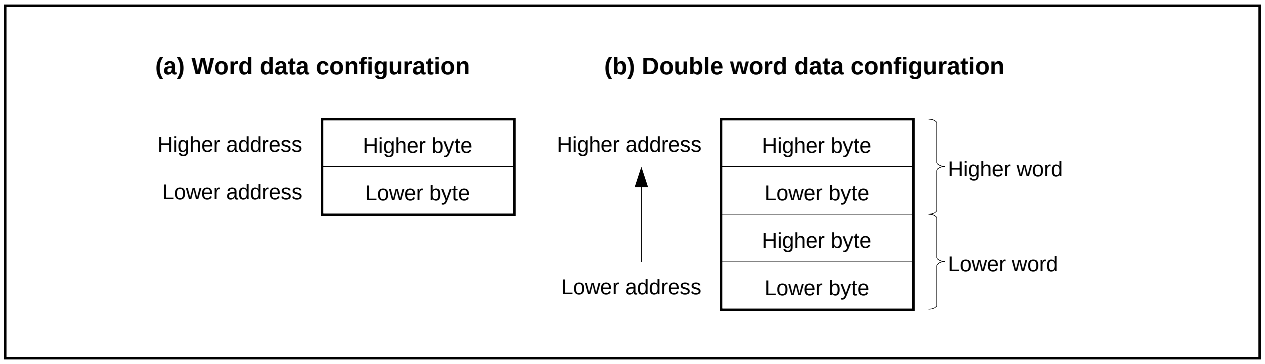
Table 3-1 shows the address and data configuration of each memory element.

Table 3-1. Address and Data Configuration of Each Memory Element

Memory Element	Address	Data Configuration
Operation code	Even/odd	1 to 6 bytes
Interrupt vector table	Even	2 words/vector
Stack	Even/odd	Word
General variable	Even/odd	Byte/word/double word

The word data configuration and double word data configuration are as follows.

Figure 3-3. Configuration of Word Data and Double Word Data



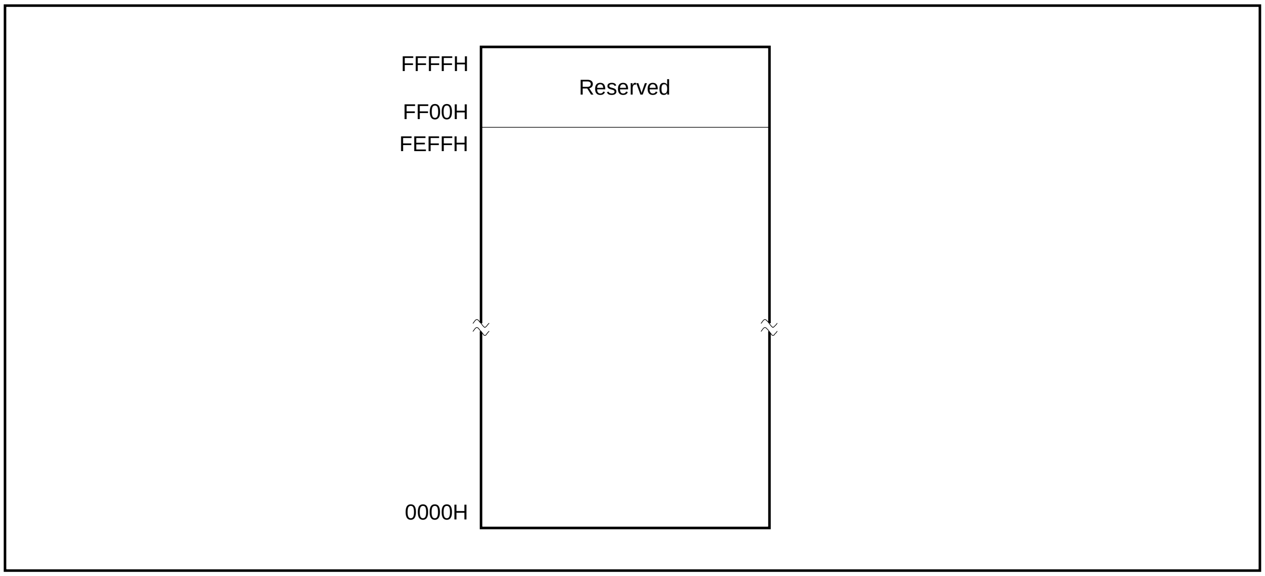
3.2.2 I/O space

The V30MZ can access an I/O space of up to 64 K bytes (32 K words) in an area independent of the memory space.

The I/O space is addressed by I/O address information output from the lower 16 bits of the address bus. Figure 3-4 shows the I/O map. The 256 bytes of FF00H to FFFFH are reserved for future use and are not available for users.

For the access method, refer to **Section 4.2 Interface between V30MZ and I/O**.

Figure 3-4. I/O Map



3.3 Instruction Prefetch

The V30MZ performs pipeline processing internally, performing instruction fetch (prefetch), instruction decode, and instruction execution in parallel. For this reason, it is difficult to determine what part of the program is currently being executed by monitoring the output of the address bus for the instruction code fetch.

If there are conditional branch instructions, even in case branching does not occur, the address of the branch destination is prefetched (only one time), so that further monitoring of the program is difficult.

The V30MZ has 8 prefetch queues (16 bytes).

3.4 Logical Address and Physical Address

There are two kinds of memory space address; logical address and physical address.

The physical address means an address that directly corresponds to hardware. The V30MZ can access a 1 M-byte memory space and so the range of a physical address value is 00000H to FFFFFH. A physical address is generated every time the bus control unit (BCU) is started which fetches an instruction and transfers data, etc.

The logical address means an address used for addressing in the segment system.

3.4.1 Segment system

The segment means an address space in small units (MAX. 64 K bytes) which do not directly depend on program creation.

Each segment consists of continuous memory and can be specified individually.

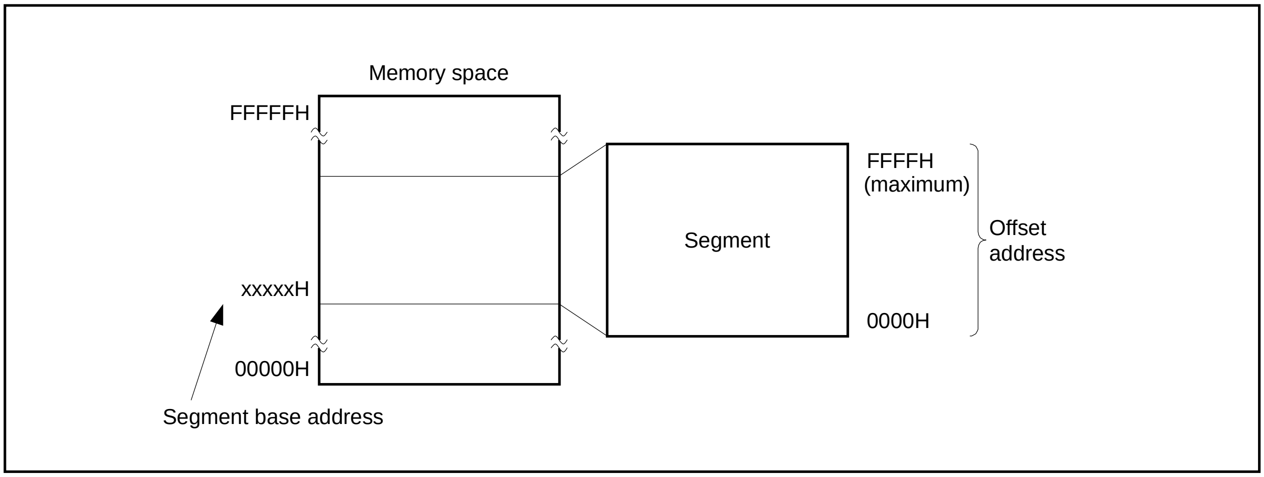
Physical addresses cannot be controlled directly in program creation in machine language. The V30MZ specifies memory addresses in a segment system.

Addressing in the segment system uses the following two types of address.

• Segment base address	:	Start address of segment (address in 1 M-byte memory space)
• Offset address	:	Address allocated to each segment

In the segment system, the segment base address is fixed as a reference point and only the offset address is treated as an address in processing within each segment.

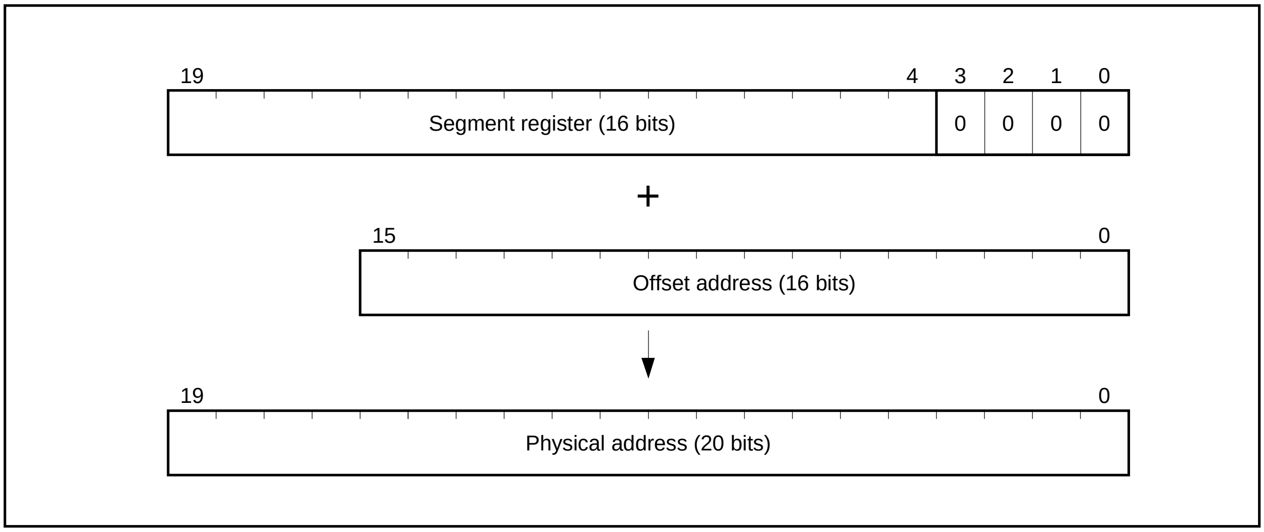
Figure 3-5. Conceptual Diagram of Segment System



The segment base address is specified by the segment register.

The physical address is a sum of the segment base address and offset address. Figure 3-6 shows the relationship between the segment register and offset address, and physical address.

Figure 3-6. Relationship between Segment Register, Offset Address and Physical Address



As shown in Figure 3-6, the physical address is a sum of 16 times the segment register content (4 bits shifted to left) and offset value. At this time, the segment register content and offset value are treated as unsigned data.

In a program which is created as a set of multiple segments for which allocation addresses are specified by physical addresses, each segment is compiled and assembled individually and becomes one or a number of object modules. Each object module has a segment name, size, content classification, control information, etc., and becomes a parameter in execution of link processing.

Multiple object modules are linked and the segment base addresses corresponding the physical addresses are specified and become ready to be loaded to actual memory.

3.4.2 Segment configuration

The V30MZ can distinguish 4 kinds of segment (program, stack, data 0, data 1) and define them. For each segment the start address is specified by one of the following 4 segment registers.

The BCU uses different segment registers for generation of physical addresses depending on the type of memory bus cycle.

- Program segment register (PS)
- Stack segment register (SS)
- Data segment 0 register (DS0)
- Data segment 1 register (DS1)

The offset address within each segment is specified by a specific register or effective address. Table 3-2 shows correspondence between each segment register and offset addressing.

Table 3-2. Segment Registers and Offset Addressing

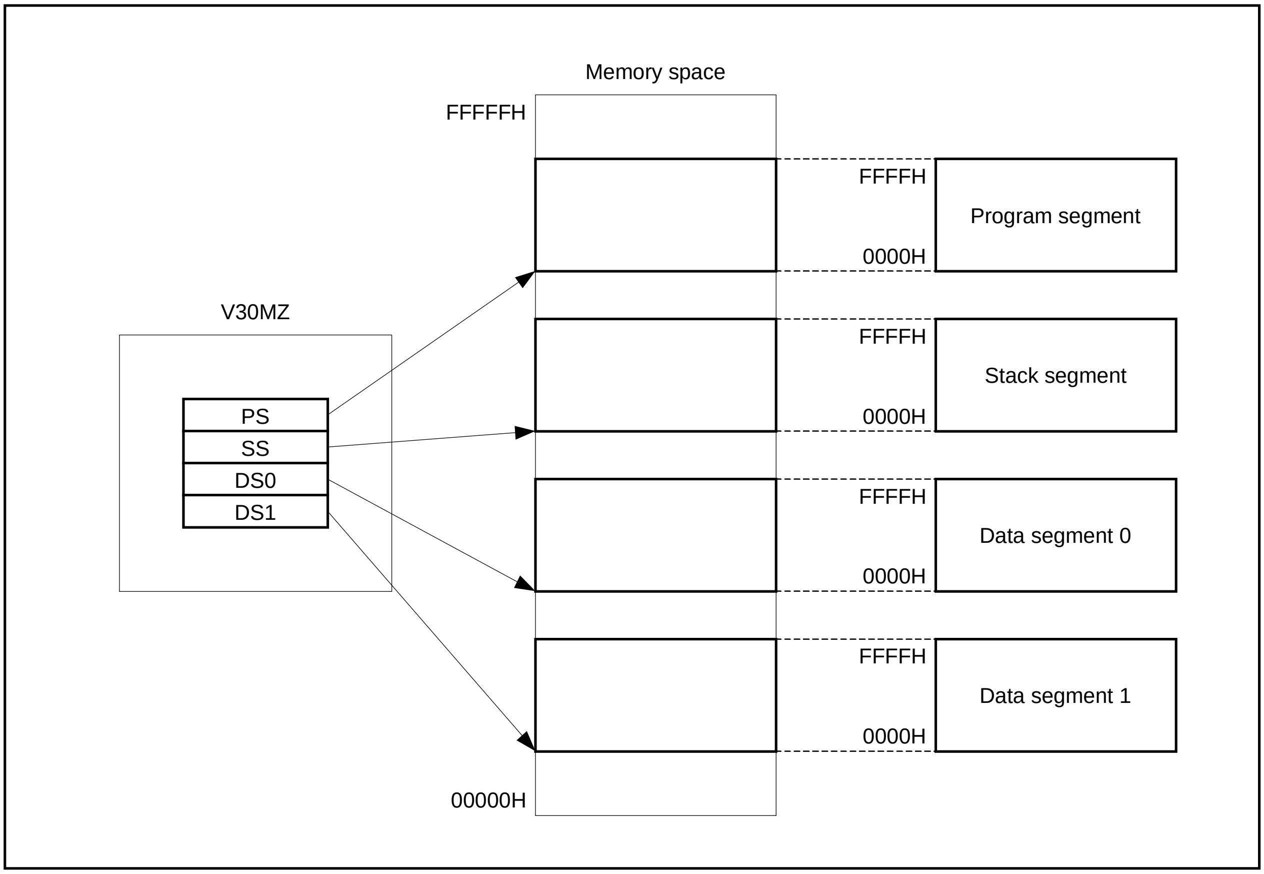
Offset \ Segment Register	Default	Override
PFP	PS	Disabled
SP	SS	Disabled
Effective address (BP base)		PS, DS0, DS1
Effective address (non-BP base)	DS0	PS, SS, DS1
IX in instruction group A (Primitive block transfer instruction, primitive output instruction, BCD string instruction)		
IY in instruction group B (Primitive block transfer instruction, primitive input instruction, BCD string instruction)	DS1	Disabled

When the default offset is a prefetch pointer (PFP), stack pointer (SP) and index register (IY) in instruction group B, the segment registers that can be combined are fixed at PS, SS, and DS1 respectively, and other segment registers cannot be used.

