NEC

User's Manual

V30MX™

CPU CORE

Document No. A11897EJ1V0UM00 (1st edition)
Date Published April 1997 N

[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.

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.

M7 96.5

PREFACE

The V30MX is an ASIC original CPU core with bus efficiency improved by separating addresses from the data bus of the μPD70116H (V30HL^{TM}), an NEC general-purpose microcomputer.

Since the V30MX has the same instruction set as that of the V30HL, existing programs can be used without changes.

Read this document thoroughly in order to carry out LSI design successfully. Be sure to observe the points that are described in the manual (general points, cautions and restrictions). Failing to observe them may cause deterioration of the quality and abnormalities in operation of the LSI product.

Readers : This manual is intended for users who have an understanding of the V30MX functions and wish to design an application system using the V30MX functions.

Purpose : This manual is intended to help users understand the V30MX hardware functions and has the following configuration.

Configuration: This V30MX User's Manual consists of a hardware volume (this manual) and an instruction volume.

This manual	Instruction
Pin functions	Outline of instructions
CPU functions	Description of instructions
Bus control functions	
Interrupt functions	
Standby functions	
Reset functions	
Test functions	

How to Read this Manual: This manual assumes that you have a general understanding of electric circuits, logical circuits and microcomputers.

To have a general understanding of the V30MX functions
→ Read it according to the table of contents.

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

Legend:

Active-low notation	:	×××B (B at the end of pin name, signal name)
Memory map address	:	Upper side – higher address, lower side – lower address
Note	:	Explanation of item marked with “Note” in the text
Caution	:	Item to be especially noted
Remark	:	Supplementary information
Numeric notations	:	Binary ... ××× or ×××B Decimal ... ××× Hexadecimal ...×××H

Related documents:

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

- **User’s Manual – Hardware**

V30MX User’s Manual	: This document
V20HL™, V30HL, User’s Manual – Hardware	: IEM-1124A

- **User’s Manual - Instruction**

16-Bit V Series User’s Manual – Instruction	: U11301E

When designing, contact your NEC sales representative or domestic sales agent to make sure you are using the latest documentations.

TABLE OF CONTENTS

LIST OF FIGURES

Fig. No. **Title, Page**

LIST OF TABLES

Table No. **Title, Page**

CHAPTER 1 GENERAL DESCRIPTION

The V30MX core is a CPU core which separates addresses from the data bus of the μPD70116H (V30HL), an NEC general-purpose microcomputer, improving bus efficiency and CPI (Cycle per Instruction) by 70 % compared with the V30HL. It also incorporates new registers that support the LIM EMS4.0.

The V30MX core has a complete static circuit configuration, which facilitates standby and clock stop and provides low power consumption.

Since the V30MX and V30HL use a common instruction set, existing programs can be used without changes.

1.1 Features

(1) A complete static circuit configuration facilitates standby and clock stop.

(2) Low power consumption

(3) Incorporates registers that support LIM EMS4.0.

(4) Address space

- Memory : 1 M bytes (16 M bytes when using EMS)
- I/O : 64 K bytes

(5) A variety of memory addressing modes

(6) 14 × 16-bit register set

(7) 101-command of instruction set (completely compatible with μPD70116H)

- Bit field manipulation instruction : Data transfer between bit field of bits 1 to 16 of memory and accumulator
- Packed BCD operation instruction : Addition, subtraction and comparison of 1 to 255-digit BCD string
- Bit manipulation instruction : Set, clear, inversion, and test of arbitrary bit of 8/16-bit register and memory

(8) High-speed effective address calculation

1.2 Outline of Differences from V30HL

(1) In the V30HL, addresses and data are multiplexed and use the same pins. In the V30MX, they are separate pins. In addition, the address bus has been extended to 24 bits. Because of this, the V30MX has a bus timing different from the V30HL.

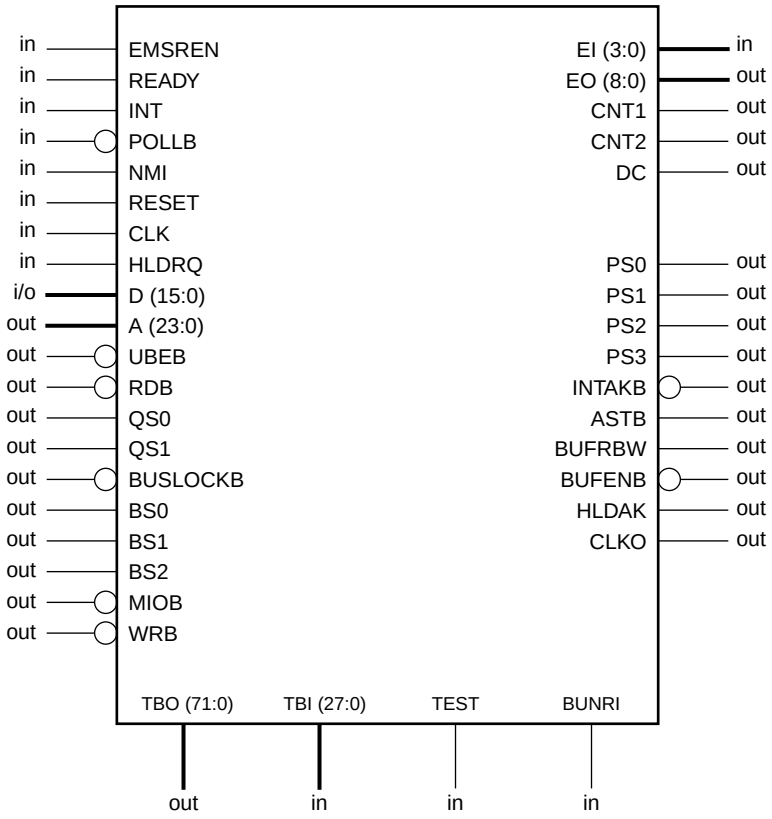
(2) The V30MX is not provided with the function of switching between a small scale and large scale as in the V30HL. Signals like INTAK (OS1) that are output by switching between small and large scales in the V30HL are output from separate pins in the V30MX.

(3) The following pins exist in the V30HL, but not in the V30MX.

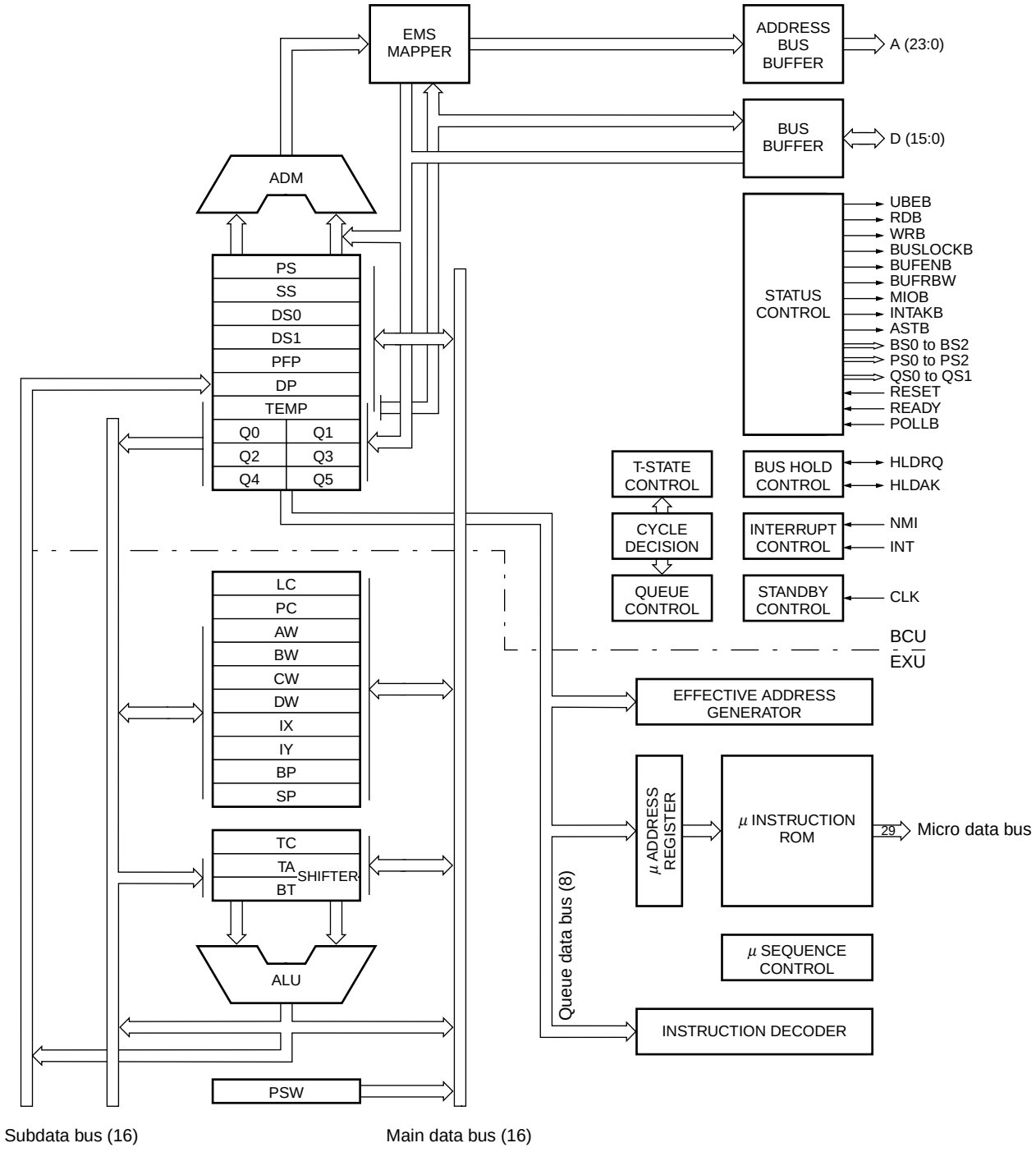
- RQ/AK1, RQ/AK0, S/LG

(4) The V30HL has a μPD8080AF emulation function, but the V30MX has no such function.

1.3 Symbol Diagram



1.4 Internal Block Diagram



[MEMO]

CHAPTER 2 PIN FUNCTIONS

This chapter describes the names and functions of the V30MX core pins.

2.1 Pin List

Pin Name	Input/Output	Function	Reference Item
A (23:0)	OZ	Address	2.2 (1)
D (15:0)	I/O	Data	2.2 (2)
UBEB	OZ	Upper Byte Enable	2.2 (3)
RDB	OZ	Read	2.2 (4)
READY	I	Bus Ready	2.2 (5)
INT	I	Interrupt Request	2.2 (6)
POLLB	I	Test	2.2 (7)
NMI	I	Non Maskable Interrupt	2.2 (8)
RESET	I	Reset	2.2 (9)
CLK	I	Clock	2.2 (10)
QS0, QS1	O	Queue Status	2.2 (11)
BUSLOCKB	OZ	Lock	2.2 (12)
BS2 to BS0	OZ	Bus Cycle Status	2.2 (13)
MIOB	OZ	I/O Memory Select	2.2 (14)
PS0, PS1	O	Processor Status (Segment Register)	2.2 (15)
PS2	O	Processor Status (PSW: interrupt)	2.2 (16)
PS3	O	Processor Status (Native/8080 emulation)	2.2 (17)
WRB	OZ	Write	2.2 (18)
INTAKB	O	Interrupt Acknowledge	2.2 (19)
ASTB	O	Address Strobe	2.2 (20)
BUFRBW	OZ	Data Transmit Receive	2.2 (21)
BUFENB	OZ	Data Enable	2.2 (22)
HLDRQ	I	Hold Request	2.2 (23)
HLDAK	O	Hold Acknolwedge	2.2 (24)
EMSREN	I	EMS Registers Access Enable	2.2 (25)
CNT1, CNT2	O	3-state pin control	2.2 (26)
DC	O	D15 to D0 control	2.2 (27)
BUNRI	I	Test Bus control	2.2 (28)
TEST	I	Test Bus control	2.2 (29)
TBI (27:0)	I	Test Bus (Input)	2.2 (30)
TBO (71:0)	OZ	Test Bus (Output)	2.2 (31)
EI (3:0)	I	NEC Reserved	2.2 (32)
EO (8:0)	O	NEC Reserved	2.2 (33)
CLKO	O	NEC Reserved	2.2 (34)

Remark The symbols in the input/output field are as follows:
I/O : Bi-directional pin
I : Input pin
O : Output (dedicated) pin
OZ : 3-state output pin

2.2 Description of Pin Functions

(1) A (23:0) (3-state output) address

24-bit address bus.