For other default offsets, any segment registers other than the default segment register can be specified by the segment override prefix.

Figure 3-7 shows the relationship between each segment register, segment and memory space.

Figure 3-7. Relationship between Each Segment Register, Segment and Memory Space



Each segment has the following meaning.

(1) Program segment

The start address of this segment is determined by the program segment register (PS) and the offset from the start address is specified by the prefetch pointer (PFP).

In this segment, an operation code, table data, etc., are placed.

By using the segment override prefix (PS:), the program segment can be used as the general variable area and source data area in execution of instruction group A.

(2) Stack segment

The start address of this segment is determined by the stack segment register (SS) and the offset from the start address is specified by the effective address when the stack pointer (SP) and base pointer (BP) as the base address are used.

This is used as an area to save the contents of the return address (PS, PC content), program status word (PSW), general register, etc., as a parameter transfer area and local variable area.

By using the segment override prefix (SS:), the stack segment can be used as a general variable area and source data area in execution of instruction group A.

(3) Data segment 0

The start address of this segment is determined by the data segment 0 register (DS0) and the offset from the start address is specified by the effective address when BP is not used as a base address.

This segment is used as an area to store general variables.

When executing instruction group A, it is used as a source data area. However, in this case, the content of the index register (IX) becomes the offset.

For the effective address when BP is used as the base address, the stack segment is used as the default, but data segment 0 can be used if the segment override prefix (DS0:) is used.

(4) Data segment 1

The start address of this segment is determined by the data segment 1 register (DS1). This can be used as a destination data area when executing instruction group B. In this case, the content of the index register (IY) becomes the offset.

If the segment override prefix (DS1:) is used, data segment 1 can be used as a general variable area or source data area in execution of instruction group A.

3.4.3 Dynamic relocation

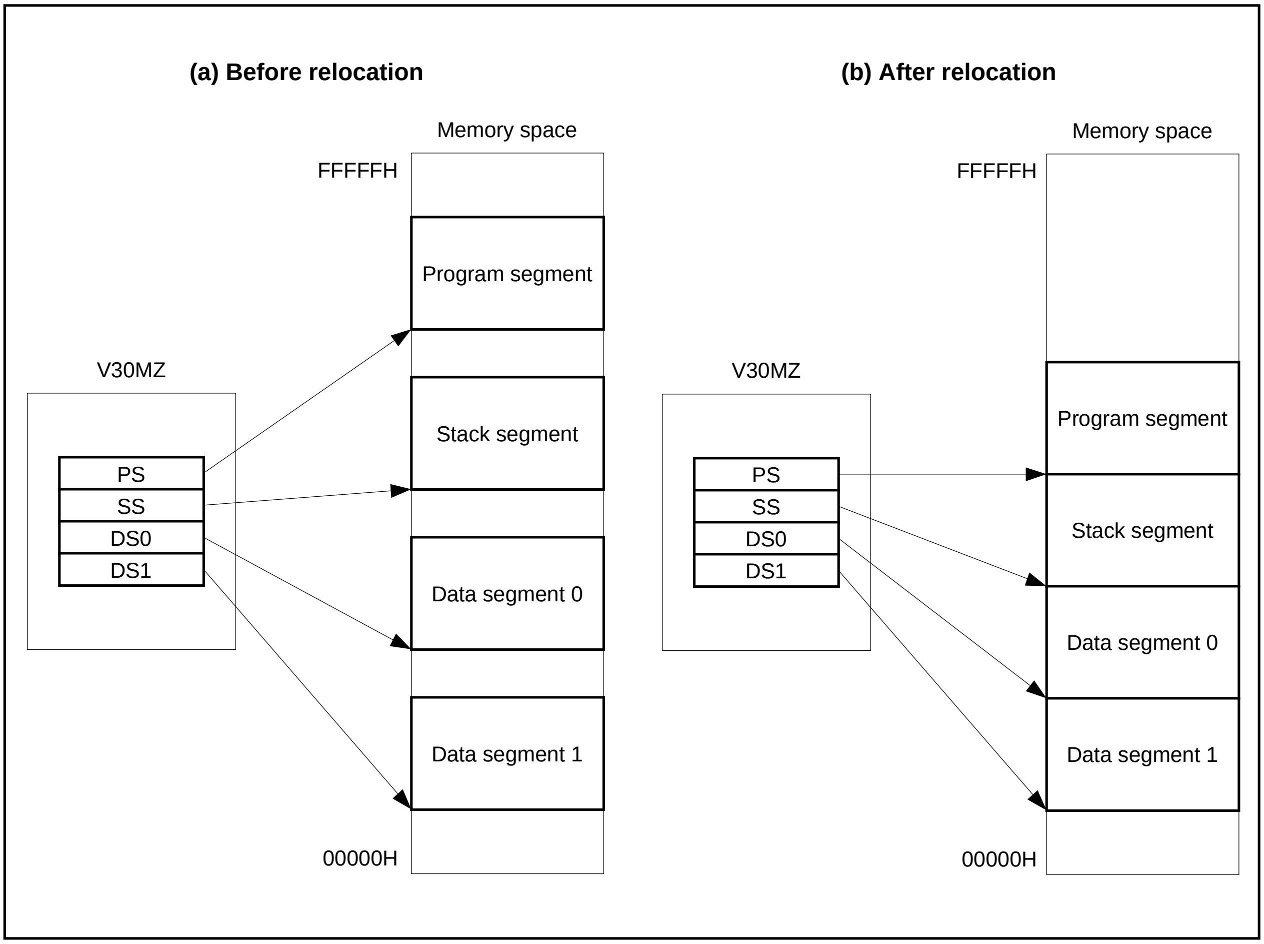
Relocating programs that are stored in two or more files separately in empty memory spaces for each execution is called dynamic relocation.

Figure 3-8 shows a conceptual diagram of dynamic relocation.

For the V30MZ, memory addressing of a program can be determined only with the offset value for the base address of each segment (specified by each segment register). Therefore, it is possible to allocate the program in an arbitrary memory space by only adjusting to the physical address of the memory at which it is to be allocated (however, this is only possible if the base address of each segment is not changed in the program). This increases the degree of freedom of program allocation in the memory (addressing is possible in 16-byte units), enabling more effective utilization of memory and making it easier to implement a system that executes multiple jobs and tasks.

This can be applied to executing a program in a file on an external storage medium such as a floppy disk and hard disk with the OS controlling the memory allocation area, type, and segment registers, and loading the program in any empty memory area.

Figure 3-8. Dynamic Relocation



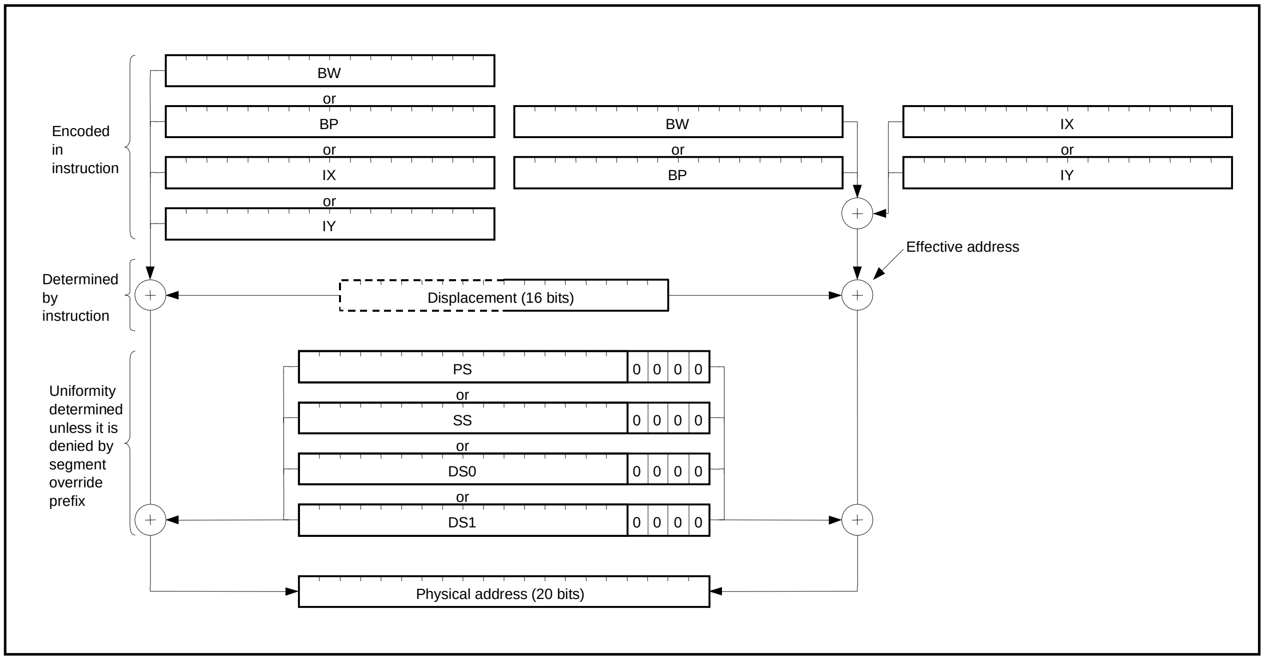
3.5 Effective Address

The effective address (EA) is an unsigned 16-bit number and is the memory address to be processed by an instruction represented by the offset value for the base address of the corresponding segment. This is calculated by the execution unit (EXU) according to the specification of an instruction operand.

The EXU calculates EA in several different methods (addressing mode). The method is selected by the 2nd byte operand of the instruction. The information encoded in the 2nd byte of the instruction indicates how the effective address of the memory indicated by the operand is calculated by the EXU. This operand code is automatically generated by a compiler or assembler from a program statement or instruction description. All addressing modes are available in assembly language (Refer to **Section 3.7 Addressing Mode**).

The method of calculation of EA is shown below. Figure 3-9 indicates that the EXU calculates EA by adding the displacement, base register contents, and index register contents. For any instruction, these three elements can be combined arbitrarily. The displacement is an 8-bit or 16-bit immediate number indicated by an operand.

Figure 3-9. Memory Address Calculation



3.6 Instruction Set

3.6.1 List of instruction sets by function

The V30MZ instruction sets by function are generally classified as follows:

Table 3-3. List of Instruction Sets by Function

Instruction Group	Mnemonic
Data transfer instruction	LDEA, MOV, TRANS, TRANSB, XCH
Repeat prefix	REP, REPE, REPNE, REPNZ, REPZ
Primitive block transfer instruction	CMPBK, CMPBKB, CMPBKW, CMPM, CMPMB, CMPMW, LDM, LDMB, LDMW, MOVBK, MOVBKB, MOVBKW, STM, STMB, STMW
Input/output instruction	IN, OUT
Primitive input/output instruction	INM, OUTM
Addition/subtraction instruction	ADD, ADDC, SUB, SUBC
Increment/decrement instruction	DEC, INC
Multiplication instruction	MUL, MULU
Division instruction	DIV, DIVU
BCD adjustment instruction	ADJ4A, ADJ4S, ADJBA, ADJBS
Data conversion instruction	CVTBD, CVTBW, CVTDB, CVTWL
Comparison instruction	CMP
Complement operation instruction	NEG, NOT
Logical operation instruction	AND, OR, TEST, XOR
Bit manipulation instruction	CLR1 CY,CLR1 DIR, SET1 CY, SET1 DIR, NOT1 CY
Shift instruction	SHL, SHR, SHRA
Rotate instruction	ROL, ROLC, ROR, RORC
Subroutine control instruction	CALL, RET
Stack manipulation instruction	DISPOSE, POP, PREPARE, PUSH
Branch instruction	BR
Conditional branch instruction	BC, BCWZ, BE, BGE, BGT, BH, BL, BLE, BLT, BN, BNC, BNE, BNH, BNL, BNV, BNZ, BP, BPE, BPO, BZ, BV, DBNZ, DBNZE, DBNZNE
Interrupt instruction	BRK, BRKV, CHKIND, RETI
CPU control instruction	BUSLOCK, DI, EI, FPO1[Note], HALT, NOP, POLL
Segment override prefix	DS0:, DS1:, PS:, SS:

Note Treated as a NOP instruction.

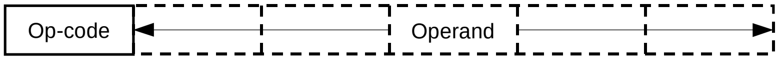
Remarks 1. The following instructions are not supported among the instructions that V30HL supports. Executing these instructions causes undefined.
ADD4S, BRKEM, CALLN, CLR1 (except CLR1 CY, CLR1 DIR), CMP4S, EXT, FPO2, INS, NOT1 (except NOT1 CY), REPC, REPNC, RETEM, ROL4, ROR4, SET1 (except SET1 CY, SET1 DIR), SUB4S, TEST1

2. For details of each instruction, refer to the **16-Bit V Series Instruction User’s Manual**.

3.6.2 Format of object code

Object codes are basically indicated by the following format.

Figure 3-10. Object Code Format



Remark Op-code : 8-bit code indicating type of instruction

Operand : Field indicating register, memory address to be processed by instruction. Indicated by field of 0 to 5 bytes

3.7 Addressing Mode

3.7.1 Instruction address

The instruction address refers to the address at which an operation code is read and, normally it is automatically incremented every time an operation code is read. However, in an instruction that controls the instruction execution sequence such as a jump instruction, subroutine call instruction, the branch destination instruction address is specified by an operand.

(1) Direct addressing

The 4-byte data in an operation code becomes an instruction address and is loaded into the PS and PC registers. This mode is used by the following instructions.

```
CALL  far_proc
BR    far_label
```

(2) PC relative addressing

The 1-byte or 2-byte data in an operation code becomes a displacement from the start address (PC value) of the next instruction and is added to the PC.

This mode is used by the following instructions.

```
CALL       near_proc
BR         near_label
BR         short_label
Bcondition short_label  ; Example  BZ   short_label
                                   BNC short_label
```

(3) Register indirect addressing

The content of any 16-bit register specified by the register specification field in an operation code becomes the instruction address and is loaded into the PC.

This mode is used by the following instructions.

```
CALL  regptr16   ; Example  CALL AW
BR    regptr16   ; Example  BR   IX
```

(4) Memory indirect addressing

The 2-byte or 4-byte data in memory specified by the memory addressing (refer to **Section 3.7.2 Data address**) indicated by the addressing mode specification field in an operation code becomes the instruction address and is directly loaded into the PC or both PS and PC.

This mode is used by the following instructions.

```
CALL  memptr16   ; Example  CALL  word_var [BW]
CALL  memptr32   ; Example  CALL  dword_var [BW+IX]
BR    memptr16   ; Example  BR    word_var [BR+2]
BR    memptr32   ; Example  BR    dword_var [BP+IY]
```

3.7.2 Data address

The data address is an address for reading/writing the operand data of each instruction. Normally, an address is a concept used for memory or I/O, but this operand address includes data in registers, immediate data and I/O data.

(1) Non-memory addressing

Non-memory addressing specifies data in registers, immediate data and I/O data.

(a) Register addressing

Specifies the register from/to which the register specification field in an operation code reads/writes the operand data.

The register addressing is shown in the following description.

General Description	Register that can be Described
reg, reg'	AW, BW, CW, DW, SP, BP, IX, IY, AL, AH, BL, BH, CL, CH, DL, DH
reg8, reg8'	AL, AH, BL, BH, CL, CH, DL, DH
reg16, reg16'	AW, BW, CW, DW, SP, BP, IX, IY
sreg	PS, SS, DS0, DS1
acc	AW, AL

Example of usage:

```
reg16  : MOV AW, IX      ; AW ← IX
reg8   : ADD AL, CH      ; AL ← AL + CH
```

(b) Immediate addressing

1-byte or 2-byte data in an operation code becomes read-only operand data. Immediate addressing cannot be used for the destination operand of an instruction.

Immediate addressing is shown in the following description.

General Description	Value that can be Described
imm8	0 to FFH (0 to 255 or –128 to +127)
imm16	0 to FFFFH (0 to 65535 or –32768 to +32767)
imm	0 to FFFFH (0 to 65535 or –32768 to +32767)
pop_value	0 to FFFFH (0 to 65535) … normally even

Example of usage:

```
imm16      : MOV AW, 216    ; AW ← 216
imm8       : SHL AL, 5      ; Shifts AL to left by 5 bits.
pop_value  : RET 16         ; Deletes unnecessary 16 bytes on stack.
```

(c) I/O addressing

I/O addressing specifies data in a 64-K byte I/O space.

There are two kinds of specification method in I/O addressing as shown below, and these are used by an input/ output instruction.

<1> imm8

8-bit data in an operation code specifies the I/O address.

In this method, specification is limited to a 256-byte space on the lower side of the 64-K byte I/O space.

This specification method is used by the following two instructions.

```
IN     acc, imm8
OUT    imm8, acc
```

<2> DW

The content of DW indicates the I/O address.

This method can be used to specify across the entire 64-K byte I/O space. This specification method is used by the following four instructions.

```
IN        acc, DW
INM       dst_block, DW
OUT       DW, acc
OUTM      DW, src_block
```

(2) Memory addressing

Memory addressing specifies the operand data in memory.

This memory addressing is further divided into several modes by the 5-bit memory addressing specification field placed after an op-code. In all memory addressing modes, a 16-bit offset address from the segment base specified by the default or segment override is specified. Memory addressing is shown in the following description.

Description	Data Length
dmem[Note]	8/16-bit data
mem	8/16-bit data
mem8	8-bit data
mem16	16-bit data

Note Description in an instruction that has no memory addressing specification field

(a) Direct addressing

Indicates the memory address at which 2-byte data in an operation code is the read/write target of operand data.

Example of usage:

```
MOV  byte_var, 216   ; bytemem (offset (byte_var)) ← 216
```

(b) Register indirect addressing

Indicates the memory address at which the 16-bit register (BW or IX or IY) specified by the memory addressing specification field in an operation code is the read/write target of operand data.

Example of usage:

```
MOV  word ptr [BW], 10  ; wordmem (BW) ← 10
ADD  AL, byte ptr [IX]  ; AL ← AL + bytemem (IX)
```

(c) Based addressing

Indicates the memory address at which the value of the 16-bit base register (BW or BP) specified by the memory addressing specification field in an operation code added to a sign extended displacement value indicated by 1-byte or 2-byte data in an operation code is the read/write target of operand data.

When BP is selected as the base register, the default segment register becomes SS, and it can be used when the data pushed to the stack as an argument in procedure calling is accessed from the procedure.

Example of usage:

```
MOV  word_var [BW+2], AW    ; wordmem (offset (word_var)+BW+2) ← AW
SUB  AW, [BP+6]             ; AW ← AW – wordmem (BP+6)
```

(d) Indexed addressing

Indicates the memory address at which the value of the 16-bit indexed register (IX or IY) specified by the memory addressing specification field in an operation code added to a sign extended displacement value indicated by 1-byte or 2-byte data in an operation code is the read/write target of operand data.

Example of usage:

```
MOV  word_var [IY+2], 0    ; wordmem (offset (word_var)+IY+2) ← 0
SUB  AW, [IX+6]            ; AW ← AW – wordmem (IX+6)
```

(e) Addressing with based index

Indicates the memory address at which the value of the 16-bit base register (BW or BP) specified by the memory addressing specification field in an operation code added to a sign extended displacement indicated by 1-byte or 2-byte data in an operation code plus the value of the 16-bit index register (IX or IY) is the read/write target of operand data. That is, it performs addressing similar to a combination of based addressing and indexed addressing.

This addressing can be used to access data that has a 2-dimensional array structure, etc.

Example of usage:

```
MOV  word_var [BW+6]  [IY+2],0    ; wordmem (offset (word_var)+BW+6+IY+2) ← 0
SUB  AW, [BP+6+IX]                ; AW ← AW – wordmem (BP+IX+6)
```

[MEMO]

The V30MZ executes 1 bus cycle in 1 clock.

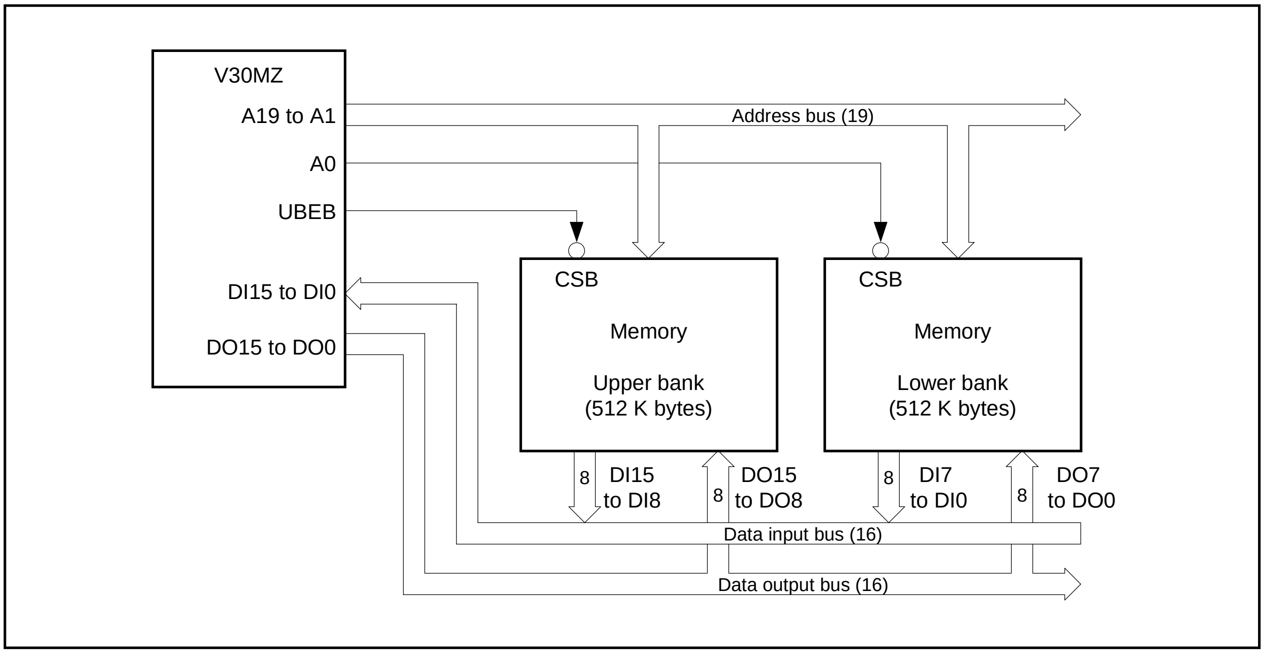
Since accessing memory integrated on the same chip is possible in just 1 clock, it is possible to configure systems that take advantage of both small-capacity, high-speed on-chip memory and large-capacity, low-cost external memory by combining a wait insertion function.

4.1 Interface between V30MZ and Memory

As the V30MZ uses a 16-bit data bus, it is capable of transferring 16-bit word data in 1 bus cycle. However, this applies only when an address generated by an instruction is even (A0 = 0), and if it is odd (A0 = 1) a word data transfer requires 2 bus cycles.

Figure 4-1 shows the interface between the V30MZ and memory.

Figure 4-1. Interface between V30MZ and Memory



In Figure 4-1, A0, when active low, enables the lower bank byte data of memory. Furthermore, aside from the information from the address bus, the UBEB signal is output and when active low this also enables the byte data of the memory higher bank.

(1) When accessing word data at odd address

In the first bus cycle, UBEB = 0 and A0 = 1, and only the higher byte is accessed and then UBEB = 1 is automatically set, the lower 16 bits (A15 to A0) of the address information is incremented (+1). That is, A0 = 0 is set, and the lower byte at the next address is accessed.

(2) When accessing word data at even address

Word data is accessed in 1-bus cycle with UBEB = 0 and A0 = 0.

Table 4-1 shows the relationship between the type of operand and the number of UBEB, A0 pins, and bus cycles.

Table 4-1. V30MZ Data Access

Operand		UBEB pin output level	A0 pin output level	Number of Bus Cycles
Word at even address		L	L	1
Word at odd address	1st bus cycle	L	H	2
	2nd bus cycle	H	L	
Byte at even address		H	L	1
Byte at odd address		L	H	1

Remark L : low level
H : high level

Normally, the V30MZ performs an access (prefetch) of an operation code in word units. However, when a branch to an odd address takes place, only 1 byte at that odd address is fetched and subsequent bytes are fetched again in word units again.

When a vector table address is generated from the vector number (0 to 225), an even address is always generated, and so an access to the interrupt vector table is always performed as word data at an even address. Therefore, a vector table access to one interrupt is always performed in 2 bus cycles for the 2 words of the segment base and offset.

4.1.1 Cautions on accessing word data

When accessing word data by the V30MZ, ensure that all the data that can be checked by the program may be placed at an even address. When it is placed at an odd address, the result will be as follows.

One bus cycle for a memory access requires 1 clock. Therefore, every time word data at an odd address is accessed, one extra clock of the instruction execution time are required compared to accessing word data at an even address. This applies when executing an instruction that has more than one word data access.

In the case of a word data transfer from memory to memory, 2 memory accesses are required for a read from the source and a write to the destination and so the execution time becomes the maximum when both are odd addresses.

This problem of odd addresses also happens in stack manipulation. Registers, etc. are automatically saved to the stack by interrupt servicing, but these are all word data and so when processed at an odd address, note that the number of bus cycles is doubled and the interrupt response time is delayed.

Example: Number of execution clocks of MOV reg, mem instruction
Byte data : 1
Word data : 2 (For odd address)
: 1 (For even address)
This is an example in which one word data access is performed.

4.2 Interface between V30MZ and I/O

The segment system is not applied to an I/O address like memory.

In I/O address output timing, 0 is output to all the higher 4 bits (A19 to A16) of the address bus.

Data can be transferred between the V30MZ and I/O in either byte units or word units and both an 8-bit I/O device and 16-bit I/O device can be connected. However, like memory for a word data access, 1 bus cycle for an even address and 2 bus cycles for an odd address are used.

When accessing an 8-bit I/O device, A0 of the I/O address information is only used for device selection and values higher than A1 are used for device selection and selection of several registers within one device. That is, all the internal registers of the I/O device at an even address are also even and all the internal registers of the I/O device at an odd address are selected with an odd number.

Use of a memory mapped I/O configuration (using the memory area by allocating it for I/O) allows the I/O to be placed in a 1 M-byte memory space not in the I/O space.

Using the memory mapped I/O configuration, it is possible to perform a variety of addressing modes and operation processing for the memory directly to the I/O device.

Caution However, with the memory mapped I/O, all control signals output from the V30MZ are for the memory and so the I/O device is distinguished only by address information. Therefore, special care is required to avoid contention between the addresses of variables and static data, etc., and the addresses allocated to the I/O.

4.3 Read/Write Timing of Memory and I/O

The V30MZ executes one bus cycle in at least 1 clock.

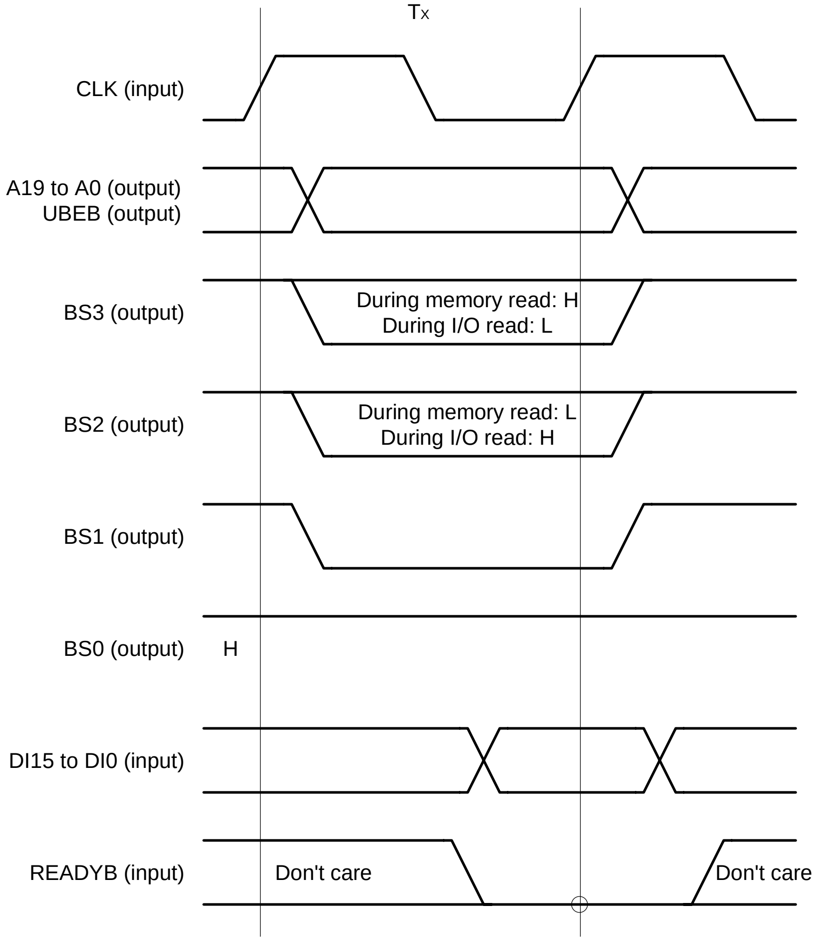
4.3.1 Read timing of memory and I/O

The V30MZ outputs the addresses (A19 to A0), UBEB signal, and bus status (BS3 to BS0) in synchronization with the rising edge of the clock (CLK).

Data (DI15 to DI0) is read at the rising edge of the next CLK signal, and the READYB signal is sampled at the same time. If the READYB signal is low at this time, the operation goes to the next bus cycle, and if the READYB signal is high, the operation goes to the wait cycle (TW), and the current bus cycle is extended. In the TW state, the A19 to A0 signals and the UBEB signal maintain their output value, but the BS3 to BS0 signals become high level.