The address is output a half clock before the bus cycle. Normally, only A (19:0) are used for output and A (23:20) are low. When the EMS function is used, these pins are all used as the outputs for all bits A (23:0).

(2) D (15:0) (3-state input/output) data

16-bit data bus.

Go to high impedance during hold acknowledge and interrupt acknowledge.

(3) UBEB (3-state output) upper byte enable

Indicates that the higher bits of D (15:0) (D (15:8)) are used in Tc of the bus cycle.

In the V30MX, memory and I/O are accessed separately for byte data banks which are accessed by an even address (A0 = 0) and for byte data banks which are accessed by an odd address (UBEB = 0) as shown in the table below.

Remark The V30MX executes one bus cycle in at least 2 clocks. The first clock is called T_S, and the next clock T_C. When a wait is inserted, T_C is repeated until the next bus cycle is started.

Operand		UBEB	A0	Number of Bus Cycles
Address of even word		0	0	1
Word at odd address	1st bus cycle	0	1	2
	2nd bus cycle	0	0	
Byte at even address		1	0	1
Byte at odd address		1	1	1

The UBEB signal is driven low continuously during interrupt acknowledge (word access at an even address is necessary due to a vector read).

This pin goes to high impedance during hold acknowledge, but because it incorporates a latch (level hold circuit) it holds the previous value until it is driven externally. In standby mode, it is fixed at high level output.

(4) RDB (3-state output) read

Outputs a signal which becomes active (low level) during a data read from memory. I/O and memory are distinguished by MIOB.

This pin goes to high impedance during hold acknowledge, but because it incorporates a latch (level hold circuit) it holds the previous value until it is driven externally. In standby mode, it is fixed at high level output.

(5) READY (input) bus read

Controls waits.

It is sampled at the end of each T_C cycle, that is, on a rise of CLK. When memory or I/O cannot complete a data read/write operation, if the READY signal is driven high, the V30MX generates a T_C cycle and so the read/write cycle can be extended. When the READY signal is low, it goes to the next bus cycle.

Caution If the READY signal does not satisfy the setup time or hold time, correct operation is not guaranteed, so establish synchronization using an external circuit.

(6) INT (input) interrupt request

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

This signal is detected in the last clock cycle of an instruction, and if the interrupt is enabled (interrupt enable flag (IE) is set (1)) it is acknowledged.

The external device checks whether the INT interrupt request has been acknowledged by the INTAKB signal output from the V30MX. Therefore, keep the INT signal high until the first INTAKB signal is output.

The priority order of interrupt request signals is as follows.

INT < NMI < HLDRQ

For example, when an INT interrupt and NMI interrupt are generated simultaneously, the NMI interrupt takes precedence and the INT interrupt is not acknowledged. A hold request can be acknowledged even during INT acknowledge.

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

(7) POLLB (input) test

This is used to synchronize between execution of the V30MX program and operation of an external device.

POLLB is checked by the POLL instruction, and if it is low, the next instruction is started, and if it is high, POLLB input is checked for each clock cycle until it is driven low.

(8) NMI (input)

Inputs an interrupt request signal which cannot be masked 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.

Keep the NMI signal high for at least 5 clock 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.

(9) RESET (input) reset

Inputs a reset signal.

A RESET signal takes precedence over all operations, and after reset release the program at memory address FFFF0H starts (segment value: FFFFH, offset value: 0H).

Keep the RESET input pin active (high) for at least 4 clock cycles.

Remark The standby mode can also be released by a RESET signal.

(10) CLK (input) clock

External clock input.

The V30MX does not divide clocks internally. Therefore, input to this CLK pin and internal operation are performed at the same frequency. When this CLK input is stopped, the STOP mode is entered. The STOP mode can help to reduce power consumption drastically.

Caution Stopping CLK input while RESET input is active (high) does not satisfy the power supply current specification. Be sure to stop CLK input when RESET input is inactive (low).

(11) QS0, QS1 (output) queue status

This is a status signal that notifies an instruction queue signal to off-chip.

The instruction queue status means a status in which the execution unit (EXU) accesses an instruction queue. Signals QS0 and QS1 are only valid in one clock cycle immediately after this queue access. The table below shows the relationship between QS0, QS1 and instruction queue status.

QS0	QS1	Instruction Queue Status
0	0	No operation
1	0	Fetch of 1st byte of instruction
0	1	Queue is empty.
1	1	Fetch of 2nd and subsequent bytes of instruction

(12) BUSLOCKB (3-state output) lock

During execution of one instruction following the BUSLOCK instruction, it outputs a signal requesting the other master CPU of the multi-processor system not to use the system bus. It also outputs the signal during interrupt acknowledge.

(13) BS0 to BS2 (3-state output) bus cycle status

(14) MIOB (3-state output) I/O memory select

BS0, BS1, BS2 and MIOB indicate bus statuses as shown in the table below. BS2 of the V30MX has the same output values as COD/INTA in the i80286.

BS2	MIOB	BS1	BS0	Bus Cycle
0	0	0	0	Interrupt acknowledge
0	0	0	1	No signal with this combination is generated.
0	0	1	0	No signal with this combination is generated.
0	0	1	1	Idle state
0	1	0	0	HALT state
0	1	0	1	Memory data read
0	1	1	0	Memory data write
0	1	1	1	Idle state
1	0	0	0	No signal with this combination is generated.
1	0	0	1	I/O read
1	0	1	0	I/O write
1	0	1	1	Idle state
1	1	0	0	No signal with this combination is generated.
1	1	0	1	Code fetch
1	1	1	0	No signal with this combination is generated.
1	1	1	1	Idle state

(15) PS0, PS1 (output) processor status (segment register)

In the V30HL, addresses and processor statuses can be output at any time on a time sharing basis from A16/PS0 to A19/PS3, while in the V30MX they are output from different pins.

The PS0 and PS1 signals indicate which segment is currently used. The table below shows the relationship between PS0, PS1 and the segment.

PS0	PS1	Segment
0	0	Data segment 1
1	0	Stack segment
0	1	Program segment
1	1	Data segment 0

(16) PS2 (output) processor status (PSW: interrupt)

The PS2 signal indicates the content of the interrupt enable flag (IE) of the program status word (PSW).

The table below shows the relationship between PS2 and IE.

PS2	IE Flag (PSW)	Status
0	0	INT interrupt enabled
1	1	INT interrupt disabled

(17) PS3 (output) processor status (native/8080 emulation)

This signal always outputs a low level.

The V30MX is not provided with the μPD8080AF emulation function.

(18) WRB (3-state output) write

Outputs a signal which becomes active (low) in a data write to I/O or memory.

I/O and memory are distinguished by MIOB.

This pin goes to high impedance during hold acknowledge, but since it incorporates a latch (level hold circuit), it holds the previous state until it is driven externally. In standby mode, it is fixed at high level output.

(19) INTAKB (output) interrupt acknowledge

Outputs an interrupt acknowledge signal.
The INTAKB signal is output when an INT signal is acknowledged, and the external device inputs the interrupt vector to the V30MX via the data bus in synchronization with this signal.

(20) ASTB (output) address strobe

Outputs a strobe signal to latch address information to off-chip. It is not activated during HALT status output (remains low).

(21) BUFRBW (3-state output) data transmit receive

Outputs a signal to determine the data transfer direction of the external bi-directional buffer.

The BUFRBW signal indicates transmission to the external device when it is high, and reception from the external device when it is low.

This pin goes to high impedance during hold acknowledge, but since it incorporates a latch (level hold circuit), it holds the previous state until it is driven externally. In standby mode, it is fixed at low level output.

(22) BUFENB (3-state output) data enable

Outputs an output enable signal for the external bi-directional buffer.

The BUFENB signal is output during data transfer from/to memory or I/O or during interrupt vector input.

This pin goes to high impedance during hold acknowledge, but since it incorporates a latch (level hold circuit), it holds the previous state until it is driven externally. In standby mode, it is fixed at high level output.

(23) HLDRQ (input) hold request

Inputs a signal from the external device to request the V30MX to release the address bus, data bus and control bus (bus hold).

The priority order among interrupt request signals and HLDRQ signal is as follows.

INT < NMI < HLDRQ

Caution Since normal operation is not guaranteed unless the HLDRQ signal satisfies the setup time, establish synchronization using an external circuit.

(24) HLDAK (output) hold acknowledge

Outputs an acknowledge signal which indicates that a bus hold request signal (HLDRQ) has been acknowledged. While the HLDAK signal is active (high), the address bus, data bus, and 3-state output control bus go to high impedance.

(25) EMSREN (input) EMS registers access enable

Enables/disables access to the I/O mapped register for the EMS function control. When EMSREN is high, access to the I/O mapped register is enabled. Access is also enabled to the same external I/O address as the I/O address assigned to the register.

Caution EMSREN can be changed only after a reset. At other timings the operation of the V30MX is not guaranteed.

(26) CNT1, CNT2 (output) 3-state pin control

(27) DC (output) D (15:0) control

CNT1 is used for control of the 3-state pins A (23:0), BS2, MIOB, UBEB, BUSLOCKB, RDB, WRB, BUFENB, and BUFRBW.

CNT2 is used for direction control of 3-state pins BS1 and BS0.

DC is used for direction control of input/output pins D (15:0).

The relationship between respective pins and control signals is shown below.

Pin to be Controlled	CNT1	Status
A (23:0), BS2, MIOB, UBEB, BUSLOCKB, RDB, WRB, BUFENB, BUFRBW	H	High impedance
	L	Output active

Pin to be Controlled	CNT2	Status
BS1, BS0	H	High impedance
	L	Output active

Pin to be Controlled	DC	Status
D (15:0)	H	Input mode
	L	Output mode

(28) BUNRI (input) test bus control

(29) TEST (input) test bus control

(30) TBI (27:0) (input) test bus (input)

(31) TBO (71:0) (3-state output) test bus (output)

These pins are used for mega-function separation tests using the test bus.

For details of using these pins, refer to **CHAPTER 8 TEST FUNCTIONS**.

(32) EI (3:0) (input) NEC reserved

These pins are reserved for NEC.

Be sure to use F091 to input the levels shown in the table below.

EI0	H
EI1	H
EI2	H
EI3	L

(33) EO (8:0) (output) NEC reserved

These pins are reserved for NEC.

Leave these pins open.

(34) CLKO (output) NEC reserved

These pins are reserved for NEC.

Leave these pins open.

2.3 Pin Information in Specific Status

Pin	Normal Mode			Test Mode	
	During HOLD	During HALT	After Reset	Standby (BUNRI = 1) (TEST = 0)	Unit Test (BUNRI = 1) (TEST = 1)
A (23:0)	Hi-Z	Continuous	Hi-Z	Hi-Z	Hi-Z
D (15:0)	Hi-Z	Continuous	Hi-Z	Hi-Z	Hi-Z
UBEB	Hi-Z	H	Hi-Z	Continuous	X
RDB	Hi-Z	H	Hi-Z	Continuous	X
QS0, QS1	L	L	L	Continuous	X
BUSLOCKB	Hi-Z	H[Note]	Hi-Z	Continuous	X
BS2 to BS0	Hi-Z	L	Hi-Z	Continuous	X
MIOB	Hi-Z	H	Hi-Z	Continuous	X
PS0	Continuous	Continuous	L	Continuous	X
PS1	Continuous	Continuous	H	Continuous	X
PS2	Continuous	Continuous	L	Continuous	X
PS3	Continuous	Continuous	L	Continuous	X
WRB	Hi-Z	H	Hi-Z	Continuous	X
INTAKB	H	H	H	Continuous	X
ASTB	L	L	L	Continuous	X
BUFRBW	Hi-Z	Continuous	Hi-Z	Continuous	X
BUFENB	Hi-Z	H	Hi-Z	Continuous	X
HLDAK	H	L	L	Continuous	X
DC	H	Continuous	H	Continuous	X
CNT1, CNT2	H	Continuous	H	Continuous	X
TBO (71:0)	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Output

Note If a BUSLOCK instruction is executed before a HALT instruction, a low level is output.