Figure 4-2. Read Timing of Memory and I/O (1/2)

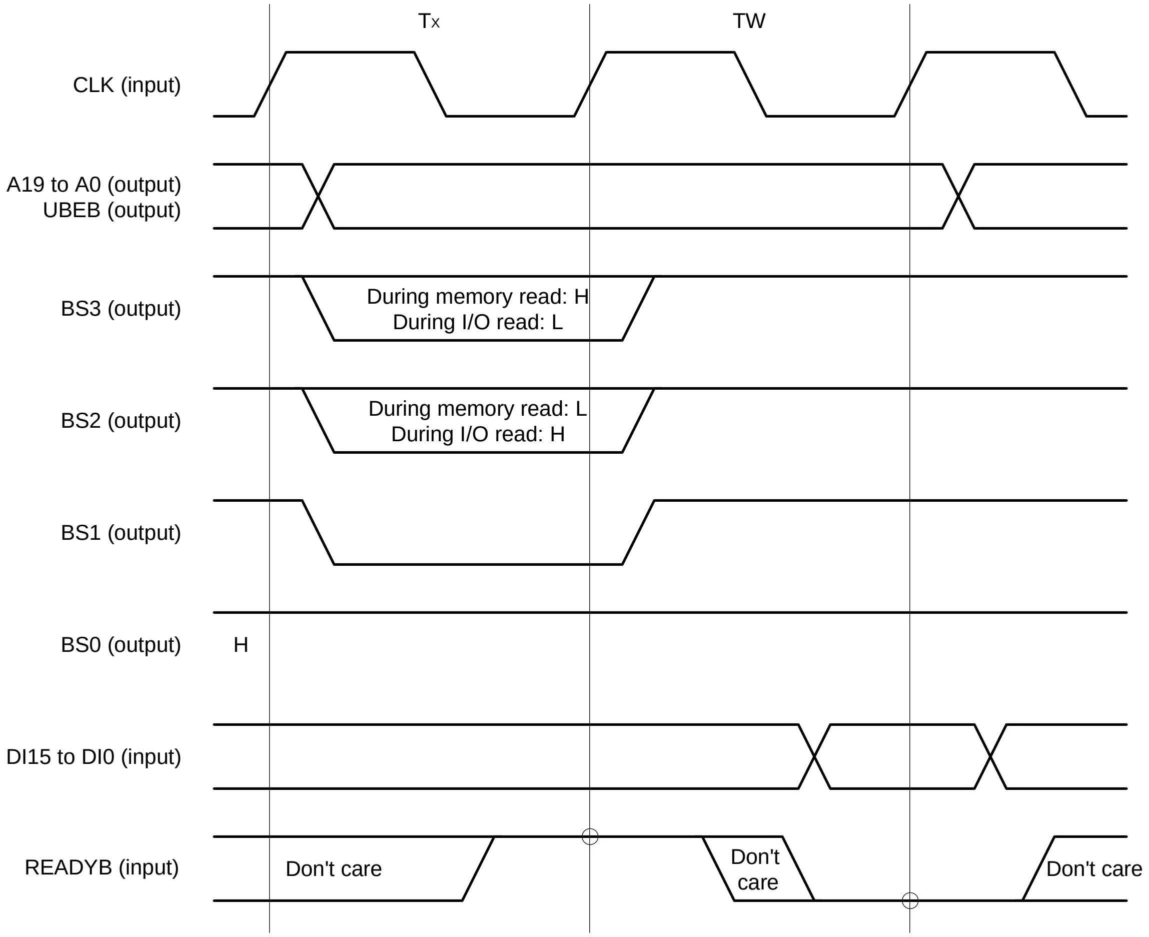
(a) With No Wait



Remark ○ indicates the sampling timing.

Figure 4-2. Read Timing of Memory and I/O (2/2)

(b) With 1 Wait



Remark ○ indicates the sampling timing.

4.3.2 Write timing of memory and I/O

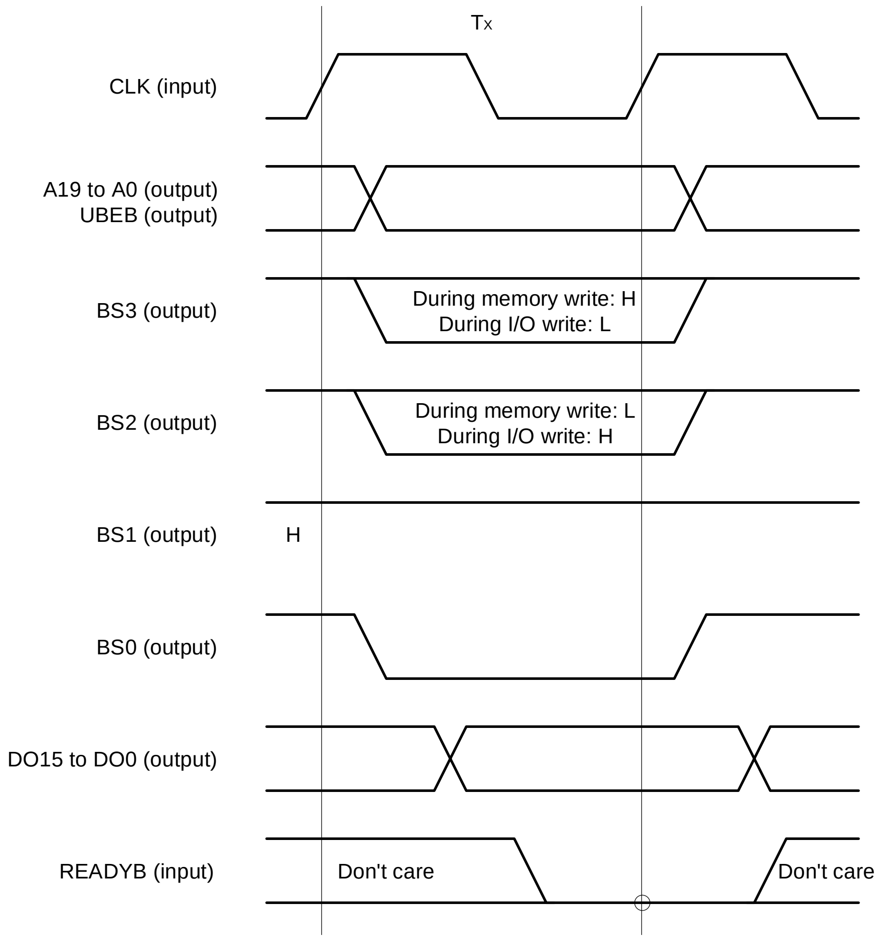
The V30MZ outputs the address (A19 to A0), UBEB signal, bus status (BS3 to BS0), and data (DO15 to DO0) in synchronization with the rising edge of the clock (CLK).

Then at the next rising edge of the CLK signal, it samples the READYB signal. If the READYB signal is low at this time, the operation goes to the next bus cycle, and if the READYB signal is high, the operation goes to the wait cycle (TW), and the current bus cycle is extended. In the TW state, invalid data is output from the DO15 to DO0 pins. Therefore, if a TW state is inserted to extend the bus cycle, latch the data from the DO15 to DO0 pins output at the first bus cycle using an external circuit.

Moreover, in the TW state, the A19 to A0 signals and the UBEB signal maintain their output value, but the BS3 to BS0 signals become high level.

Figure 4-3. Write Timing of Memory and I/O (1/2)

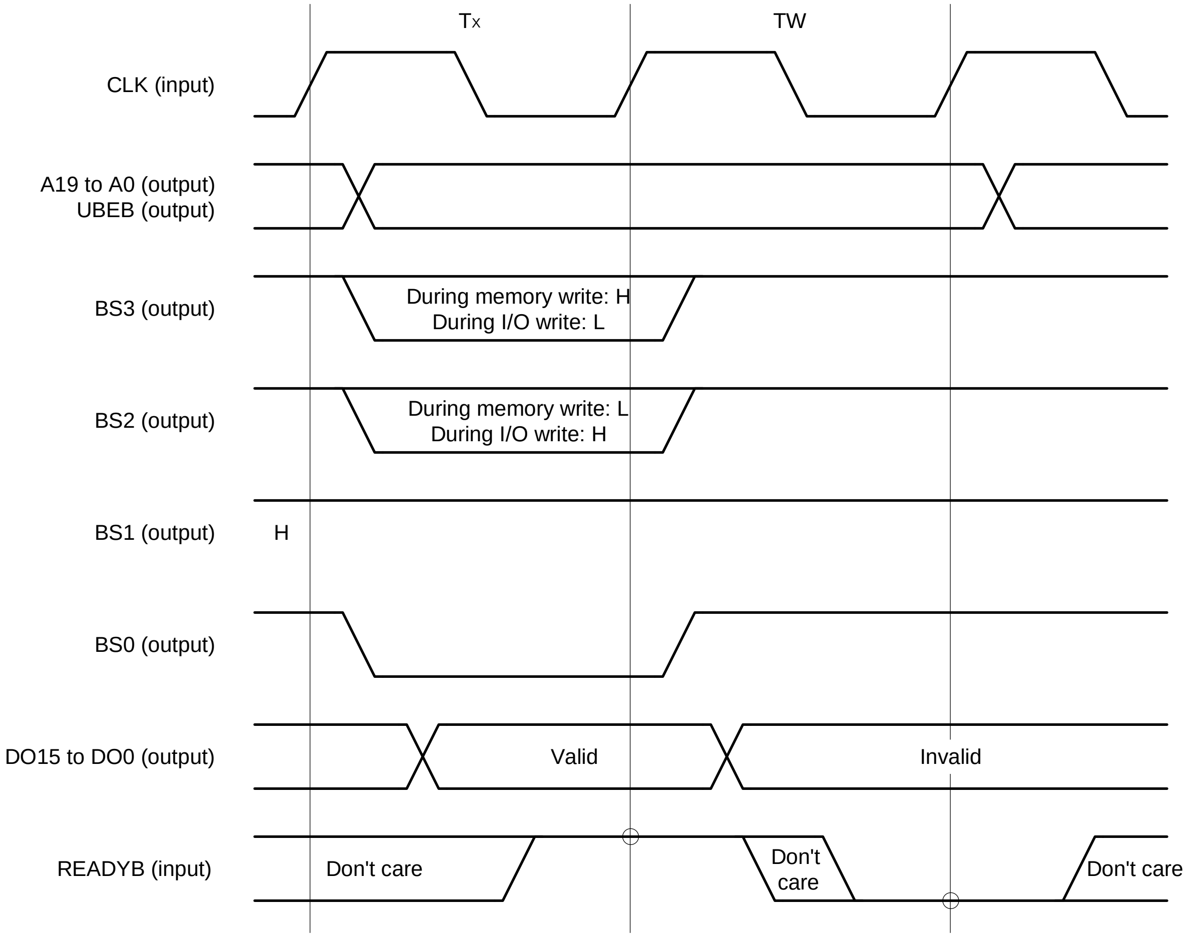
(a) With No Wait



Remark ○ indicates the sampling timing.

Figure 4-3. Write Timing of Memory and I/O (2/2)

(b) With 1 Wait



Remark ○ indicates the sampling timing.

4.4 Bus Hold Function

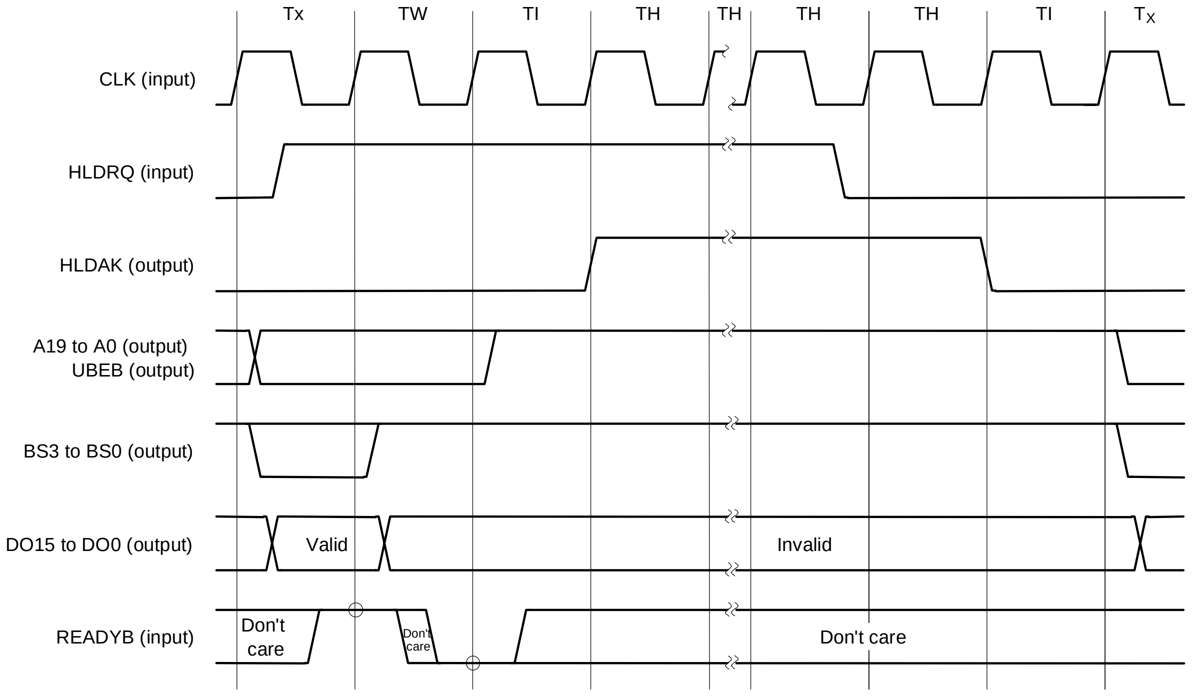
When a high level is input to the HLDRQ pin, the HLDAK signal becomes high level at the end of the bus cycle that is currently being executed, and the V30MZ enters the bus hold state (TH). However, an idle cycle (TI) lasting 1 clock is always inserted immediately before the TH state.

In the TH state, the A19 to A0 signals, the UBEB signal, and the BS3 to BS0 signal become high level, but the DO15 to DO0 signals output undefined data.

Next, when a low level is input to the HLDRQ pin, the HLDAK signal becomes low level, and the V30MZ returns to the normal bus cycle. However, a TI state lasting 1 clock is always inserted immediately after the TH state.

In the TH state, no code fetch cycle is generated.

Figure 4-4. Bus Hold Timing



Remark ○ indicates the sampling timing.

CHAPTER 5 INTERRUPT FUNCTIONS

Interrupts of the V30MZ are roughly divided into two kinds; hardware interrupts and software interrupts. These interrupts are all vectored interrupts that reference a vector table. An interrupt vector table stores the start address of an interrupt service routine.

When an interrupt is generated, the V30MZ references the fixed 4 bytes (fixed vector) in the vector table corresponding to the interrupt source or any 4 bytes (variable vector) specified each time and branches to the address stored there (start address of the interrupt service routine).

The interrupt vector table is assigned to a 1 K-byte area 000H to 3FFH of the memory space and can define a maximum of 256 vectors.

Table 5-1 shows the number of interrupt source clocks processed, vector numbers and priority order.

Figure 5-1 shows the interrupt vector table configuration.