Remarks **1.** “Continuous” in the table indicates that the previous operation is continued.
2. “Standby” in the test mode refers to the standby test mode (refer to **CHAPTER 8 TEST FUNCTIONS**).
3. X : Don’t care

[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 change
- AL : Byte multiplication/division, byte input/output, BCD rotate, data conversion
- AH : Byte multiplication/division
- BW : Data conversion
- CW : Loop control branch, repeat, prefix
- CL : Shift instruction, rotation instruction, BCD operation
- DW : Word multiplication/division, indirect addressing input

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

The V30MX 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 **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 which the execution unit (EXU) is about to execute.

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

Furthermore, in execution of a branch instruction, call instruction, return instruction and break 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.

Status flag

Carry flag (CY)	Refer to **(a)**.
Parity flag (P)	Refer to **(b)**.
Auxiliary carry flag (AC)	Refer to **(c)**.
Zero flag (Z)	Refer to **(d)**.
Sign flag (S)	Refer to **(e)**.
Overflow flag (V)	Refer to **(f)**.

Control flag

Break flag (BRK)	Refer to **(g)**.
Interrupt enable flag (IE)	Refer to **(h)**.
Direction flag (DIR)	Refer to **(i)**.
Mode flag (MD)	Refer to **(j)**.

The status flag is automatically set (1) and reset (0) according to the execution result (data value) of each instruction.

The CY flag can directly be set/reset or inverted by an instruction.
The control flag is set/reset by an instruction and controls the operation of the V30MX.

The IE flag and BRK flag are reset when interrupt servicing is started.

RESET input resets (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).

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 0 to 7 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 and restored by a return instruction (RETI, RETM).

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 reset otherwise.

In the case of word operation, CY is set when there is a carry or borrow from operation result bit 15, and reset otherwise. It is not changed by an increment or decrement instruction.

< 2 > Logical operation

CY is reset without regard to the operation result.

< 3 > Binary multiplication

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

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

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

CY is reset 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 reset 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 reset 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 reset 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 reset 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 reset 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 reset when it is 0 in the case of a byte operation.

Set when bit 15 of the result is 1 and reset 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 reset when they are the same in the case of a byte operation.

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

< 2 > Binary multiplication

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

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

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

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

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

< 3 > Binary division

Reset.

< 4 > Logical operation

Reset.

< 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: Reset
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: Reset
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 an 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 INT interrupt is enabled. It is reset by a DI instruction and the INT interrupt is disabled.

(i) Direction flag (DIR)

Set by a SET1 DIR instruction and reset by a CLR1 DIR instruction.

When the DIR flag is set, processing is carried out from the higher addresses to the lower addresses in block transfer/input/output type instructions. When it is reset, processing is carried out from the lower addresses to the higher addresses.

(j) Mode flag (MD)

This is a μPD8080AF emulation function related flag which conforms to the previous V30HL. Since the V30MX 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
Base register in variable length bit field manipulation instruction
Address register for source operand in BCD string operation instruction

IY: Address register for destination operand in block data manipulation
Base register in variable length bit field manipulation
Address register for destination operand in BCD string operation instruction

3.2 Address Space

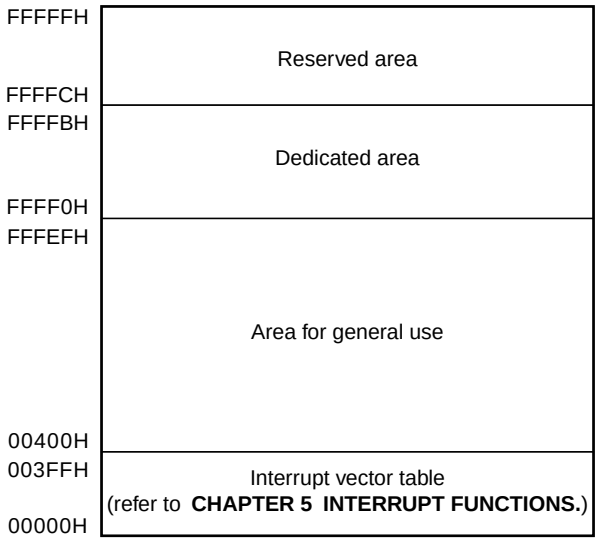
3.2.1 Memory space

< 1 > When low level is input to EMSREN pin (when EMS is not used)

The V30MX uses 20-bit address information and can access 1 M (512 K word) of memory. The start address after a reset is FFFF0H.

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

Figure 3-1. Memory Map (when EMSREN = 0)



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)[Note]. Word data in the V30MX 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 the memory space access method.

Note This excludes the case where the address of the area in which the interrupt start address (interrupt vector table) is stored is always even (A0 = 0).

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

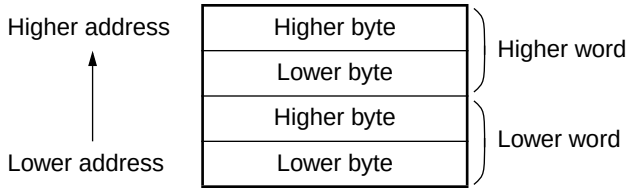
Table 3-1. Address and Data Configuration of Each Memory Element

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 data configuration are as follows.

Word data configuration

Double word data configuration



< 2 > When high level is input to EMSREN pin (when EMS is used)

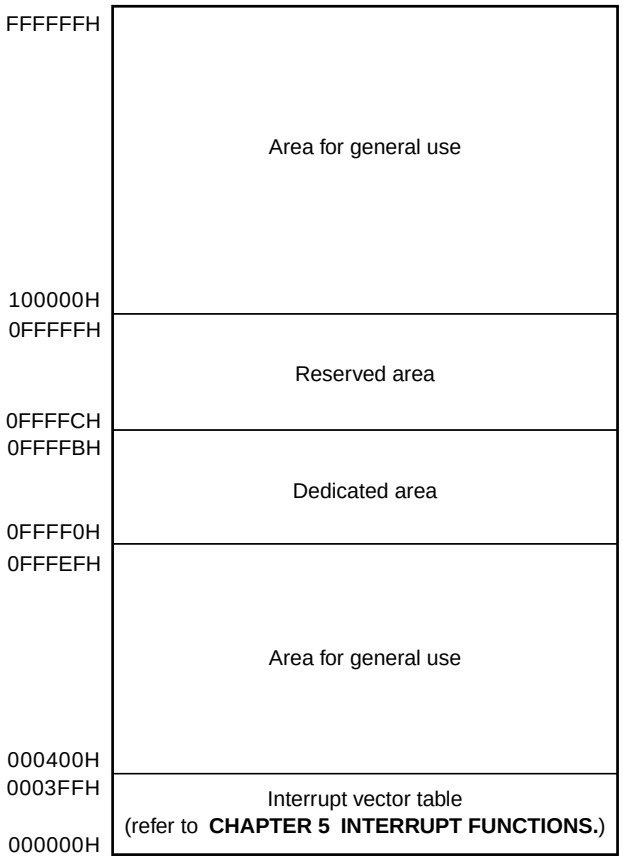
When EMS is used, the address space becomes 16 M (24 bits).

The mapping sources for memory extension are 80000H to 9FFFFH (fixed), C0000H to CFFFFH, D0000H to DFFFFH and E0000H to EFFFFH (selected from these ranges). The mapping destinations are determined by setting the EMS data register (refer to **3.9 EMS Functions**).

The area 000000H to 0FFFFFH is the same as the case where EMS is not used.

The start address after a reset is 0FFFF0H whether the EMS function is used or not.

Figure 3-2. Memory Map (when EMSREN = 1)



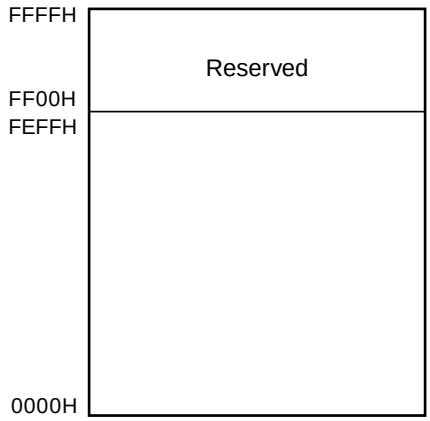
3.2.2 I/O space

The V30MX 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 3-3 and Figure 3-4 show I/O maps. The 256 bytes of FF00H to FFFFH are reserved for future use and are not available to users.

< 1 > When low level is input to EMSREN pin (when EMS is not used)

Figure 3-3. I/O Map (when EMSREN = 0)

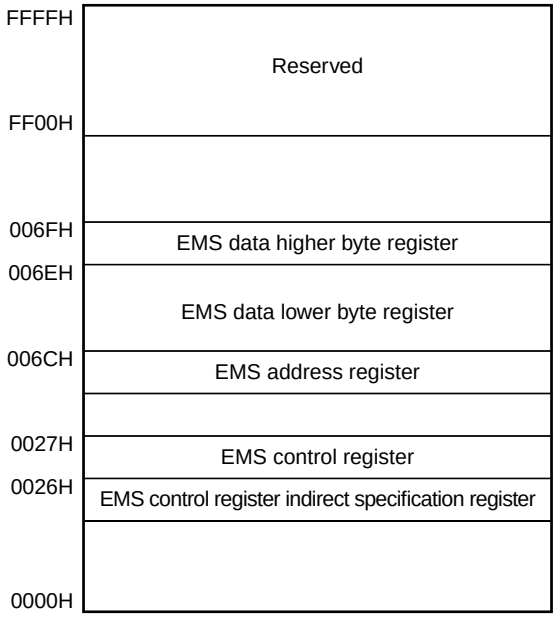


< 2 > When high level is input to EMSREN pin (when EMS is used)

Each register for EMS control is placed in the I/O space and is accessed by an input/output instruction. Writing 01H to the EMS control register indirect specification register (0026H) beforehand allows the EMS control register to be accessed. All the 8 bits of the EMS control register indirect specification register are valid and initialized to 00H after a reset.

For the I/O addresses of these registers, only the lower 10 bits are decoded and the higher 6 bits are not decoded. Therefore, in software it operates so that the same internal I/O exists in multiple addresses.

Figure 3-4. I/O Map (when EMSREN = 1)



3.3 Internal Block Functions

The V30MX can be divided roughly into 2 processing units; the execution unit (EXU) and bus control unit (BCU). Each unit operates asynchronously, independent of the other, thus improving bus utilization efficiency and achieving high-speed execution of instructions.

3.3.1 Bus control unit (BCU)

The BCU has the following functions.

- **Instruction prefetch**

  Performs prefetch of instructions using the prefetch queue (Q5 to Q0).

  Prefetch means pre-loading of the next instruction to be executed while the bus is free during execution of an instruction.

  The BCU of the V30MX carries out prefetch automatically when there are a lot of free spaces in the queue.

- **Control of external address/data bus**

  Drives the bus in response to a request from the EXU.

- **Control of internal/external control functions**

  Controls the interrupt function, bus hold function and standby function.

The function of each BCU block is as follows.

(1) Prefetch pointer (PFP)

This is a 16-bit binary counter to hold the offset information of the program memory address that the BCU is about to prefetch in the instruction queue.

The PFP is incremented (+1) every time an instruction byte is prefetched from the program memory. Furthermore, in execution of a branch, call, return or break instruction, a new location is loaded and the PFP value becomes the same as that of the program counter (PC).

The PFP is always used together with the program segment (PS).

(2) Prefetch queue (Q5 to Q0)

Provided with 6-byte instruction queue (FIFO), it can store a maximum of 6 bytes of an operation code that is prefetched by the BCU.

The operation code stored in the queue is fetched and executed by the EXU.

In execution of a branch, call, return or break instruction or servicing of an external interrupt, the content of the queue is cleared and an instruction at a new location is prefetched.

Normally, a prefetch is carried out when there is a free space of one word (2 bytes) or more. If the average execution time of instructions that are executed consecutively is greater than the number of clocks necessary to prefetch each operation code to some degree, operation codes that can be executed by the EXU are ready in the queue every time the EXU completes execution of one instruction, and it is possible to omit the time for fetch from memory from the instruction execution time. As a result, this improves the processing speed compared to other CPUs that fetch and execute one instruction at a time.

Caution In the following cases, the effect of the queue is reduced.

- **When there are many instructions in which the queues are cleared as with execution of branch instructions**
- **When there are a series of instructions with short execution time**

(3) Data pointer (DP)

This is a 16-bit register pointing to read/write addresses of variables.

The register contents including the effective addresses generated by the effective address generator (EAG) and offset values of the memory addresses are transferred to the DP.

(4) Communication temporary register (TEMP)

This is a 16-bit temporary register for communications between the external data bus and EXU.
Since TEMP uses byte accesses, it can read/write the higher byte and lower byte separately.

Basically, the EXU ends a write operation by carrying out data transfer to TEMP and ends a read operation after conforming that data is transferred from the external data bus to TEMP.

(5) Address modification circuit (ADM)

Performs generation of physical addresses (addition of segment register and prefetch pointer (PFP) or data pointer (DP)) and increments (+1) of the PFP.

(6) Bus buffer

Performs signal output to the external address bus and signal input/output with the external data bus.

(7) Status controller

Determines the internal state and bus state, transmits them to each internal block and generates signals output to off-chip.

(8) Bus hold controller

When there is a signal requesting bus release from off-chip, it controls each bus corresponding to the signal.

(9) Interrupt controller

Carries out servicing necessary for an interrupt request from off-chip.

(10) Standby controller

In standby mode, supplies a clock only to the circuits related to the function necessary to release the bus hold function and standby mode and stops other functions.

Furthermore, performs control when the system returns to the normal state if necessary.

3.3.2 Execution unit (EXU)

The EXU decodes instructions prefetched by the BCU and executes them by microprogram control. The function of each EXU block is as follows.

(1) Arithmetic and logical unit (ALU)

Consists of a full adder and logical operation circuit and executes arithmetic operations (addition/subtraction, multiplication/division, increment/decrement, auxiliary operations) and logical operations (test, AND, OR, XOR and bit-wise testing, set, clear, inversion).

(2) Temporary register/shifter A/B (TA/TB)

This is a 16-bit temporary register/shifter used for multiplication/division and shift/rotate (including BCD rotate) instructions.

In execution of multiplication/division instructions, operates as a TA+TB 32-bit temporary register/shifter, and in execution of shift/rotate instructions, only TB operates as a 16-bit temporary register/shifter.

Both TA and TB can read/write separately the higher byte and lower byte from/to the internal bus.

TA/TB holds ALU data.

(3) Temporary register C (TC)

This is a 16-bit temporary register used for multiplication/division and other internal processing.

TC holds ALU data.

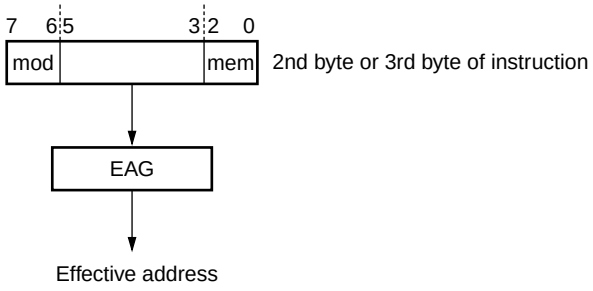
(4) Loop counter (LC)

This is a 16-bit register which counts the number of loops of primitive block transfer and input/output instructions (MOVBK, OUTM, etc.) controlled by repeat prefix instructions (REP, REPC, etc.), the number of shifts of multi-bit shift/rotate instructions.

(5) Effective address generator (EAG)

This is a circuit which performs high-speed calculation of effective addresses necessary in memory accesses. Ends calculations in 2 clocks for all addressing modes.

Figure 3-5. Generation of Effective Address



Fetches the byte (2nd or 3rd byte) specifying an instruction operand, generates a control signal about ALC and corresponding register manipulation if a memory access is required, and calculates the effective address and transfers it to the data pointer (DP).

Furthermore, performs control requesting startup of a bus cycle (memory read) to the BCU if necessary.

(6) Instruction decoder

Classifies the 1st byte of an operation code into a group with a specific function and holds it during execution of a micro instruction.

(7) Micro address register

Specifies the ROM address of the next micro instruction to be executed.
At the start of execution of a micro instruction, fetches the 1st byte of an instruction stored in the queue as the start address to this register and specifies the start address of the prescribed micro instruction sequence.

(8) Micro instruction ROM

Holds 1024 words of a 29-bit micro instruction.

(9) Micro instruction sequence circuit

Controls micro address registers, outputs of the micro instruction ROM and synchronization between the EXU and BCU.

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 V30MX 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.

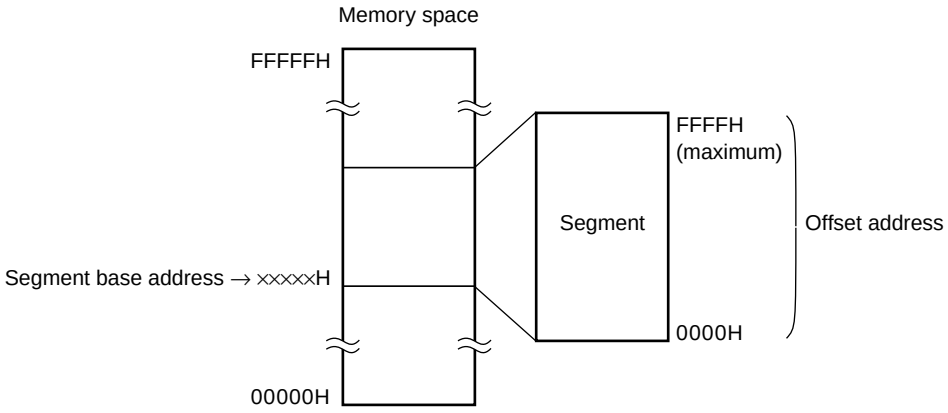
Physical addresses cannot be controlled directly in program creation in machine language. The V30MX specifies memory addresses in a segment system.

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.

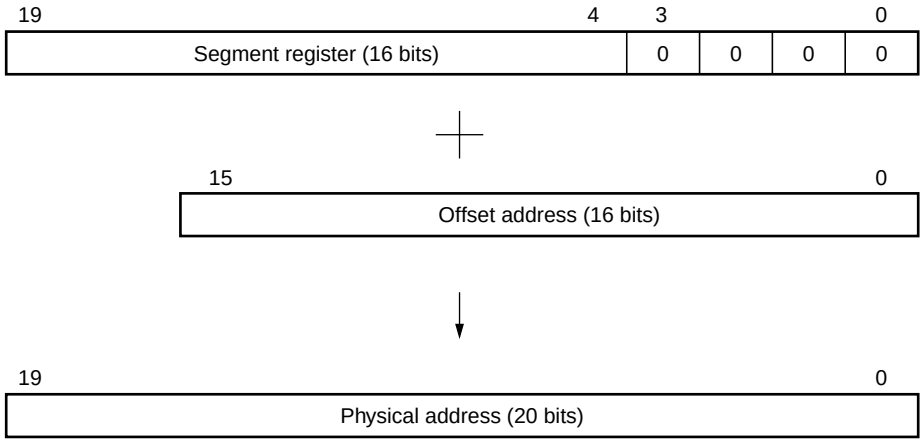
Figure 3-6. Conceptual Diagram of Segment System



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 3-7 shows the relationship between the segment register, offset address and physical address.

Figure 3-7. Relationship between Segment Register, Offset Address and Physical Address



As shown in Figure 3-7, 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 V30MX 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	Overwrite
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[Note]		
IY in instruction group B[Note]	DS1	Disabled

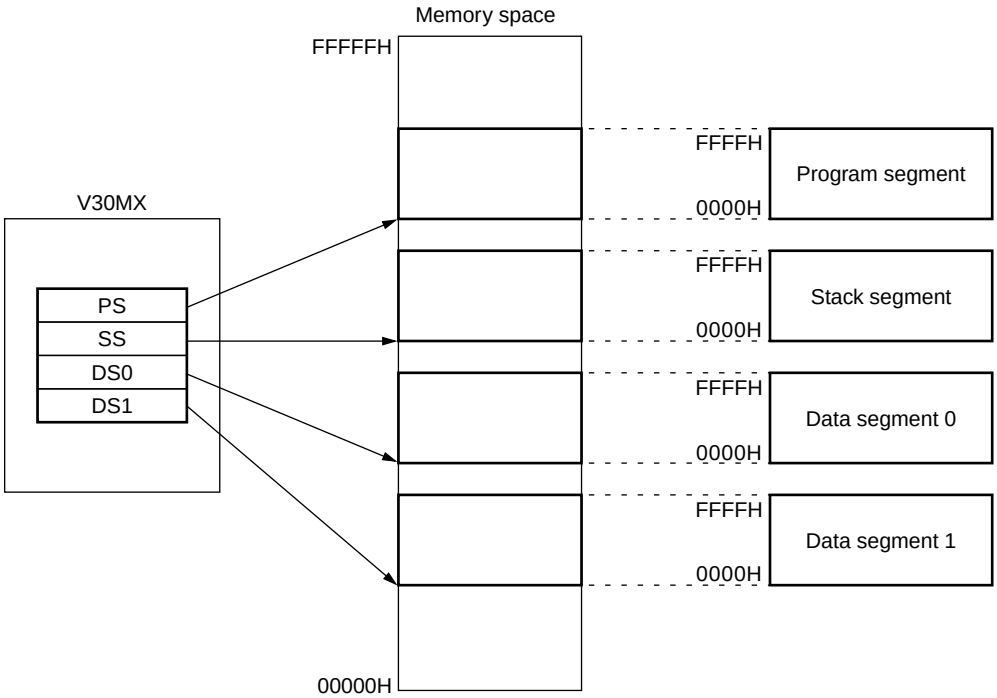
Note Instruction group A: Primitive block transfer instruction, primitive output instruction, BCD string instruction, EXT instruction
Instruction group B: Primitive block transfer instruction, primitive input instruction, BCD string instruction, INS instruction

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 3-8 shows the relationship between each segment register, segment and memory space.

Figure 3-8. Relationship between Each Segment Register, Segment and Memory Space



(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

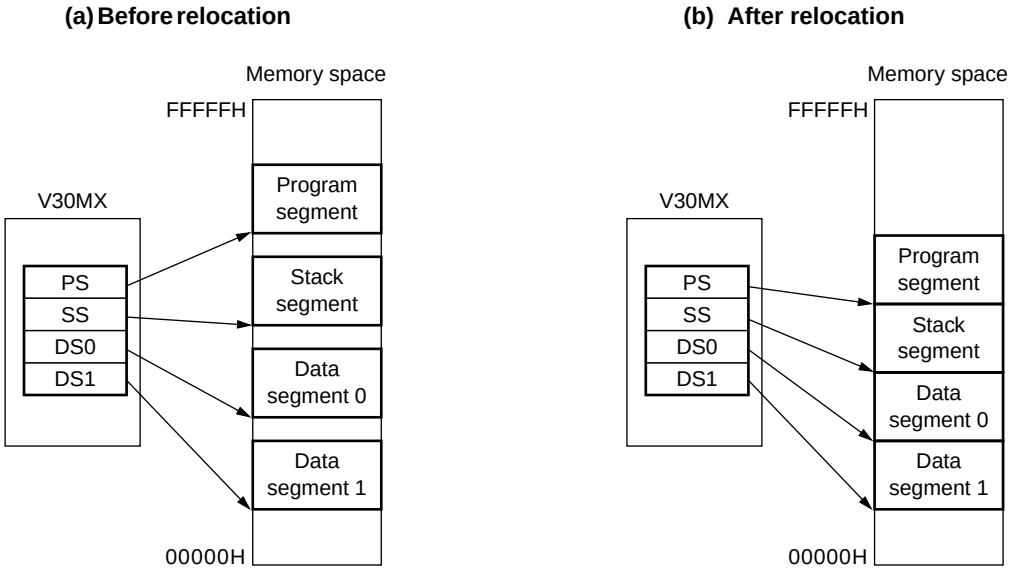
Relocating programs that are stored in two or more files separately in empty memory spaces for each execution is called dynamic relocation.