Table 5-1. Interrupt Source List

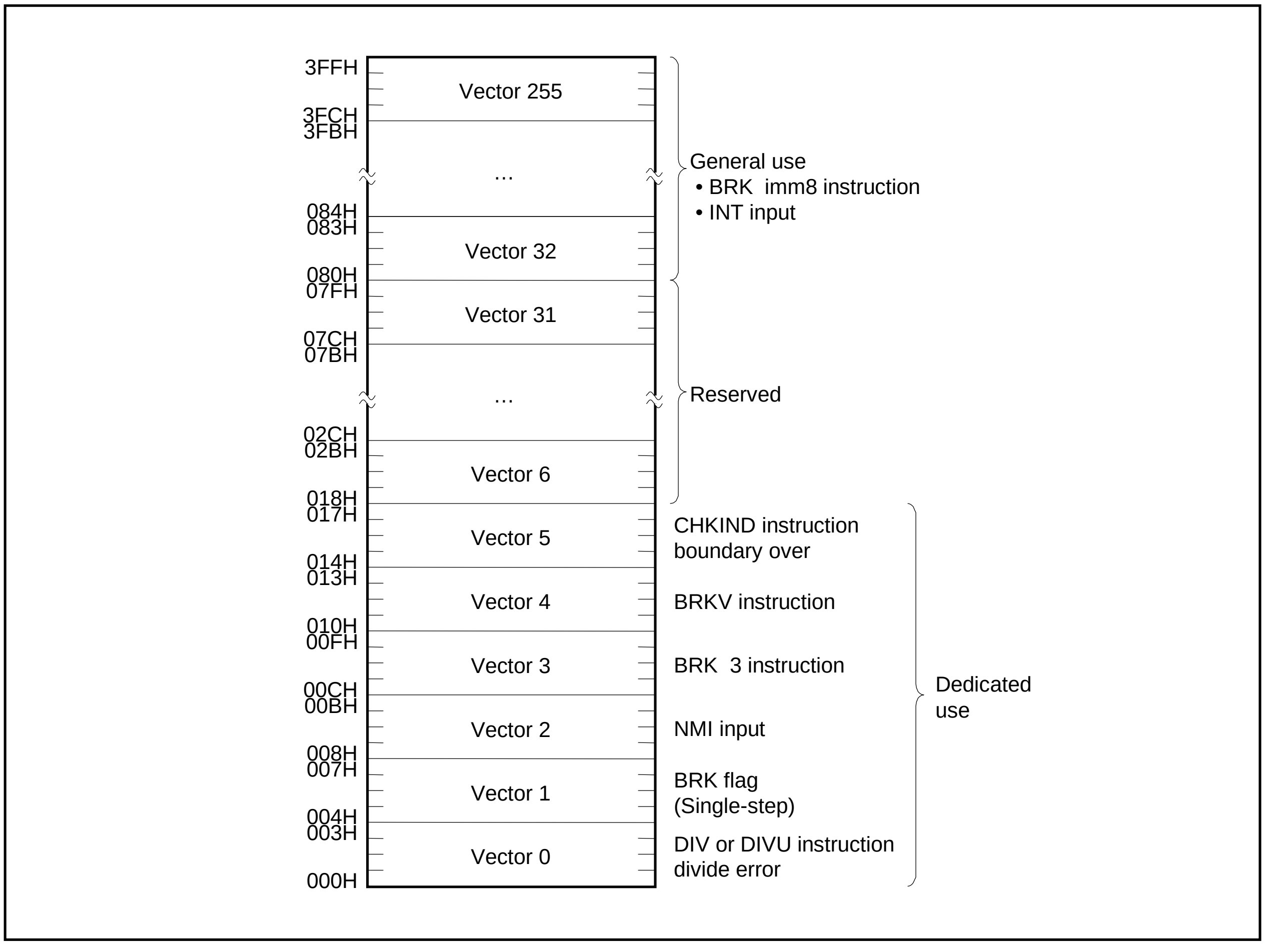
Interrupt Source		Number of Clocks Processed[Note]	Vector No.	Priority Order
Hardware interrupt	NMI input (rising edge active)	26	2	2
	INT input (high level active)	32	32 to 255	3
Software interrupt	DIV or DIVU instruction divide error	25	0	1
	CHKIND instruction boundary over		5	
	BRKV instruction		4	
	BRK 3 instruction		3	
	BRK imm8 instruction		32 to 255	
	BRK flag (single-step)		1	4

Note The number of clocks after execution of an instruction is aborted by an interrupt until the program branches to the start address of the interrupt service routine (progression of the wait state into the memory bus cycle and bus hold request are not taken into account).

Remark The following three instructions have a relatively long execution time, and even if an interrupt request is generated during their execution, their execution is not interrupted, and the interrupt request is acknowledged after the execution is completed. This point should be paid attention to in the case of systems for which the interrupt response time is particularly crucial.

Instruction	Number of Execution Clocks	Remarks
DIVU	24	When divide error not generated due to DIVU instruction
DIV	25	When divide error not generated due to DIV instruction
PREPARE	139	When 2nd operand = 31

Figure 5-1. Interrupt Vector Table Configuration



For vectors 0 to 5, the interrupt sources to be used are specified and vectors 6 to 31 are reserved and are not available for general use.

For vectors 32 to 255, BRK imm8 instruction, and INT input are available for general use.

One interrupt vector consists of 4 bytes and the higher address 2 bytes are loaded to the program segment register (PS) as a base address pointer (program segment value) and the lower address 2 bytes are loaded to the program counter (PC) as an offset value.

Example: Vector 0

003H
002H
001H
000H

PS ← (003H, 002H)
PC ← (001H, 000H)

When creating a program, initialize the content of each vector used based on the example above in the beginning of the program.

The following are the basic steps when jumping to an interrupt service routine.

TA← vector lower word data (offset value)
TC← vector higher word data (program segment value)
SP← SP–2, (SP+1, SP) ← PSW
IE← 0, BRK ← 0, MD ← 1
SP ← SP–2, (SP+1, SP) ← PS
PS ← TC
SP ← SP–2, (SP+1, SP) ← PC
PC ← TA

Caution Since the interrupt enable flag (IE) and break flag (BRK) of the program status word (PSW) are cleared (0) when interrupt servicing is started, no maskable interrupt (INT) or BRK flag (single-step) interrupt is acknowledged any longer.

5.1 Hardware Interrupt

There are two kinds of hardware interrupt.

- Non-maskable interrupt (NMI)
- Maskable interrupt (INT)

5.1.1 Non-maskable interrupt (NMI)

NMI is a non-maskable interrupt and cannot be disabled by software. Whenever there is an input to the NMI pin from a peripheral device, it is always acknowledged and detected on a rising edge.

NMI takes precedence over INT and is used to cope with abrupt variation of the normal power supply (instantaneous power failure) and memory error, bus error, etc.

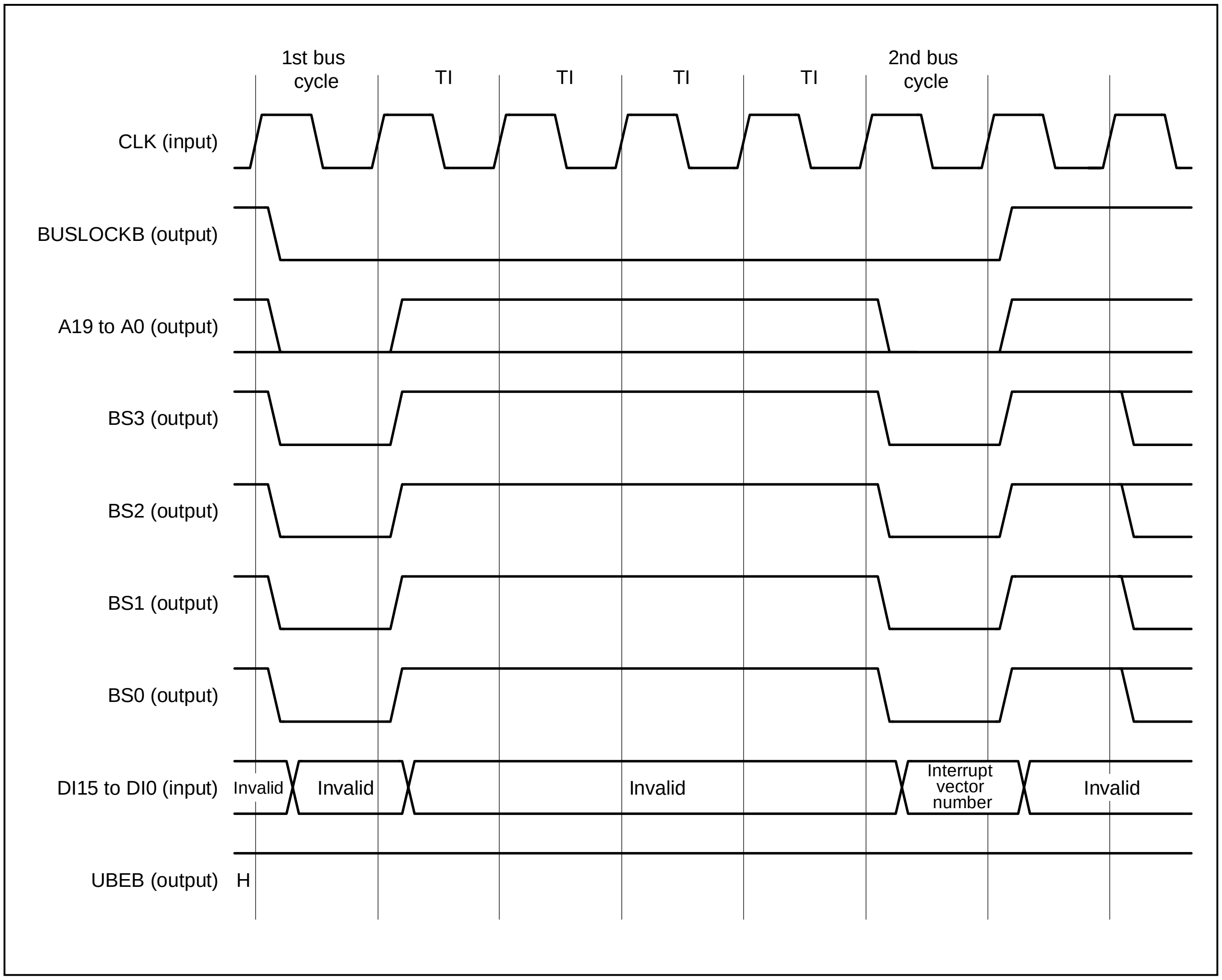
No interrupt acknowledge cycle is issued by NMI and no interrupt acknowledge is output, either.

Caution NMI requests are acknowledged even immediately after reset (INT requests are not acknowledged), but until the correct value is loaded to the stack pointer (SS:SP), normal NMI processing cannot be performed. Therefore, implement measures such as masking the NMI input with an external circuit.

5.1.2 Maskable interrupt (INT)

Maskable interrupts (INT) are acknowledged with the following response sequence. Prepare the interrupt acknowledge signal by decoding the BS3 to BS0 signals (when BS3 to BS0 are all low level).

Figure 5-2. Interrupt Acknowledge Cycle



(1) The 1st bus cycle is activated to obtain synchronization with the external interrupt controller. The values read from the data bus (DI15 to DI0) are not used.

At this time, address 00000H is output to the address bus (A19 to A0), but this value has no meaning. Moreover, a wait cycle can be inserted by using the READYB signal.

(2) Four clocks of idle cycle (TI) are inserted between the 1st bus cycle and the 2nd bus cycle. During this interval, the BUSLOCKB signal remains low. No code fetch cycle is generated between the 1st bus cycle and the 2nd bus cycle.

(3) The V30MZ reads the vector number from the interrupt controller during the 2nd bus cycle (only lower byte of data bus is valid). At this time, address 00000H is output to the address bus, but this value is meaningless.

Moreover, similarly to the 1st bus cycle, a wait cycle can be inserted by using the READYB signal.

Remark Whereas in the case of the V30HL, $\overline{\text{UBE}}$ output is always low level during the interrupt acknowledge cycle, in the case of the V30MZ, UBEB output is always high level. (UBEB output becomes low level only when the start address of the interrupt processing routine is read.)

5.2 Software Interrupts

Software interrupts take precedence over hardware interrupts except a BRK flag (single-step) interrupts.
They can be divided as follows.

(1) Interrupt by instruction result

- Divide error by DIV instruction or DIVU instruction
- Boundary over detection by CHKIND instruction

When the processing result of an instruction is invalid, an interrupt is automatically generated to allow exception handling.

(2) Interrupt by conditional break (execution of BRKV instruction)

In execution of a BRKV instruction, if the V flag is set (1), an interrupt is generated. It is used for processing an overflow of the operation result.

(3) Interrupt by unconditional break instruction

- 1-byte break instruction (BRK 3)
- 2-byte break instruction (BRK imm8 (≠ 3))

This interrupt is used when branching to a subroutine by a system call or inter-segment call without being aware of the branch destination.

(4) BRK flag (single-step) interrupt

This is a useful function for program debugging, etc.

This interrupt is controlled by the BRK flag of PSW. However, it is manipulated with the PSW saved to the stack, not by an instruction which directly sets/clears the BRK flag and set/cleared processing is indirectly performed by restoring it to the PSW.

When the BRK flag is set (1), after the next one instruction is executed, the interrupt routine (monitor program, etc.) specified by vector 1 is started and the BRK flag is also cleared (0) together with the IE flag at that time.

Therefore, once the vector 1 interrupt is started, interrupt routine instructions are not executed one by one but continuously in the same way as for other interrupts. Here, the internal registers, flag state, memory content, etc., can be checked and dumped.

In this interrupt routine, the number of single-steps is checked and if it is possible to terminate the single-step operation, the BRK flag in the stack is cleared (0) by a memory manipulation instruction and returned. This allows instructions to be executed continuously after returning to the main routine.

When returning without manipulating the BRK flag, BRK = 1 saved in the stack is restored to the PSW and after execution of one instruction in the main routine a vector 1 interrupt is generated again.

5.3 Timing at which Interrupt is Not Acknowledged

In the timing shown in (1) to (4) below, that is, between an instruction in which data is directly set in the segment register or 3 types of prefix and the following one instruction, no hardware interrupt or BRK flag (single-step) interrupt is acknowledged. Furthermore, only the INT interrupt is not acknowledged in the timing shown in (5).

With the following 5 timings, no interrupt is acknowledged.

(1) Between each of MOV SS, reg 16; MOV SS, mem16; POP SS instructions and the next instruction
(2) Between segment override prefix (PS:, SS:, DS0:, DS1:) and the next instruction
(3) Between repeat prefix (REP, REPE, REPNE) and the next instruction
(4) Between BUSLOCK instruction and the next instruction
(5) Between each of EI, RETI, and POP PSW instructions (in case an IE flag of the PSW register is set (1) by executing the instruction) and the next instruction (only INT interrupt)

However, an NMI request signal generated in interrupt disable timings in (1) to (4) is held pending internally and acknowledged after execution of the next one instruction is completed.

Moreover, if the timing of (5) is generated with an IE flag set, the INT interrupt is acknowledged even immediately after the execution of these instructions.

5.4 Interrupt Servicing in Execution of Block Processing Instruction

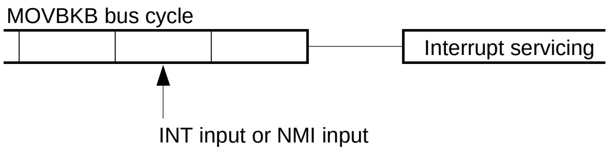
When a hardware interrupt request is generated in execution of a primitive block transfer/comparison, or input instruction, the V30MZ acknowledges it and branches to the corresponding interrupt address.

However, in a block processing instruction, immediately after completion of the bus cycle in which an interrupt is generated, the interrupt may not be acknowledged. In that case, it takes several bus cycles after generation of the interrupt until the V30MZ can acknowledge the interrupt. Table 5-2 shows the number of bus cycles. In this table, the bus cycle in which an interrupt is generated is counted as the first bus cycle.