Figure 3-9 shows a conceptual diagram of dynamic relocation.

For the V30MX, 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 3-9. Dynamic Relocation



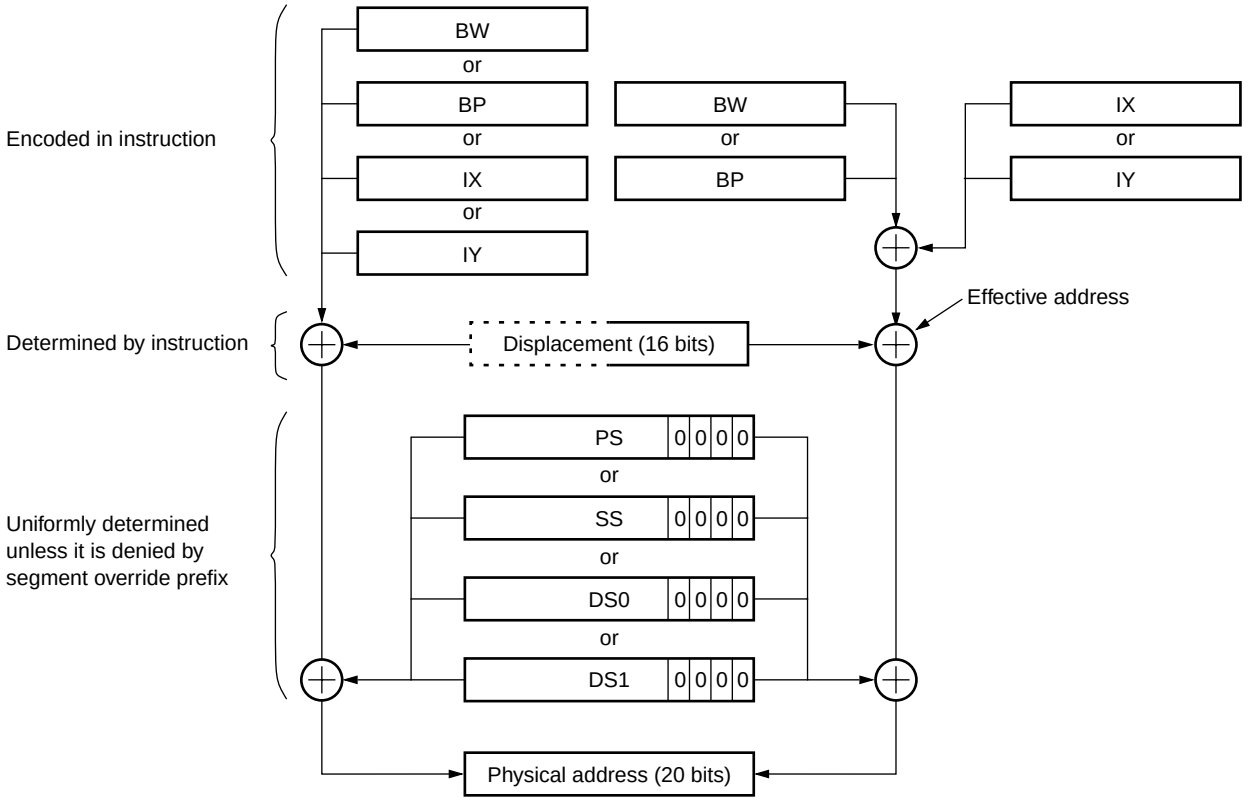
3.5 Effective Address

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.

The method of calculation of EA is shown below. Figure 3-10 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 3-10. Memory Address Calculation



3.6 Instruction Set

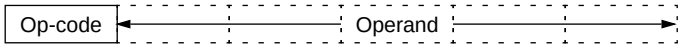
3.6.1 List of instruction sets by function

The V30MX instructions consist of 101 instructions in its instruction set and functionally these are roughly classified into the following 27 kinds.

Refer to the **16-Bit V Series User's Manual – Instruction** for details of each instruction. The number of clocks is shown in **APPENDIX LIST OF NUMBER OF INSTRUCTION EXECUTION CLOCKS.**

Instruction Group	Mnemonic
Data transfer instruction	LDEA, MOV, TRANS, TRANSB, XCH
Repeat prefix	REP, REPC, REPE, REPNC, REPNE, REPNZ, REPZ
Primitive block transfer instruction	CMPBK, CMPBKB, CMPBKW, CMPM, CMPMB, CMPMW, LDM, LDMB, LDMW, MOVBK, MOVBKB, MOVBKW, STM, STMB, STMW
Bit field manipulation instruction	EXT, INS
Input/output instruction	IN, OUT
Primitive input/output instruction	INM, OUTM
Addition/subtraction instruction	ADD, ADDC, SUB, SUBC
BCD operation instruction	ADD4S, CMP4S, ROL4, ROR4, SUB4S
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, NOT1, SET1, TEST1
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, BRKEM, BRKV, CHKIND, RETI
CPU control instruction	BUSLOCK, DI, EI, FPO1, FPO2, HALT, NOP, POLL
Segment override prefix	DS0:, DS1:, PS:, SS:
Emulation mode dedicated instruction	CALLN, RETEM

3.6.2 Format after instruction



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
                                     BNZ   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 **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 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.

(I) 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
```

(II) DW

The content of 16-bit register 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 which 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 [IX+2], 0    ; wordmem (offset (word_var)+IY+2) ← 0
SUB  AW, [IX+6]            ; AW ← AW – wordmem (BP+IX+6)
```

(d) Addressing with based index

Indicates the memory address at which the value of the 16-bit base register (IX 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 which has a 2-dimensional array structure, etc.

Example of usage:

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

(3) Bit addressing

Bit addressing specifies 1-bit data in the 8/16-bit register or 8/16-bit memory.

In bit addressing, data is specified by an operand which specifies a register or memory and the 2nd operand of the bit offset which specifies any 1 bit within it. For the operand which specifies a register or memory, any of register addressing or memory addressing can be used, and the following three kinds of bit offset can be used.

Description	1 Bit Specified
imm3	1-bit data in byte register/memory
imm4	1-bit data in word register/memory
CL	1-bit data in byte/word register/memory

For the bit offset, only the lower 3 bits are valid in the case of byte register/memory and only the lower 4 bits are valid in the case of word register/memory and their ranges are 0 to 7 and 0 to 15, respectively.

Example of usage:

```
TEST1   word ptr [BW+2], 5   ; Check bit 5 of wordmem (BW+2).
CLR1      byte var, CL       ; Set bit 1 specified by CL of bytemem (offset (byte_var)) to 0.
```

3.8 Faster Execution of Instructions

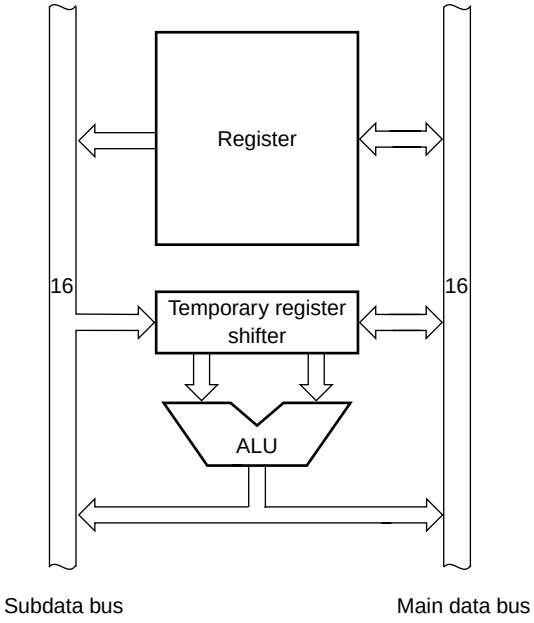
The V30MX is provided with the following hardware to shorten instruction execution time. For details of each function, refer to **3.1.4 Program counter (PC)** and **3.3.1 (1) Prefetch pointer (PFP)**, and **3.3.2 Execution unit (EXU)**.

- EXU internal dual data bus
- Effective address generator
- Temporary register/shifter (TA/TB)
- Loop counter (LC) and prefetch pointer (PFP)

3.8.1 Dual data bus system

In order to reduce the number of processing steps necessary for instruction execution, a dual data bus system composed of the main data bus (16 bits) and subdata bus (16 bits) is used. Use of this system reduces processing time for addition/subtraction, logical operation, compare instruction, etc., by approximately 30 % compared with the single bus system.

Figure 3-11. Dual Data Bus System



Example: ADD AW,BW:AW ← AW+BW

		Single bus	Dual bus
Step	1	ALU ← AW	ALU ← AW, BW
	2	ALU ← BW	AW ← ALU
	3	AW ← ALU	

3.8.2 Effective address generator (EAG)

This is a circuit to calculate an effective address necessary for memory accessing (refer to **3.5 Effective Address**) at high speed. In the microprogram system, 5 to 12 clocks were needed for calculation of an effective address. However, with this dedicated hardware only 2 clocks are needed in all addressing modes, achieving operation several times faster.

3.8.3 Temporary register/shifter A and B (TA, TB)

Temporary register/shifter A and B (TA, TB) are available for multiplication/division and shift/rotate instructions.

Use of this circuit increases the speed of execution, especially, of multiplication/division instructions approximately 4 times that of the microprogram system.

- TA+TB : 32-bit temporary register/shifter (for multiplication/division instructions)
- TB : 16-bit temporary register/shifter (for shift/rotate instruction)

3.8.4 Loop counter (LC)

Counts the number of loops of the primitive block transfer/input/output instructions controlled by the repeat prefix and the number of shifts of multi-bit shift/rotate instructions.

For example, in the case of multi-bit rotate of a register, the LC is as follows and the speed is doubled in average compared to the microprogram system.

```
RORC  AW,CL;CL = 5

Microprogram system    LC system
8 + 4 × 5 = 28 clocks  7 + 5 = 12 clocks
```

3.8.5 Program counter (PC) and prefetch pointer (PFP)

When performing a prefetch, provision of hardware for both the PFP which addresses the program memory and the PC which addresses the program memory currently to be executed shortens the instruction execution time of branch, call, return break instructions by the time corresponding to several blocks compared to hardware with only one PFP.

3.9 EMS Functions

The V30MX supports EMS memory of a maximum of 16 M bytes. When EMS is used, the higher 6 bits of the address are extended to 10 bits, totaling to 24 bits. This means mapping in 16 K-byte units. The lower 14 bits of the address are used without conversion irrespective of whether the EMS function is used or not.

The mapping source of 16 K bytes in this memory space is called the physical page and the mapping destination of 16 K bytes in the EMS space is called logical page. Furthermore, an aggregate of physical pages is called page frame.

In the case of the V30MX, addresses of the logical pages corresponding to 12 physical pages can be set. Therefore, there will be no problem if we assume that the physical page and page frame have the same meaning. The subsequent pages will explain the page frame as also representing physical page.

Of the 12 page frames, 8 page frames are fixed at 80000H to 9FFFFH in the memory space and other 4 page frames are allocated to 64 K bytes at one of C0000H to CFFFFH, D0000H to DFFFFH, or E0000H to EFFFFH.

The 12 page frames can be mapped on any logical pages in the EMS space (16 K bytes: however, the lower 16 bits are allocated to one of 0000H to 3FFFH, 4000H to 7FFFH, 8000H to BFFFH or C000H to FFFFH) in the EMS space.

3.9.1 EMS control registers

The EMS control registers are allocated to the I/O space and accessed by input/output instructions. However, when accessing the EMS control register (0027H), write 01H to I/O address 0026H beforehand.

Register	I/O Byte	Reference Item
EMS control indirect specification register	0026H	(1)
EMS control register	0027H	(2)
EMS address register	006CH	(5)
EMS data register (lower byte)	006EH	(3)
EMS data register (higher byte)	006FH	(4)

(1) EMS control indirect specification register (IDX) [I/O address: 0026H]

This is an indirect specification register to enable accesses to the EMS control register.

When any value other than 01H is set in this register, the EMS control register cannot be accessed (the I/O space other than the core is accessed).

After a reset, this register is initialized to 00H. When accessing the EMS control register after a reset, set 01H in this register first. In this register, all 8 bits are valid and any value can be set.

(2) EMS control register (EC) [I/O address: 0027H]

EMSPF1 and EMSPF0 perform address setting of 4-page page frames for which addresses can be selected.

EMSEN specifies collectively whether the EMS function for full page frames is enabled or not. If this bit disables EMS, settings of all other EMS related registers are disabled and normal memory accesses are performed.

The EMS control indirect specification register and this register cannot be set by an word access all together at a time. Be sure to set 01H in the EMS control indirect specification register by a byte access and then access this register.

Figure 3-12. EMS Control Register (EC)

7	6	5	4	3	2	1	0
EMSPF1	EMSPF0	0	EMSEN	0	0	0	0

EMSEN	EMS Function Enable/Disable
0	EMS disabled
1	EMS enabled

EMSPF1	EMSPF0	Page Frame Address
0	0	C0000H to CFFFFH
0	1	D0000H to DFFFFH
1	0	E0000H to EFFFFH
1	1	Setting prohibited

(3) EMS data register (lower byte) (EDL0 to EDL11) [I/O address: 006EH]

(4) EMS data register (higher byte) (EDH0 to EDH11) [I/O address: 006FH]

For EMSA23 to EMSA14, the higher 10 bits of the logical page corresponding to each page frame are specified.

EMSPEN enables/disables mapping of each page frame onto the EMS space.

Setting 1 will convert a memory access for the corresponding page frame to the EMS address set by EMSA23 to EMSA14.

Setting 0 will not perform conversion.

This register is accessed by an input/output instruction by indirectly specifying the higher 6 bits of the page frame address in the EMS address register.

Figure 3-13. EMS Data Register (EDL11 to EDL0, EDH11 to EDH0)

(a) EDL11 to EDL0

7	6	5	4	3	2	1	0
EMSA21	EMSA20	EMSA19	EMSA18	EMSA17	EMSA16	EMSA15	EMSA14

EMSA21 to EMSA14	21st to 14th bits of logical page address

(b) EDH11 to EDH0

7	6	5	4	3	2	1	0
EMSPEN	0	0	0	0	0	EMSA23	EMSA22

EMSA23 to EMSA22	23rd and 22nd bits of logical page address

EMSPEN	Enable/disable EMS function of each page frame
0	Disabled
1	Enabled

(5) EMS address register (EA) [I/O address: 006CH]

This register is used to indirectly specify 12 pairs of EMS data registers.

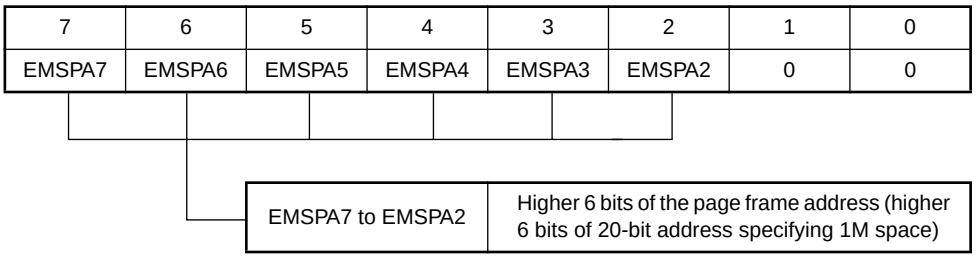
Specifies the higher 6 bits of the address of each page frame.

In other words, the value set in this register is one of the following 12 values.

- 8 values: 80H, 84H, 88H, 8CH, 90H, 94H, 98H, 9CH
- Any 4 values below depending on the values of EMSPF1 and EMSPF2

  → 4 values: C0H, C4H, C8H, CCH
  → 4 values: D0H, D4H, D8H, DCH
  → 4 values: 0DH, E4H, E8H, ECH

Figure 3-14. EMS Address Register (EMSPA7 to EMSPA2)



3.9.2 Caution on accessing EMS control registers

In an access cycle to the above EMS control registers, the control signals in the table below remain inactive as follows. This access is always performed with 0 waits without regard to input of a READY signal.

Pin	Value when Accessing EMS Control Register
RDB	H (inactive)
WRB	
BUFENB	
BUFRBW	
UBEB	

3.9.3 EMS setting example

The following example illustrates a case where the EMS page frame is set only at C0000H to CFFFFH and this page frame is mapped on to 120000H to 12FFFFH of the physical memory.

```
; Selection of EMS control register
; This setting can be done only the first one time.

mov     al, 01h
out     26h, al

; Setting of EMS control register
; EMS function enabled, page frame selected

mov     al, 00010000b
out     27h, al

; Page frame 80000H to 9FFFFH is unused.

mov     dl, 80h             ; DL = 80H, 84H, 88H, 8CH, 90H, 94H, 98H, 9CH
mov     cx, 8a
```

```
inhph:

        ; Set the page frame number (address) that you want to select to EMS address register.
        ; EMS data register is set to indirect address via 006E to 006FH.

        mov     al, dl
        out     6ch, al

        ; Write 00H to EMS data higher byte register (register corresponding to address set to 006CH)
        ; and disable EMS function.

        xor     al, al
        out     6fh, al

        ; Add 04H to DL register and calculate page frame number (address) to be selected next.
        ; Ends when CX register is set to 0.

        add     al, 04h
        loop    inhph

        ; Setting of page frame C0000H to C3FFFFH

        mov     al, 0c0h
        out     6ch, al             ;indirect specification of C0000H

        ; Setting to physical address 120000H

        mov     ax, 8048h           ; EMS enabled, bit23 to bit14 at 120000H
        out     6eh, ax             ; EMS data register (word access)

        ; Setting of page frame C4000H to C7FFFH

        mov     al, 0c4h
        out     6ch, al             ; Indirect specification of C4000H

        ; Setting to physical address 124000H

        mov     ax, 8049h           ; EMS enabled, bit23 to bit14 at 124000H
        out     6eh, ax             ; EMS data register (word access)

        ; Setting of page frame C8000H to CBFFFH

        mov     al, 0c8h
        out  6ch, al                ; Indirect specification of C8000H

        ; Setting of physical address 128000H

        mov     ax, 804ah           ; EMS enabled, bit23 to bit14 at 128000H
        out     6eh, ax             ; EMS data register (word access)

        ; Setting of page frame CC000H to CFFFFH

        mov     al, 0cch
        out     6ch, al             ; Indirect specification of CC000H
```

```
; Setting of physical address 12C000H

mov     ax, 804bh       ; EMS enabled, bit23 to bit14 at 12C000H
out     6eh, ax         ; EMS data register (word access)
```

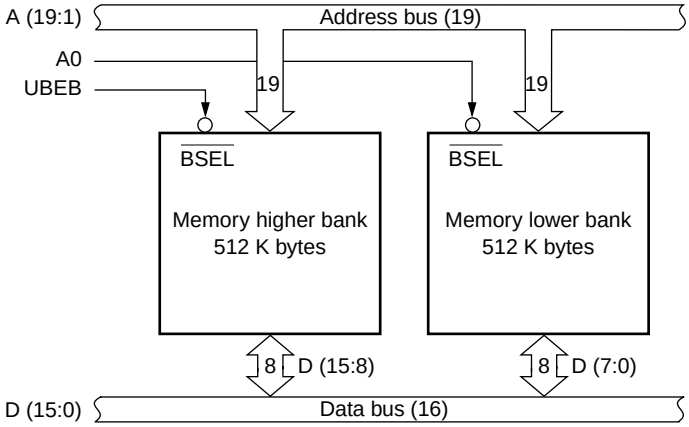
CHAPTER 4 BUS CONTROL FUNCTIONS

4.1 Interface between V30MX and Memory

As the V30MX 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 V30MX and memory.

Figure 4-1. Interface between V30MX 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.

- **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 (A (15:0)) of the address information is incremented (+1). That is, A0 = 0 is set, and the lower byte at the next address is accessed.

- **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, UBEB, A0 and number of bus cycles.

Table 4-1. V30MX Data Access

Operand		UBEB	A0	Number of Bus Cycles
Word at even address		0	0	1
Word at odd address	**Note 1**	0	1	2
	Note 2	1	0	
Byte at even address		1	0	1
Byte at odd address		0	1	1

Notes 1. 1st bus cycle
2. 2nd bus cycle

Normally, the V30MX 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 V30MX, 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 2 clocks. Therefore, every time word data at an odd address is accessed, two extra clocks of the instruction execution time are required compared to accessing word data at an even address. This applies when executing an instruction which 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: Execution time of MOV reg, mem instruction (number of clocks)

Byte data: 9 :
Word data: 11 : For odd address
9 : For even address

This is an example in which one word data access is performed.

4.2 Accessing I/O Space

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 (A (19:16)) of the address bus.

Data can be transferred between the V30MX and I/O in both 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. For example, using a bit manipulation instruction for the memory, it is possible to test (decide 1 or 0), set (1), clear (0), or invert one line of an I/O port.

Caution However, with the memory mapped I/O, all control signals output from the V30MX 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 Timing of Memory and I/O

The V30MX executes one bus cycle in at least 2 clocks. The first clock is called T_S, and the next clock T_C.

The V30MX performs pipelined addressing.

< 1 > Outputs an address 1/2 clock earlier than T_S of the relevant cycle, that is, in synchronization with a clock fall. At the same time, it outputs MIOB and BS2 indicating the memory R/W or I/O RW cycle.

< 2 > In synchronization with a clock rise which is the start of T_S, the address strobe signal ASTB changes to a high level, while BS0 retains a high level. At the same time it outputs UBEB and BUSLOCKB. These UBEB and BUSLOCKB retain those values until the bus cycle ends.

< 3 > In synchronization with a clock rise of T_S, the address strobe signal ASTB changes to a high level.

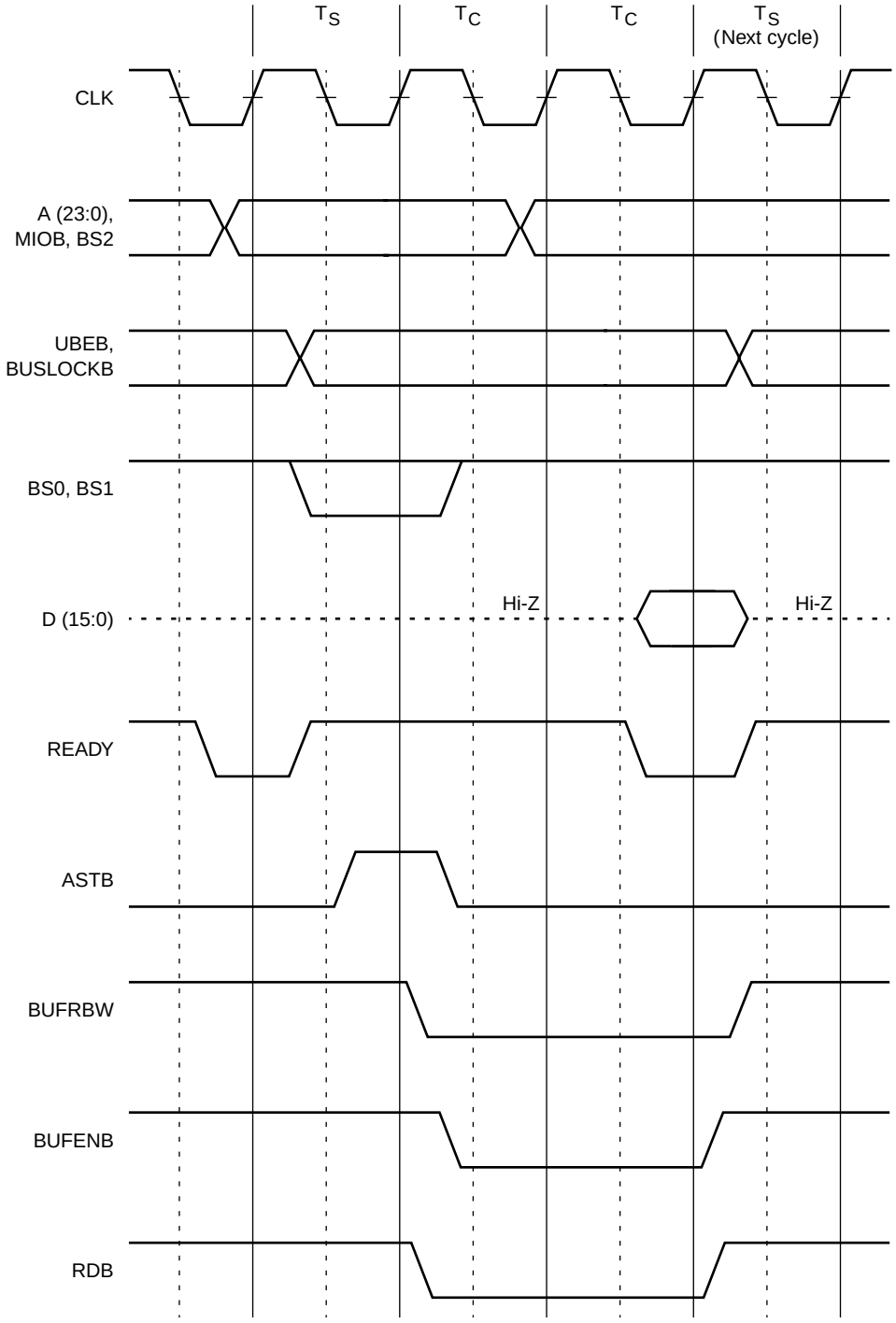
< 4 > T_S ends in 1 clock and T_C starts. ASTB returns to a low level in synchronization with a clock rise which is the start of T_C. On the other hand, the RDB signal which requests the peripheral device to output data and the BUFENB and BUFRBW signals change to a low level to avoid collision between the RDB signal requesting the peripheral device to output data and a signal on the data bus.