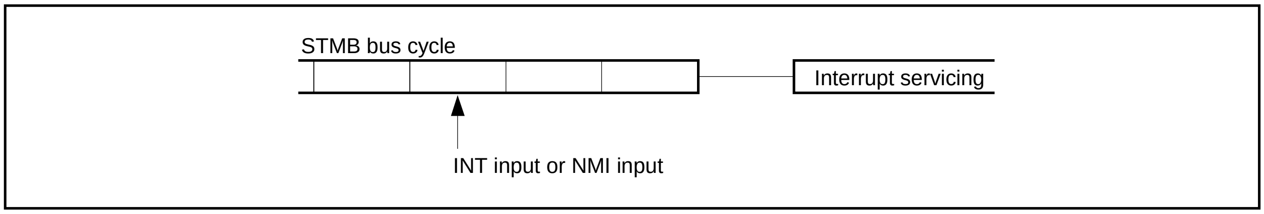
Table 5-2. Number of Bus Cycles Required until Interrupt is Acknowledged

Instruction	IX Register	IY Register	Number of Bus Cycles Required until Interrupt is Acknowledged
MOVBKW	Even	Even	2 to 4
	Even	Odd	3 to 6
	Odd	Even	2 to 5
	Odd	Odd	3 to 7
MOVBKB	–	–	2 to 4
CMPBKW	Even	Even	1, 2
	Even	Odd	1 to 3
	Odd	Even	1 to 3
	Odd	Odd	1 to 4
CMPBKB	–	–	1, 2
CMPMW	–	Even	1
	–	Odd	1, 2
CMPMB	–	–	1
LDMW	Even	–	1
	Odd	–	1, 2
LDMB	–	–	1
STMW	–	Even	3, 4
	–	Odd	3 to 5
STMB	–	–	3, 4

Example 1. When an interrupt request is generated in execution of MOVBKB instruction



Example 2. When an interrupt request is generated in execution of STMB instruction



If at the start of an interrupt service routine started in this way the CW register operating as a counter for the block data is saved to the stack and the saved CW register is restored at the end of the interrupt service routine and then the original routine is returned to by an RETI instruction, the suspended block processing can be restarted.

At this time, if a prefix is placed before the block processing instruction, the return address is modified (–1 address for one kind of prefix) and saved so that up to 3 kinds of prefix are stored and can be returned to the address at which the prefix is placed when returning from the interrupt service routine.

In order to use these functions effectively, set the sum of prefixes placed before a block processing instruction to three or less.

[MEMO]

CHAPTER 6 STANDBY FUNCTIONS

6.1 Setting of Standby Mode

Executing a HALT instruction sets the standby (HALT) mode.

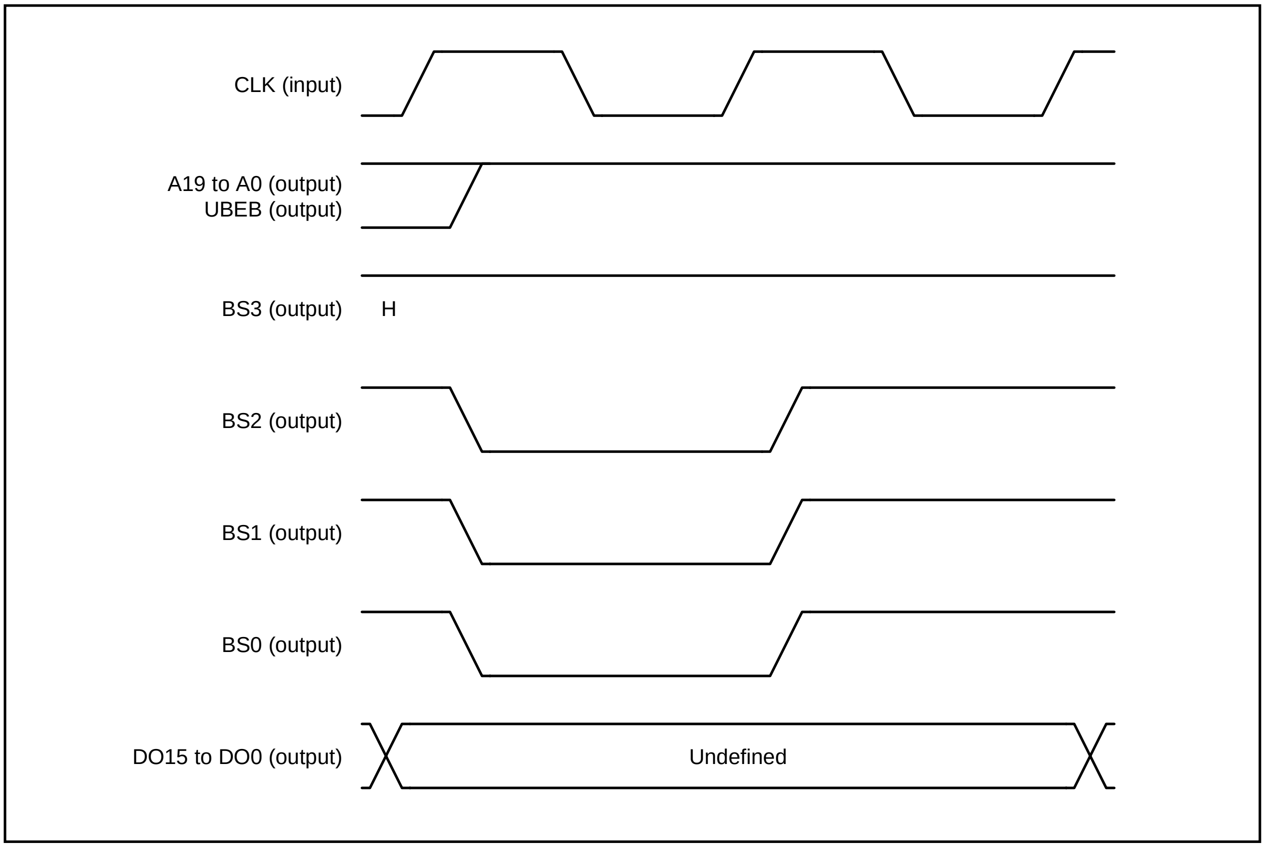
In the standby mode, the clock is supplied only to the circuit related to the function required for releasing the standby mode and the circuit related to the bus hold function, and its supply to all other circuits is stopped.

As a result, the system's power consumption is considerably reduced.

6.2 Standby Mode

When the V30MZ enters the standby mode, the BS3 to BS0 signals output the HALT status for 1 clock. Then the A19 to A0 signals, the UBEB signal, and the BS3 to BS0 signals become high level. DO15 to DO0 outputs become undefined.

Figure 6-1. Timing to Enter Standby Mode



The bus hold function is also valid in the standby mode, but when the bus hold acknowledge period ends, the system returns to the standby mode. However, when the system returns to the standby mode, there is no HALT status output from the BS3 to BS0 pins (the idle status continues).

6.3 Release of Standby Mode

There are two ways to release the standby mode: release by a hardware interrupt (NMI input or INT input) and release by RESET input. When both inputs become active at the same time, the normal interrupt priority order applies.

6.3.1 Release by hardware interrupt request

Cautions 1. When the bus hold request and hardware interrupt request are issued at the same time, the bus hold request takes priority, and after the bus hold cycle ends, the standby mode is released by hardware interrupt request.

2. When the HALT instruction and the hardware interrupt request are issued at the same time, the HALT status is output from the BS3 to BS0 pins, but the V30MZ does not enter the standby mode, and instead immediately performs interrupt processing (or the instruction following the HALT instruction).

(1) Release by NMI input

Upon detection of the rising edge of NMI input, the standby mode is released and the interrupt processing (NMI routine) starts. Then, when the RETI instruction is executed upon completion of the NMI routine, program execution resumes from the instruction following the HALT instruction.

(2) Release by INT input

The operation after release of the standby mode differs depending on whether the system is in the interrupt enable status (IE flag of PSW = 1) or in the interrupt disable status (IE flag of PSW = 0).

(a) Interrupt enable status

When there is an INT input, the standby mode is released and interrupt processing (INT routine) starts. Then, when the RETI instruction is executed upon completion of the INT routine, program execution resumes from the instruction following the HALT instruction.

Caution Input the high level to the INT pin until the 1st bus cycle of the interrupt acknowledge cycle.

(b) Interrupt disable status

When there is an INT input, the standby mode is released and program execution resumes from the instruction following the HALT instruction.

Caution Input the high level to the INT pin for the interval of one clock or longer.

6.3.2 Release by RESET input

When the RESET signal is input in the standby mode, the standby mode is released unconditionally, and the system starts normal reset operation. Therefore, the status that was held in the standby mode becomes invalid, and the program that was stopped by the standby mode cannot be resumed.

[MEMO]

CHAPTER 7 RESET FUNCTIONS

When a high level is input to the RESET pin for 4 clocks or more, each output pin of the V30MZ changes to the statuses shown in Table 7-1. They retain these values while the high level is input.

A setup time and hold time are prescribed for the rising edge of the CLK pin for RESET input. Be sure to perform RESET input so as to satisfy these prescriptions.

Table 7-1. Status of Output Pins after Reset

Pin	Status after Reset
A19 to A0	High-level output
DO15 to DO0	Undefined
UBEB	High-level output
BUSLOCKB	
BS3 to BS0	
HLDAK	Low-level output
TBO42 to TBO0	High impedance

Each register is initialized to the value shown in Table 7-2 following reset.

Table 7-2. Initial Value of Registers after Reset

<table>
<tr><th>Register</th><th>Initial Value</th></tr>
<tr><td>PC</td><td>0000H</td></tr>
<tr><td>PFP</td><td></td></tr>
<tr><td>PS</td><td>FFFFH</td></tr>
<tr><td>SS, DS0, DS1</td><td>0000H</td></tr>
<tr><td>AW, BW, CW, DW</td><td>Undefined</td></tr>
<tr><td>SP, BP</td><td></td></tr>
<tr><td>IX, IY</td><td></td></tr>
<tr><td>PSW</td><td><table>
<tr><td></td><td>MD</td><td></td><td></td><td></td><td>V</td><td>DIR</td><td>IE</td><td>BRK</td></tr>
<tr><td>Higher</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td></td><td>S</td><td>Z</td><td></td><td>AC</td><td></td><td>P</td><td></td><td>CY</td></tr>
<tr><td>Lower</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
</table></td></tr>
</table>

When the signal input to the RESET pin goes back to low level, the V30MZ starts instruction prefetch from address FFFF0H (segment value: FFFFH, offset value: 0000H).

[MEMO]

CHAPTER 8 TEST FUNCTIONS

The V30MZ has unit test functions using the test bus like other CBIC cores.

8.1 Test Pins

The V30MZ has the following test pins.

- TBI22 to TBI0
- TBO42 to TBO0
- BUNRI
- TEST

8.1.1 Test bus pins (TBI22 to TBI0, TBO42 to TBO0)

The test bus pins are used instead of normal pins in the unit test mode.
For details, see the Design Manual User's Manual of each cell-based IC Family.

8.1.2 BUNRI, TEST pins

These pins are used to select the normal, unit test, or standby test mode.

Table 8-1. Test Mode Selection List

BUNRI Pin Input Level	TEST Pin Input Level	Mode
Low level	don't care	Normal mode
High level	Low level	Standby test mode
High level	High level	Unit test mode

8.2 Normal Mode

This is the mode normally used by the user.

When the low level is input to the BUNRI pin, pins other than test pins become valid and the normal mode is entered. At this time, inputs to the TBI22 to TBI0 pins are ignored, and the TBO42 to TBO0 pins go into high impedance.

8.3 Unit Test Mode and Standby Test Mode

When the high level is input to the BUNRI pin, inputs to pins other than test pins are ignored (become invalid), and the system enters the test mode. There are two test modes, the unit test mode and the standby test mode.

Perform circuit design so that the bus configuration pins (except test pins) do not become floating level or cause bus contention during the unit test mode and the standby test mode. (For details on the status of pins in each mode, see **Section 2.2 Pin Statuses**.)

8.3.1 Unit test mode

When the high level is input to the BUNRI pin and the TEST pin, the system enters the unit test mode. In the unit test mode, inputs to pins other than test pins are ignored, and inputs to the TBI22 to TBI0 pins become valid instead. Moreover, values for the pins other than test pins are output to the TBO42 to TBO0 pins.

Caution The unit test mode is a mode for testing performed by NEC.

8.3.2 Standby test mode

When the high level is input to the BUNRI pin and the low level is input to the TEST pin, the system enters the standby test mode.

This mode is used for cores that are not tested during test circuit verification simulation and user logic separation simulation.

Inputs to the TBI22 to TBI0 pins are ignored and the TBO42 to TBO0 pins go into high impedance.

APPENDIX A LIST OF INSTRUCTION EXECUTION CLOCK COUNTS

This appendix shows the number of execution clocks for each instruction under conditions (1) to (7) listed below. For details on the functions of each instruction, refer to the **16-Bit V Series Instruction User's Manual**.

(1) Instruction decoding is completed.
(2) No wait state occurs during memory access or I/O access.[Note 1]
(3) There is no bus hold request.
(4) Word data is allocated to even addresses.[Note 2]
(5) Registers required for calculating effective addresses (BW, BP, SP, IX, IY, etc.) do not change at immediately preceding instruction.[Note 3]
(6) There is only 1 register required for calculating effective addresses.[Note 4]
(7) The branching destination of a branch instruction is an even address.[Note 5]

Notes **1.** If a wait state is generated, add the number of clocks of the wait state to the number of instruction execution clocks.

2. When access to word data allocated to odd addresses is performed, add 1 clock.

3. When using a register that changed in the immediately preceding instruction for calculating the effective address, add 1 clock.
(However, 1 clock does not increase in the case of the LDEA instruction and the repeat prefetch instruction.)
Moreover, in the case of consecutive PUSH or POP instruction, the number of execution clocks does not increase (they are all executed in 1 clock.)