< 5 > In synchronization with a fall of the first T_C clock, the address for the next cycle, MIOB and BS2 are output.

< 6 > At the end of T_C in synchronization with a clock rise, the READY input is sampled. If READY is low, the bus cycle is terminated and data is fetched. RDB, BUFENB and BUFRBW change to a high level. If READY is high, the bus cycle continues and T_C is repeated.

At this time, RDB, BUFENB and BUFRBW retain low levels.

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



4.4 Write Timing of Memory and I/O

< 1 > Like the read cycle, outputs an address 1/2 clock earlier than T_S of the relevant cycle, that is, in synchronization with a clock fall. At the same time, it outputs MIOB and BS2 indicating the memory R/W or I/O RW cycle.

< 2 > In synchronization with a clock rise which is the start of T_S, BS0 changes to a low level, while BS1 retains a high level (indicating a write cycle). At the same time it outputs UBEB and BUSLOCKB. These UBEB and BUSLOCKB retain those values until the bus cycle ends.

< 3 > In synchronization with a clock rise of T_S, the address strobe signal ASTB changes to a high level. Output of data to be written is started and BUFENB changes to a low level. BUFRBW retains a high level in the write cycle.

< 4 > T_S ends in 1 clock and T_C starts. ASTB returns to a low level in synchronization with a clock rise which is the start of T_C. On the other hand, the WRB signal which requests the peripheral device to input data changes to a low level.

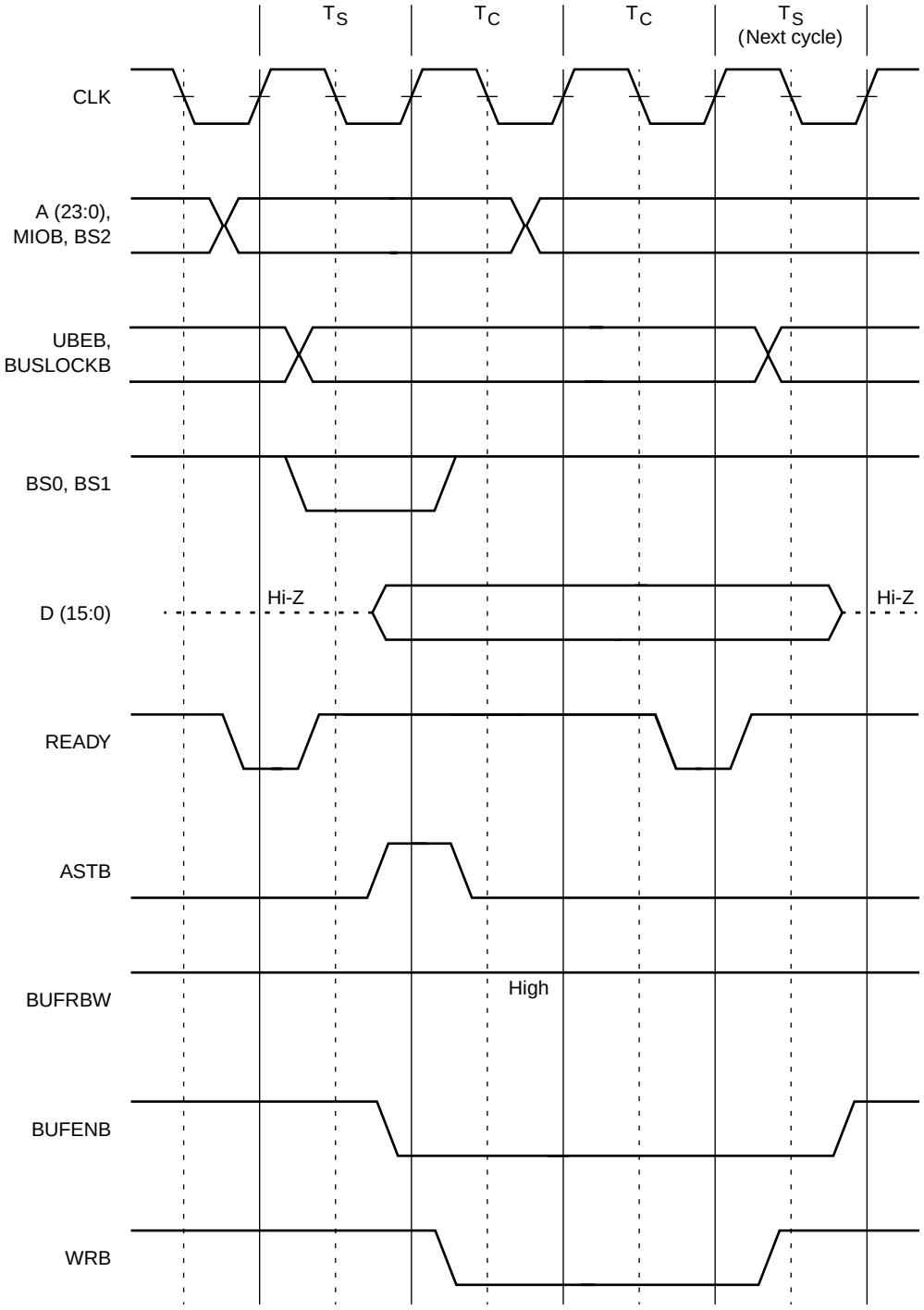
< 5 > In synchronization with a fall of the first T_C clock, the address for the next cycle, MIOB and BS2 are output.

< 6 > At the end of T_C in synchronization with a clock rise, the READY input is sampled. If READY is low, the bus cycle is terminated and WRB changes to a high level indicating the peripheral device of the data fetch timing. If READY is high, the bus cycle continues and T_C is repeated.

At this time, WRB and BUFENB retain low levels.

< 7 > In <6>, a low-level READY signal is sampled and in synchronization with a clock rise 1/2 clock later the data output ends. BUFENB changes to a high level and the bus cycle ends.

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



4.5 Bus Hold Function

When HLDRQ becomes active high, an HLDAK signal is output.

While the HLDAK signal is active, it makes the address bus, data bus and some control signals go to high impedance and hands over control of the bus cycle related signals to the device which requested a hold (refer to **(24)** in **2.2 Description of Pin Functions**).

Figure 4-4. Bus Hold Timing (Write Operation → Bus Hold State)

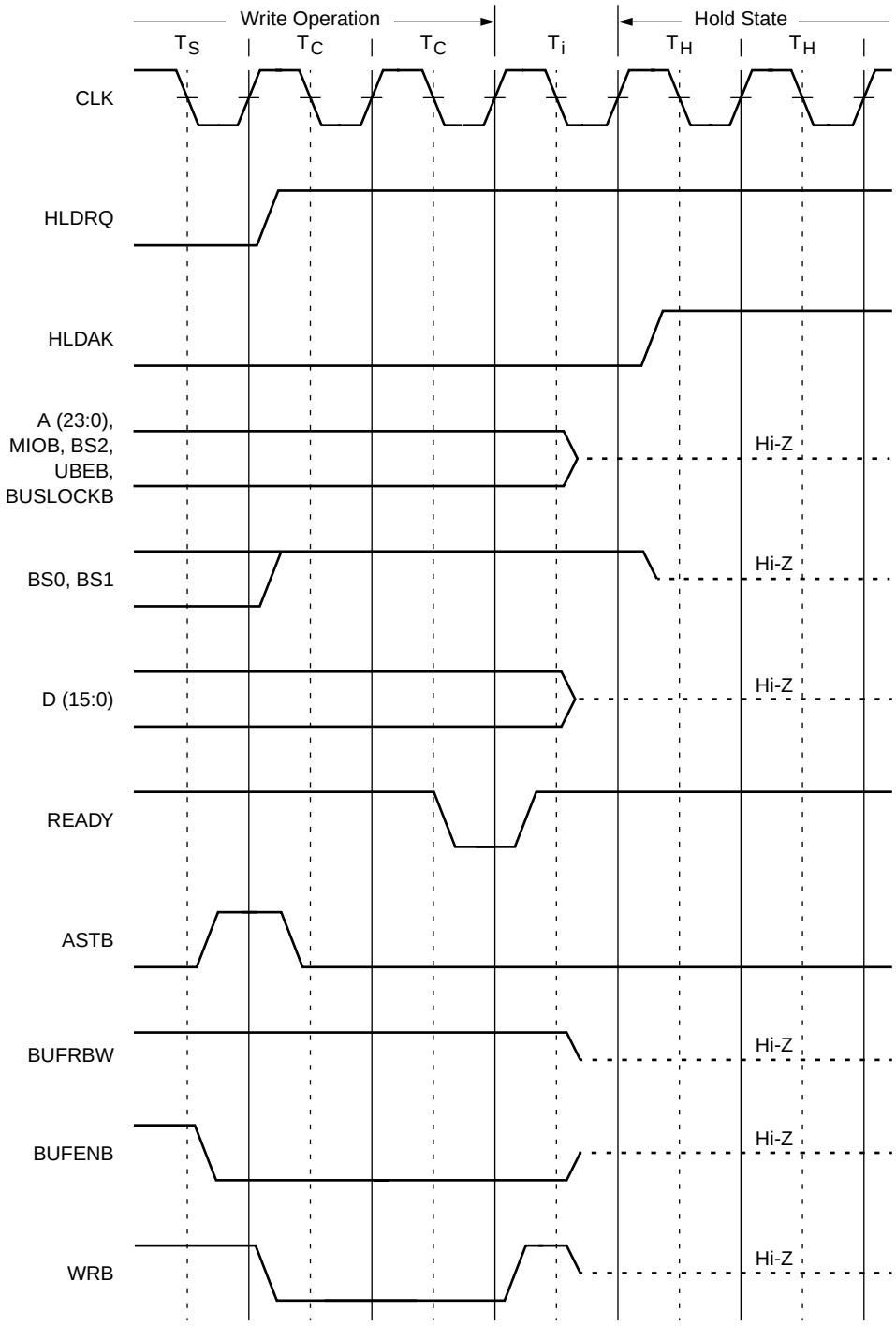
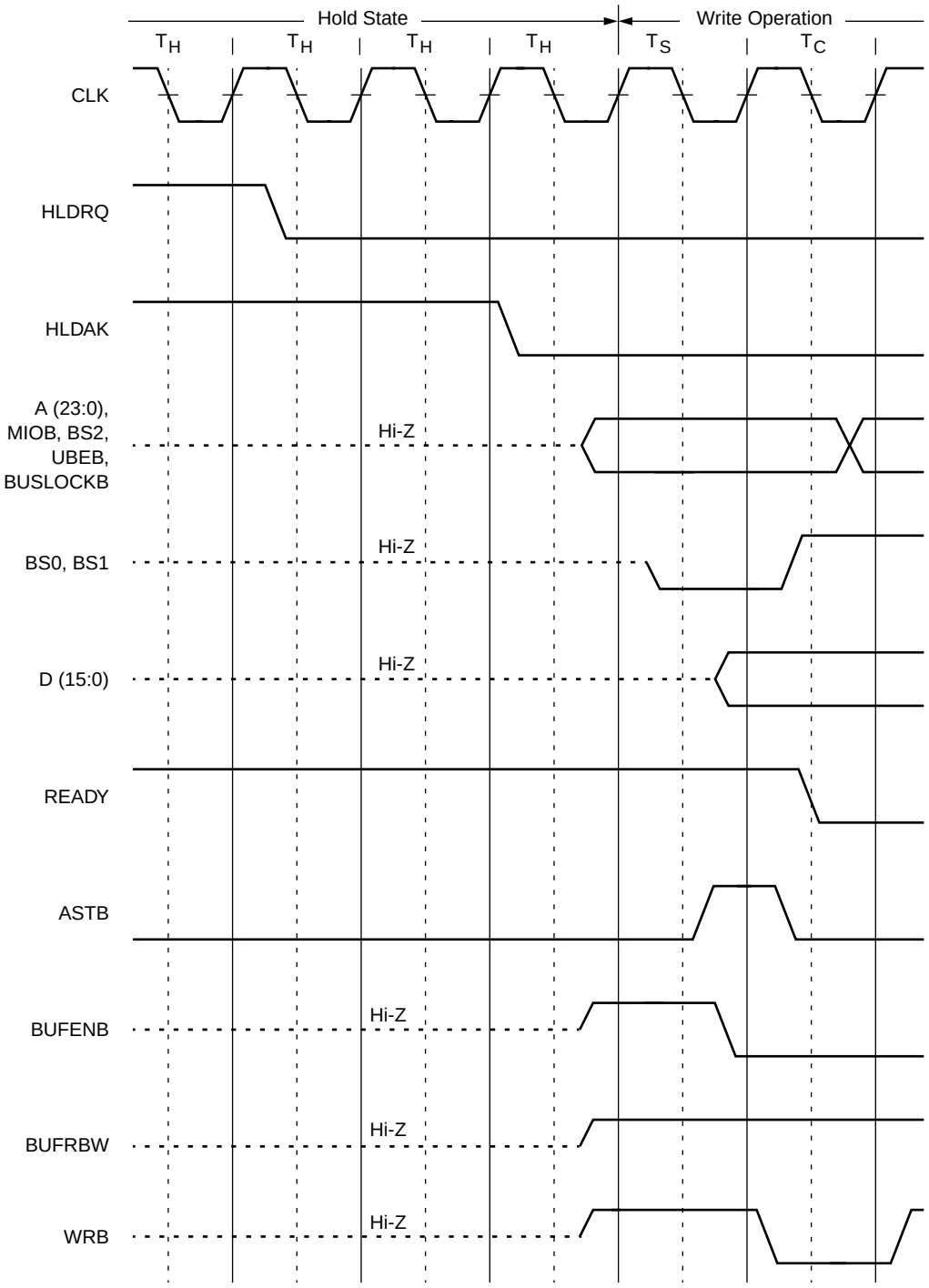