4. If there are two registers required for calculating the effective address (for example MOV AW,[BW+IX]), add 1 clock.

5. If branching to an odd address is performed, add 1 clock.

Table A-1. List of Instruction Execution Clock Counts (1/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
ADD acc, imm	0	0	0	0	0	1	0	W	imm8 or imm16-low								1
	imm16-high								—								
ADD mem, imm	1	0	0	0	0	0	s	W	mod		0	0	0	mem			3
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-high								
ADD mem, reg	0	0	0	0	0	0	0	W	mod		reg			mem			3
	(disp-low)								(disp-high)								
ADD reg, imm	1	0	0	0	0	0	s	W	1	1	0	0	0	reg			1
	imm8 or imm16-low								imm16-high								
ADD reg, mem	0	0	0	0	0	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
ADD reg, reg'	0	0	0	0	0	0	1	W	1	1	reg			reg'			1
ADD reg, reg'	0	0	0	0	0	0	0	W	1	1	reg			reg'			1
ADDC acc, imm	0	0	0	1	0	1	0	W	imm8 or imm16-low								1
	imm16-high								—								
ADDC mem, imm	1	0	0	0	0	0	s	W	mod		0	1	0	mem			3
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-high								
ADDC mem, reg	0	0	0	1	0	0	0	W	mod		reg			mem			3
	(disp-low)								(disp-high)								
ADDC reg, imm	1	0	0	0	0	0	s	W	1	1	0	1	0	reg			1
	imm8 or imm16-low								imm16-high								
ADDC reg, mem	0	0	0	1	0	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
ADDC reg, reg'	0	0	0	1	0	0	1	W	1	1	reg			reg'			1
ADDC reg, reg'	0	0	0	1	0	0	0	W	1	1	reg			reg'			1
ADJ4A	0	0	1	0	0	1	1	1	—								10
ADJ4S	0	0	1	0	1	1	1	1	—								10
ADJBA	0	0	1	1	0	1	1	1	—								9
ADJBS	0	0	1	1	1	1	1	1	—								9
AND acc, imm	0	0	1	0	0	1	0	W	imm8 or imm16-low								1
	imm16-high								—								
AND mem, imm	1	0	0	0	0	0	0	W	mod		1	0	0	mem			3
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-high								
AND mem, reg	0	0	1	0	0	0	0	W	mod		reg			mem			3
	(disp-low)								(disp-high)								
AND reg, imm	1	0	0	0	0	0	0	W	1	1	1	0	0	reg			1
	imm8 or imm16-low								imm16-high								
AND reg, mem	0	0	1	0	0	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								

Table A-1. List of Instruction Execution Clock Counts (2/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
AND reg, reg'	0	0	1	0	0	0	1	W	1	1	reg			reg'			1
AND reg, reg'	0	0	1	0	0	0	0	W	1	1	reg			reg'			1
BC short-label	0	1	1	1	0	0	1	0	disp8								1 (When CY = 0)
BC short-label	0	1	1	1	0	0	1	0	disp8								4 (When CY = 1)
BCWZ short-label	1	1	1	0	0	0	1	1	disp8								1 (When CW ≠ 0)
BCWZ short-label	1	1	1	0	0	0	1	1	disp8								4 (When CW = 0)
BE short-label	0	1	1	1	0	1	0	0	disp8								1 (When Z = 0)
BE short-label	0	1	1	1	0	1	0	0	disp8								4 (When Z = 1)
BGE short-label	0	1	1	1	1	1	0	1	disp8								1 (When S ∀ V = 1)
BGE short-label	0	1	1	1	1	1	0	1	disp8								4 (When S ∀ V = 0)
BGT short-label	0	1	1	1	1	1	1	1	disp8								1 (When (S ∀ V) ∨ Z = 1)
BGT short-label	0	1	1	1	1	1	1	1	disp8								4 (When (S ∀ V) ∨ Z = 0)
BH short-label	0	1	1	1	0	1	1	1	disp8								1 (When CY ∨ Z = 1)
BH short-label	0	1	1	1	0	1	1	1	disp8								4 (When CY ∨ Z = 0)
BL short-label	0	1	1	1	0	0	1	0	disp8								1 (When CY = 0)
BL short-label	0	1	1	1	0	0	1	0	disp8								4 (When CY = 1)
BLE short-label	0	1	1	1	1	1	1	0	disp8								1 (When (S ∀ V) ∨ Z = 0)
BLE short-label	0	1	1	1	1	1	1	0	disp8								4 (When (S ∀ V) ∨ Z = 1)
BLT short-label	0	1	1	1	1	1	0	0	disp8								1 (When S ∀ V = 0)
BLT short-label	0	1	1	1	1	1	0	0	disp8								4 (When S ∀ V = 1)
BN short-label	0	1	1	1	1	0	0	0	disp8								1 (When S = 0)
BN short-label	0	1	1	1	1	0	0	0	disp8								4 (When S = 1)
BNC short-label	0	1	1	1	0	0	1	1	disp8								1 (When CY = 1)
BNC short-label	0	1	1	1	0	0	1	1	disp8								4 (When CY = 0)
BNE short-label	0	1	1	1	0	1	0	1	disp8								1 (When Z = 1)
BNE short-label	0	1	1	1	0	1	0	1	disp8								4 (When Z = 0)
BNH short-label	0	1	1	1	0	1	1	0	disp8								1 (When CY ∨ Z = 0)
BNH short-label	0	1	1	1	0	1	1	0	disp8								4 (When CY ∨ Z = 1)
BNL short-label	0	1	1	1	0	0	1	1	disp8								1 (When CY = 1)
BNL short-label	0	1	1	1	0	0	1	1	disp8								4 (When CY = 0)
BNV short-label	0	1	1	1	0	0	0	1	disp8								1 (When V = 1)
BNV short-label	0	1	1	1	0	0	0	1	disp8								4 (When V = 0)
BNZ short-label	0	1	1	1	0	1	0	1	disp8								1 (When Z = 1)
BNZ short-label	0	1	1	1	0	1	0	1	disp8								4 (When Z = 0)
BP short-label	0	1	1	1	1	0	0	1	disp8								1 (When S = 1)
BP short-label	0	1	1	1	1	0	0	1	disp8								4 (When S = 0)
BPE short-label	0	1	1	1	1	0	1	0	disp8								1 (When P = 0)
BPE short-label	0	1	1	1	1	0	1	0	disp8								4 (When P = 1)
BPO short-label	0	1	1	1	1	0	1	1	disp8								1 (When P = 1)
BPO short-label	0	1	1	1	1	0	1	1	disp8								4 (When P = 0)

Table A-1. List of Instruction Execution Clock Counts (3/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
BR far-label	1	1	1	0	1	0	1	0	offset-low								7
	offset-high								seg-low								
	seg-high								—								
BR memptr16	1	1	1	1	1	1	1	1	mod		1	0	0	mem			5
	(disp-low)								(disp-high)								
BR memptr32	1	1	1	1	1	1	1	1	mod		1	0	1	mem			10
	(disp-low)								(disp-high)								
BR near-label	1	1	1	0	1	0	0	1	disp-low								4
	disp-high								—								
BR regptr16	1	1	1	1	1	1	1	1	1	1	1	0	0	reg			4
BR short-label	1	1	1	0	1	0	1	1	disp8								4
BRK 3	1	1	0	0	1	1	0	0	—								9
BRK imm8 (≠ 3)	1	1	0	0	1	1	0	1	imm8								10
BRKV	1	1	0	0	1	1	1	0	—								6 (When V = 0)
BRKV	1	1	0	0	1	1	1	0	—								13 (When V = 1)
BUSLOCK	1	1	1	1	0	0	0	0	—								1
BV short-label	0	1	1	1	0	0	0	0	disp8								1 (When V = 0)
BV short-label	0	1	1	1	0	0	0	0	disp8								4 (When V = 1)
BZ short-label	0	1	1	1	0	1	0	0	disp8								1 (When Z = 0)
BZ short-label	0	1	1	1	0	1	0	0	disp8								4 (When Z = 1)
CALL far-proc	1	0	0	1	1	0	1	0	offset-low								10
	offset-high								seg-low								
	seg-high								—								
CALL memptr16	1	1	1	1	1	1	1	1	mod		0	1	0	mem			6
	(disp-low)								(disp-high)								
CALL memptr32	1	1	1	1	1	1	1	1	mod		0	1	1	mem			12
	(disp-low)								(disp-high)								
CALL near-proc	1	1	1	0	1	0	0	0	disp-low								5
	disp-high								—								
CALL regptr16	1	1	1	1	1	1	1	1	1	1	0	1	0	reg			5
CHKIND reg16, mem32	0	1	1	0	0	0	1	0	mod		reg			mem			13 (when interrupt condition is not satisfied)
	(disp-low)								(disp-high)								
CHKIND reg16, mem32	0	1	1	0	0	0	1	0	mod		reg			mem			20 (when interrupt condition is satisfied)
	(disp-low)								(disp-high)								
CLR1 CY	1	1	1	1	1	0	0	0	—								4
CLR1 DIR	1	1	1	1	1	1	0	0	—								4
CMP acc, imm	0	0	1	1	1	1	0	W	imm8 or imm16-low								1
	imm16-high								—								
CMP mem, imm	1	0	0	0	0	0	s	W	mod		1	1	1	mem			2
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-high								

Table A-1. List of Instruction Execution Clock Counts (4/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
CMP mem, reg	0	0	1	1	1	0	0	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
CMP reg, imm	1	0	0	0	0	0	s	W	1	1	1	1	1	reg			1
	imm8 or imm16-low								imm16-high								
CMP reg, mem	0	0	1	1	1	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
CMP reg, reg'	0	0	1	1	1	0	1	W	1	1	reg			reg'			1
CMP reg, reg'	0	0	1	1	1	0	0	W	1	1	reg			reg'			1
CMPBK [DS1-spec:]dst-block	1	0	1	0	0	1	1	W	—								6
CMPBK [Seg-spec:]src-block, [DS1-spec:]dst-block	1	0	1	0	0	1	1	W	—								6
CMPBKB	1	0	1	0	0	1	1	W	—								6
CMPBKW	1	0	1	0	0	1	1	W	—								6
CMPM [DS1-spec:]dst-block	1	0	1	0	1	1	1	W	—								4
CMPMB	1	0	1	0	1	1	1	W	—								4
CMPMW	1	0	1	0	1	1	1	W	—								4
CVTBD	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	0	17
CVTBW	1	0	0	1	1	0	0	0	—								1
CVTDB	1	1	0	1	0	1	0	1	0	0	0	0	1	0	1	0	6
CVTWL	1	0	0	1	1	0	0	1	—								1
DBNZ short-label	1	1	1	0	0	0	1	0	disp8								2 (When CW = 0)
DBNZ short-label	1	1	1	0	0	0	1	0	disp8								5 (When CW ≠ 0)
DBNZE short-label	1	1	1	0	0	0	0	1	disp8								6 (when CW ≠ 0 and Z = 1)
DBNZE short-label	1	1	1	0	0	0	0	1	disp8								3 (in cases other than above)
DBNZNE short-label	1	1	1	0	0	0	0	0	disp8								6 (when CW ≠ 0 and Z = 0)
DBNZNE short-label	1	1	1	0	0	0	0	0	disp8								3 (in cases other than above)
DEC mem	1	1	1	1	1	1	1	W	mod		0	0	1	mem			3
	(disp-low)								(disp-high)								
DEC reg16	0	1	0	0	1	reg			—								1
DEC reg8	1	1	1	1	1	1	1	W	1	1	0	0	1	reg			1
DI	1	1	1	1	1	0	1	0	—								4
DISPOSE	1	1	0	0	1	0	0	1	—								2
DIV mem16	1	1	1	1	0	1	1	W	mod		1	1	1	mem			25
	(disp-low)								(disp-high)								
DIV mem8	1	1	1	1	0	1	1	W	mod		1	1	1	mem			18
	(disp-low)								(disp-high)								
DIV reg16	1	1	1	1	0	1	1	W	1	1	1	1	1	reg			24
DIV reg8	1	1	1	1	0	1	1	W	1	1	1	1	1	reg			17
DIVU mem16	1	1	1	1	0	1	1	W	mod		1	1	0	mem			24
	(disp-low)								(disp-high)								
DIVU mem8	1	1	1	1	0	1	1	W	mod		1	1	0	mem			16
	(disp-low)								(disp-high)								

Table A-1. List of Instruction Execution Clock Counts (5/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
DIVU reg16	1	1	1	1	0	1	1	W	1	1	1	1	0	reg			23
DIVU reg8	1	1	1	1	0	1	1	W	1	1	1	1	0	reg			15
DS0:	0	0	1	1	1	1	1	0	—								1
DS1:	0	0	1	0	0	1	1	0	—								1
EI	1	1	1	1	1	0	1	1	—								4
FPO1 fp-op, mem	1	1	0	1	1	X	X	X	mod		Y	Y	Y	mem			1
	(disp-low)								(disp-high)								
HALT	1	1	1	1	0	1	0	0	—								9
IN acc, DW	1	1	1	0	1	1	0	W	—								6
IN acc, imm8	1	1	1	0	0	1	0	W	imm8								6
INC mem	1	1	1	1	1	1	1	W	mod		0	0	0	mem			3
	(disp-low)								(disp-high)								
INC reg16	0	1	0	0	0	reg			—								1
INC reg8	1	1	1	1	1	1	1	W	1	1	0	0	0	reg			1
INM [DS1-spec:]dst-block, DW	0	1	1	0	1	1	0	W	—								6
LDEA reg16, mem16	1	0	0	0	1	1	0	1	mod		reg			mem			1
	(disp-low)								(disp-high)								
LDM [Seg-spec:]src-block	1	0	1	0	1	1	0	W	—								3
LDMB	1	0	1	0	1	1	0	W	—								3
LDMW	1	0	1	0	1	1	0	W	—								3
MOV acc, dmem	1	0	1	0	0	0	0	W	addr-low								1
	addr-high								—								
MOV AH, PSW	1	0	0	1	1	1	1	1	—								2
MOV dmem, acc	1	0	1	0	0	0	1	W	addr-low								1
	addr-high								—								
MOV DS0, reg16, mem32	1	1	0	0	0	1	0	1	mod		reg			mem			6
	(disp-low)								(disp-high)								
MOV DS1, reg16, mem32	1	1	0	0	0	1	0	0	mod		reg			mem			6
	(disp-low)								(disp-high)								
MOV mem, imm	1	1	0	0	0	1	1	W	mod		0	0	0	mem			1
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-high								
MOV mem, reg	1	0	0	0	1	0	0	W	mod		reg			mem			1
	(disp-low)								(disp-high)								
MOV mem16, sreg	1	0	0	0	1	1	0	0	mod		0	sreg		mem			3
	(disp-low)								(disp-high)								
MOV PSW, AH	1	0	0	1	1	1	1	0	—								4
MOV reg, imm	1	0	1	1	W	reg			imm8 or imm16-low								1
	imm16-high								—								
MOV reg, imm	1	1	0	0	0	1	1	W	1	1	0	0	0	reg			1
	imm8 or imm16-low								imm16-high								

Table A-1. List of Instruction Execution Clock Counts (6/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
MOV reg, mem	1	0	0	0	1	0	1	W	mod		reg			mem			1
	(disp-low)								(disp-high)								
MOV reg, reg'	1	0	0	0	1	0	1	W	1	1	reg			reg'			1
MOV reg, reg'	1	0	0	0	1	0	0	W	1	1	reg			reg'			1
MOV reg16, sreg	1	0	0	0	1	1	0	0	1	1	0	sreg		reg			1
MOV sreg, mem16	1	0	0	0	1	1	1	0	mod		0	sreg		mem			3
	(disp-low)								(disp-high)								
MOV sreg, reg16	1	0	0	0	1	1	1	0	1	1	0	sreg		reg			2
MOVBK [DS1-spec:]dst-block, [Seg-spec:]src-block	1	0	1	0	0	1	0	W	—								5
MOVBK [Seg-spec:]src-block	1	0	1	0	0	1	0	W	—								5
MOVBKB	1	0	1	0	0	1	0	W	—								5
MOVBKW	1	0	1	0	0	1	0	W	—								5
MUL mem16	1	1	1	1	0	1	1	W	mod		1	0	1	mem			4
	(disp-low)								(disp-high)								
MUL mem8	1	1	1	1	0	1	1	W	mod		1	0	1	mem			4
	(disp-low)								(disp-high)								
MUL reg16	1	1	1	1	0	1	1	W	1	1	1	0	1	reg			3
MUL reg16, imm16	0	1	1	0	1	0	s	1	1	1	reg			reg'			3
	imm16-low								imm16-high								
MUL reg16, imm8	0	1	1	0	1	0	s	1	1	1	reg			reg'			3
	imm8								—								
MUL reg16, mem16, imm16	0	1	1	0	1	0	s	1	mod		reg			mem			4
	(disp-low)								(disp-high)								
	imm16-low								imm16-high								
MUL reg16, mem16, imm8	0	1	1	0	1	0	s	1	mod		reg			mem			4
	(disp-low)								(disp-high)								
	imm8								—								
MUL reg16, reg16' , imm16	0	1	1	0	1	0	s	1	1	1	reg			reg'			3
	imm16-low								imm16-high								
MUL reg16, reg16' , imm8	0	1	1	0	1	0	s	1	1	1	reg			reg'			3
	imm8								—								
MUL reg8	1	1	1	1	0	1	1	W	1	1	1	0	1	reg			3
MULU mem16	1	1	1	1	0	1	1	W	mod		1	0	0	mem			4
	(disp-low)								(disp-high)								
MULU mem8	1	1	1	1	0	1	1	W	mod		1	0	0	mem			4
	(disp-low)								(disp-high)								
MULU reg16	1	1	1	1	0	1	1	W	1	1	1	0	0	reg			3
MULU reg8	1	1	1	1	0	1	1	W	1	1	1	0	0	reg			3
NEG mem	1	1	1	1	0	1	1	W	mod		0	1	1	mem			3
	(disp-low)								(disp-high)								
NEG reg	1	1	1	1	0	1	1	W	1	1	0	1	1	reg			1

Table A-1. List of Instruction Execution Clock Counts (7/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
NOP	1	0	0	1	0	0	0	0	—								1
NOT mem	1	1	1	1	0	1	1	W	mod		0	1	0	mem			3
	(disp-low)								(disp-high)								
NOT reg	1	1	1	1	0	1	1	W	1	1	0	1	0	reg			1
NOT1 CY	1	1	1	1	0	1	0	1	—								4
OR acc, imm	0	0	0	0	1	1	0	W	imm8 or imm16-low								1
	imm16-high																
OR mem, imm	1	0	0	0	0	0	0	W	mod		0	0	1	mem			3
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-high								
OR mem, reg	0	0	0	0	1	0	0	W	mod		reg			mem			3
	(disp-low)								(disp-high)								
OR reg, imm	1	0	0	0	0	0	0	W	1	1	0	0	1	reg			1
	imm8 or imm16-low								imm16-high								
OR reg, mem	0	0	0	0	1	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
OR reg, reg'	0	0	0	0	1	0	1	W	1	1	reg			reg'			1
OR reg, reg'	0	0	0	0	1	0	0	W	1	1	reg			reg'			1
OUT DW, acc	1	1	1	0	1	1	1	W	—								6
OUT imm8, acc	1	1	1	0	0	1	1	W	imm8								6
OUTM DW, [Seg-spec:]src-block	0	1	1	0	1	1	1	W	—								7
POLL	1	0	0	1	1	0	1	1	—								Number of 9 + 9 × POLLB pin samplings
POP mem16	1	0	0	0	1	1	1	1	mod		0	0	0	mem			3
	(disp-low)								(disp-high)								
POP PSW	1	0	0	1	1	1	0	1	—								3
POP R	0	1	1	0	0	0	0	1	—								8
POP reg16	0	1	0	1	1	reg			—								1
POP reg16	1	0	0	0	1	1	1	1	1	1	0	0	0	reg			1
POP sreg	0	0	0	sreg		1	1	1	—								3
PREPARE imm16, imm8 (when imm8 = 0)	1	1	0	0	1	0	0	0	imm16-low								8
	imm16-high								imm8								
PREPARE imm16, imm8 (when imm8 = 1)	1	1	0	0	1	0	0	0	imm16-low								14
	imm16-high								imm8								
PREPARE imm16, imm8 (when imm8 > 1)	1	1	0	0	1	0	0	0	imm16-low								15 + 4 × imm8
	imm16-high								imm8								
PS:	0	0	1	0	1	1	1	0	—								1
PUSH imm16	0	1	1	0	1	0	s	0	imm16-low								1
	imm16-high								—								
PUSH imm8	0	1	1	0	1	0	s	0	imm8								1
PUSH mem16	1	1	1	1	1	1	1	1	mod		1	1	0	mem			2
	(disp-low)								(disp-high)								

Table A-1. List of Instruction Execution Clock Counts (8/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
PUSH PSW	1	0	0	1	1	1	0	0	—								2
PUSH R	0	1	1	0	0	0	0	0	—								9
PUSH reg16	0	1	0	1	0	reg			—								1
PUSH reg16	1	1	1	1	1	1	1	1	1	1	1	1	0	reg			1
PUSH sreg	0	0	0	sreg		1	1	0	—								2
REP INM	1	1	1	1	0	0	1	1	0	1	1	0	1	1	0	W	5 + 6 × rep
REP LDM/LDMB/LDMW	1	1	1	1	0	0	1	1	1	0	1	0	1	1	0	W	5 + 6 × rep
REP MOVBK	1	1	1	1	0	0	1	1	1	0	1	0	0	1	0	W	5 + 7 × rep
REP OUTM	1	1	1	1	0	0	1	1	0	1	1	0	1	1	1	W	5 + 6 × rep
REP STM/STMB/STMW	1	1	1	1	0	0	1	1	1	0	1	0	1	0	1	W	5 + 6 × rep
REPE CMPM/CMPMB/CMPMW	1	1	1	1	0	0	1	1	1	0	1	0	1	1	1	W	5 + 9 × rep
REPNE CMPM/CMPMB/CMPMW	1	1	1	1	0	0	1	0	1	0	1	0	1	1	1	W	5 + 9 × rep
REPNZ CMPBK/CMPBKB/CMPBKW	1	1	1	1	0	0	1	0	1	0	1	0	0	1	1	W	5 + 9 × rep
REPZ CMPBK/CMPBKB/CMPBKW	1	1	1	1	0	0	1	1	1	0	1	0	0	1	1	W	5 + 10 × rep
RET pop-value (segment-external call)	1	1	0	0	1	0	1	0	pop-value-low								9
	pop-value-high								—								
RET pop-value (segment-internal call)	1	1	0	0	0	0	1	0	pop-value-low								6
	pop-value-high								—								
RET (segment-external call)	1	1	0	0	1	0	1	1	—								8
RET (segment-internal call)	1	1	0	0	0	0	1	1	—								6
RETI	1	1	0	0	1	1	1	1	—								10
ROL mem, 1	1	1	0	1	0	0	0	W	mod		0	0	0	mem			3
	(disp-low)								(disp-high)								
ROL mem, CL	1	1	0	1	0	0	1	W	mod		0	0	0	mem			5
	(disp-low)								(disp-high)								
ROL mem, imm8	1	1	0	0	0	0	0	W	mod		0	0	0	mem			5
	(disp-low)								(disp-high)								
	imm8								—								
ROL reg, 1	1	1	0	1	0	0	0	W	1	1	0	0	0	reg			1
ROL reg, CL	1	1	0	1	0	0	1	W	1	1	0	0	0	reg			3
ROL reg, imm8	1	1	0	0	0	0	0	W	1	1	0	0	0	reg			3
	imm8								—								
ROLC mem, 1	1	1	0	1	0	0	0	W	mod		0	1	0	mem			3
	(disp-low)								(disp-high)								
ROLC mem, CL	1	1	0	1	0	0	1	W	mod		0	1	0	mem			5
	(disp-low)								(disp-high)								
ROLC mem, imm8	1	1	0	0	0	0	0	W	mod		0	1	0	mem			5
	(disp-low)								(disp-high)								
	imm8								—								
ROLC reg, 1	1	1	0	1	0	0	0	W	1	1	0	1	0	reg			1
ROLC reg, CL	1	1	0	1	0	0	1	W	1	1	0	1	0	reg			3

Table A-1. List of Instruction Execution Clock Counts (9/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
ROLC reg, imm8	1	1	0	0	0	0	0	W	1	1	0	1	0	reg			3
	imm8								—								
ROR mem, 1	1	1	0	1	0	0	0	W	mod		0	0	1	mem			3
	(disp-low)								(disp-high)								
ROR mem, CL	1	1	0	1	0	0	1	W	mod		0	0	1	mem			5
	(disp-low)								(disp-high)								
ROR mem, imm8	1	1	0	0	0	0	0	W	mod		0	0	1	mem			5
	(disp-low)								(disp-high)								
	imm8								—								
ROR reg, 1	1	1	0	1	0	0	0	W	1	1	0	0	1	reg			1
ROR reg, CL	1	1	0	1	0	0	1	W	1	1	0	0	1	reg			3
ROR reg, imm8	1	1	0	0	0	0	0	W	1	1	0	0	1	reg			3
	imm8								—								
RORC mem, 1	1	1	0	1	0	0	0	W	mod		0	1	1	mem			3
	(disp-low)								(disp-high)								
RORC mem, CL	1	1	0	1	0	0	1	W	mod		0	1	1	mem			5
	(disp-low)								(disp-high)								
RORC mem, imm8	1	1	0	0	0	0	0	W	mod		0	1	1	mem			5
	(disp-low)								(disp-high)								
	imm8								—								
RORC reg, 1	1	1	0	1	0	0	0	W	1	1	0	1	1	reg			1
RORC reg, CL	1	1	0	1	0	0	1	W	1	1	0	1	1	reg			3
RORC reg, imm8	1	1	0	0	0	0	0	W	1	1	0	1	1	reg			3
	imm8								—								
SET1 CY	1	1	1	1	1	0	0	1	—								4
SET1 DIR	1	1	1	1	1	1	0	1	—								4
SHL mem, 1	1	1	0	1	0	0	0	W	mod		1	0	0	mem			3
	(disp-low)								(disp-high)								
SHL mem, CL	1	1	0	1	0	0	1	W	mod		1	0	0	mem			5
	(disp-low)								(disp-high)								
SHL mem, imm8	1	1	0	0	0	0	0	W	mod		1	0	0	mem			5
	(disp-low)								(disp-high)								
	imm8								—								
SHL reg, 1	1	1	0	1	0	0	0	W	1	1	1	0	0	reg			1
SHL reg, CL	1	1	0	1	0	0	1	W	1	1	1	0	0	reg			3
SHL reg, imm8	1	1	0	0	0	0	0	W	1	1	1	0	0	reg			3
	imm8								—								
SHR mem, 1	1	1	0	1	0	0	0	W	mod		1	0	1	mem			3
	(disp-low)								(disp-high)								
SHR mem, CL	1	1	0	1	0	0	1	W	mod		1	0	1	mem			5
	(disp-low)								(disp-high)								

Table A-1. List of Instruction Execution Clock Counts (10/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
SHR mem, imm8	1	1	0	0	0	0	0	W	mod		1	0	1	mem			5
	(disp-low)								(disp-high)								
	imm8								—								
SHR reg, 1	1	1	0	1	0	0	0	W	1	1	1	0	1	reg			1
SHR reg, CL	1	1	0	1	0	0	1	W	1	1	1	0	1	reg			3
SHR reg, imm8	1	1	0	0	0	0	0	W	1	1	1	0	1	reg			3
	imm8								—								
SHRA mem, 1	1	1	0	1	0	0	0	W	mod		1	1	1	mem			3
	(disp-low)								(disp-high)								
SHRA mem, CL	1	1	0	1	0	0	1	W	mod		1	1	1	mem			5
	(disp-low)								(disp-high)								
SHRA mem, imm8	1	1	0	0	0	0	0	W	mod		1	1	1	mem			5
	(disp-low)								(disp-high)								
	imm8								—								
SHRA reg, 1	1	1	0	1	0	0	0	W	1	1	1	1	1	reg			1
SHRA reg, CL	1	1	0	1	0	0	1	W	1	1	1	1	1	reg			3
SHRA reg, imm8	1	1	0	0	0	0	0	W	1	1	1	1	1	reg			3
	imm8								—								
SS:	0	0	1	1	0	1	1	0	—								1
STM [DS1-spec:]dst-block	1	0	1	0	1	0	1	W	—								3
STMB	1	0	1	0	1	0	1	W	—								3
STMW	1	0	1	0	1	0	1	W	—								3
SUB acc, imm	0	0	1	0	1	1	0	W	imm8 or imm16-low								1
	imm16-high								—								
SUB mem, imm	1	0	0	0	0	0	s	W	mod		1	0	1	mem			3
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-lhigh								
SUB mem, reg	0	0	1	0	1	0	0	W	mod		reg			mem			3
	(disp-low)								(disp-high)								
SUB reg, imm	1	0	0	0	0	0	s	W	1	1	1	0	1	reg			1
	imm8 or imm16-low								imm16-high								
SUB reg, mem	0	0	1	0	1	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
SUB reg, reg'	0	0	1	0	1	0	1	W	1	1	reg			reg'			1
SUB reg, reg'	0	0	1	0	1	0	0	W	1	1	reg			reg'			1
SUBC acc, imm	0	0	0	1	1	1	0	W	imm8 or imm16-low								1
	imm16-high								—								
SUBC mem, imm	1	0	0	0	0	0	s	W	mod		0	1	1	mem			3
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-lhigh								

Table A-1. List of Instruction Execution Clock Counts (11/11)

Mnemonic, Operand	Operation Code																Clocks
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
SUBC mem, reg	0	0	0	1	1	0	0	W	mod		reg			mem			3
	(disp-low)								(disp-high)								
SUBC reg, imm	1	0	0	0	0	0	s	W	1	1	0	1	1	reg			1
	imm8 or imm16-low								imm16-high								
SUBC reg, mem	0	0	0	1	1	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
SUBC reg, reg'	0	0	0	1	1	0	1	W	1	1	reg			reg'			1
SUBC reg, reg'	0	0	0	1	1	0	0	W	1	1	reg			reg'			1
TEST acc, imm	1	0	1	0	1	0	0	W	imm8 or imm16-low								1
	imm16-high								—								
TEST mem, imm	1	1	1	1	0	1	1	W	mod		0	0	0	mem			2
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-lhigh								
TEST mem,reg	1	0	0	0	0	1	0	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
TEST reg, imm	1	1	1	1	0	1	1	W	1	1	0	0	0	reg			1
	imm8 or imm16-low								imm16-high								
TEST reg, mem	1	0	0	0	0	1	0	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
TEST reg, reg'	1	0	0	0	0	1	0	W	1	1	reg			reg'			1
TRANS	1	1	0	1	0	1	1	1	—								5
TRANS src-table	1	1	0	1	0	1	1	1	—								5
TRANSB	1	1	0	1	0	1	1	1	—								5
XCH AW, reg16	1	0	0	1	0	reg			—								3
XCH mem, reg	1	0	0	0	0	1	1	W	mod		reg			mem			5
	(disp-low)								(disp-high)								
XCH reg, mem	1	0	0	0	0	1	1	W	mod		reg			mem			5
	(disp-low)								(disp-high)								
XCH reg, reg'	1	0	0	0	0	1	1	W	1	1	reg			reg'			3
XCH reg16, AW	1	0	0	1	0	reg			—								3
XOR acc, imm	0	0	1	1	0	1	0	W	imm8 or imm16-low								1
	imm16-high								—								
XOR mem, imm	1	0	0	0	0	0	0	W	mod		1	1	0	mem			3
	(disp-low)								(disp-high)								
	imm8 or imm16-low								imm16-lhigh								
XOR mem, reg	0	0	1	1	0	0	0	W	mod		reg			mem			3
	(disp-low)								(disp-high)								
XOR reg, imm	1	0	0	0	0	0	0	W	1	1	1	1	0	reg			1
	imm8 or imm16-low								imm16-high								
XOR reg, mem	0	0	1	1	0	0	1	W	mod		reg			mem			2
	(disp-low)								(disp-high)								
XOR reg, reg'	0	0	1	1	0	0	1	W	1	1	reg			reg'			1
XOR reg, reg'	0	0	1	1	0	0	0	W	1	1	reg			reg'			1

APPENDIX B INDEX

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: ______________________________

Document number: ______________________ Page number: ____________

____

____

____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