Figure 4-5. Bus Hold Timing (Bus Hold State → Write Operation)



[MEMO]

CHAPTER 5 INTERRUPT FUNCTIONS

Interrupts of the V30MX are roughly divided into two kinds; hardware interrupts and software interrupts. These interrupts are all vectored interrupts which reference a vector table. An interrupt vector table stores the start address of an interrupt service routine.

When an interrupt is generated, the V30MX 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.

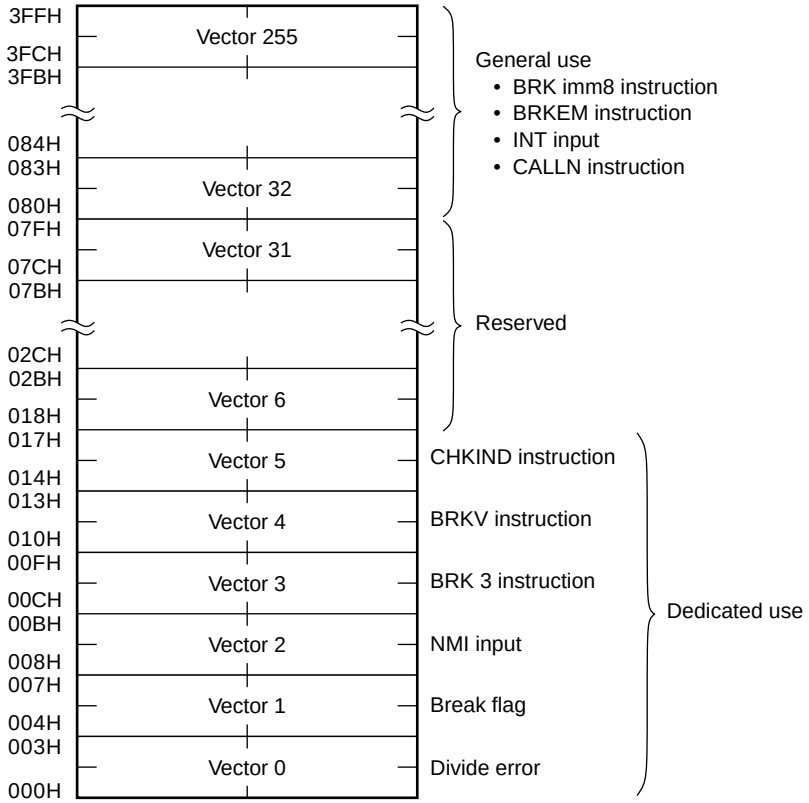
Figure 5-1 shows the interrupt vector table configuration.

Table 5-1. Interrupt Source List

Interrupt Source		Number of Clocks Processed[Note]	Vector No.	Priority Order
		V30MX		
Hardware	NMI (rising edge active)	38	2	2
	INT (high level active)	49	32 to 255	3
Software	DIVU divide error	45	0	1
	DIV divide error	45 to 55		
	CHKIND boundary over	53 to 56	5	
	BRKV	40	4	
	BRK 3	38	3	
	BRK imm8		32 to 255	
	BRKEM imm8			
	CALLN imm8	38		
	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).

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, 4 kinds of instructions, BRK imm8 instruction, BRKEM instruction, INT input and CALLN instruction (in emulation) are possible for general use.

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

Example: Vector 0

002H	003H
000H	001H

PS ← (003H, 002H)
PC ← (001H, 000H)

When creating a program, initialize the content of each vector used based on the example in Figure 5-1 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 reset (0) when interrupt servicing is started, no maskable interrupt (INT) or 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 acknowledge cycle is issued by this interrupt and no INTAK signal is output, either.

5.1.2 Maskable interrupt (INT)

In the 1st acknowledge cycle, it synchronizes with the interrupt controller and in the 2nd acknowledge cycle it reads an interrupt vector (refer to **Figure 5-2 Interrupt Acknowledge Cycle**).

< 1 > In synchronization with a clock fall in the previous cycle, MIOB and BS2 change to a low level to indicate the acknowledge cycle. ASTB is also output.

< 2 > At the start of T_S, BUSLOCKB, BS0 and BS1 change to a low level.

< 3 > At the start of the 1st T_C, BS0 and BS1 are driven high. INTAKB also changes to a low level.

< 4 > Like the normal R/W bus cycle, the input to READY is sampled at the end of each T_C and any number of waits can be inserted. Figure 5-2 shows a case with 1 wait.

< 5 > INTAKB retains a low level during T_C.

< 6 > After the 1st acknowledge cycle, 3 cycles of the T_i state are inserted. BUSLOCKB holds a low level after <2> during the T_i state.

< 7 > The operation of the 2nd acknowledge cycle is almost the same as the 1st acknowledge cycle. However, BUSLOCKB is driven high (inactive) from the start of T_S in the 2nd acknowledge cycle.

< 8 > At the end of the last T_C, an interrupt vector is read from the lower byte of the data bus. Then, at least 7 cycles of T_i are inserted.

The CPU does not drive the data bus until a fall of the last T_i clock. During this period, the address bus and UBEB are driven but those values are invalid.

If the prefetch queue is empty immediately before interrupt acknowledge, a code fetch cycle may be started between the 1st acknowledge cycle and 2nd acknowledge cycle (refer to **Figure 5-3 Interrupt Acknowledge Cycle (with code fetch)**).

Figure 5-2. Interrupt Acknowledge Cycle

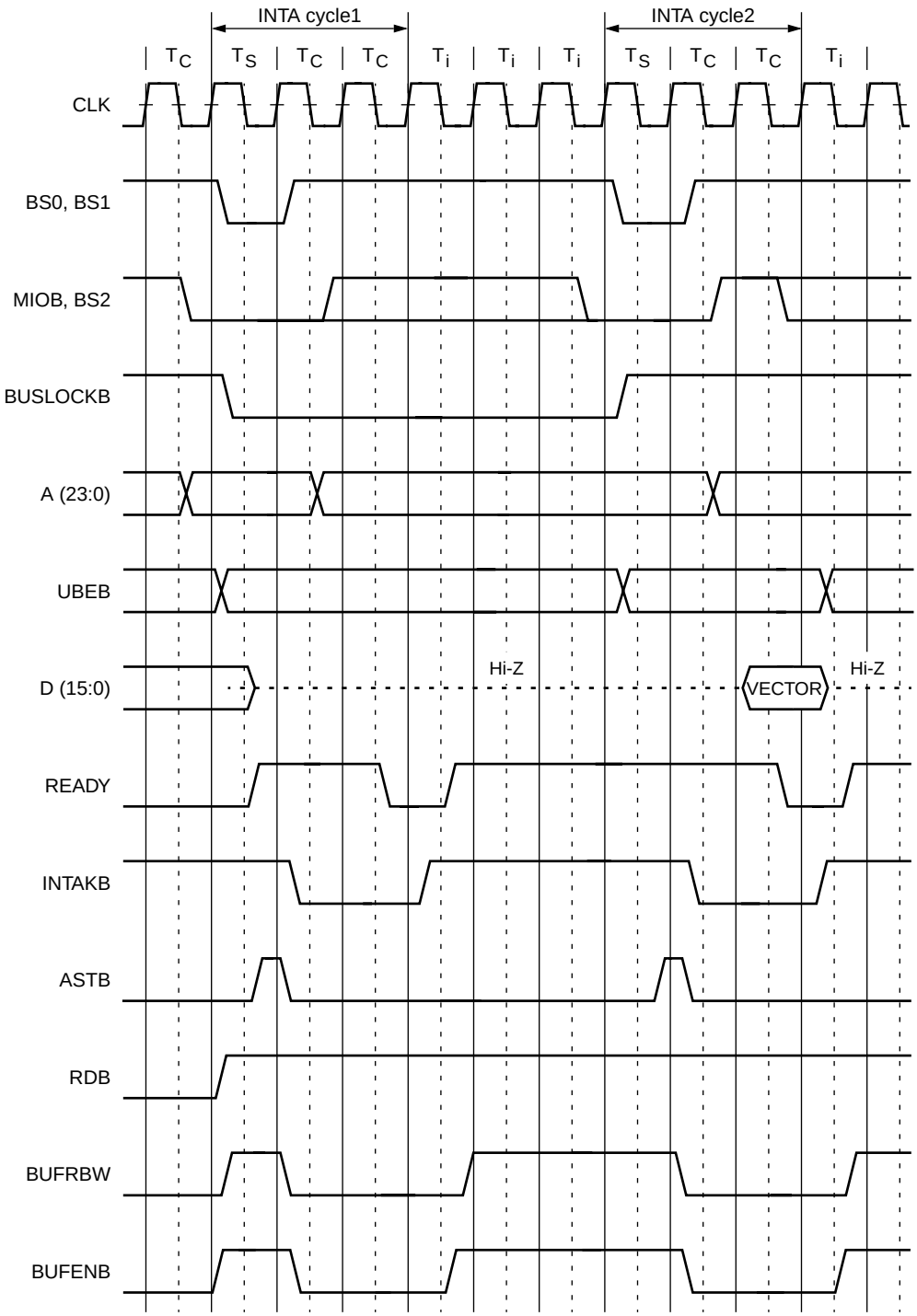
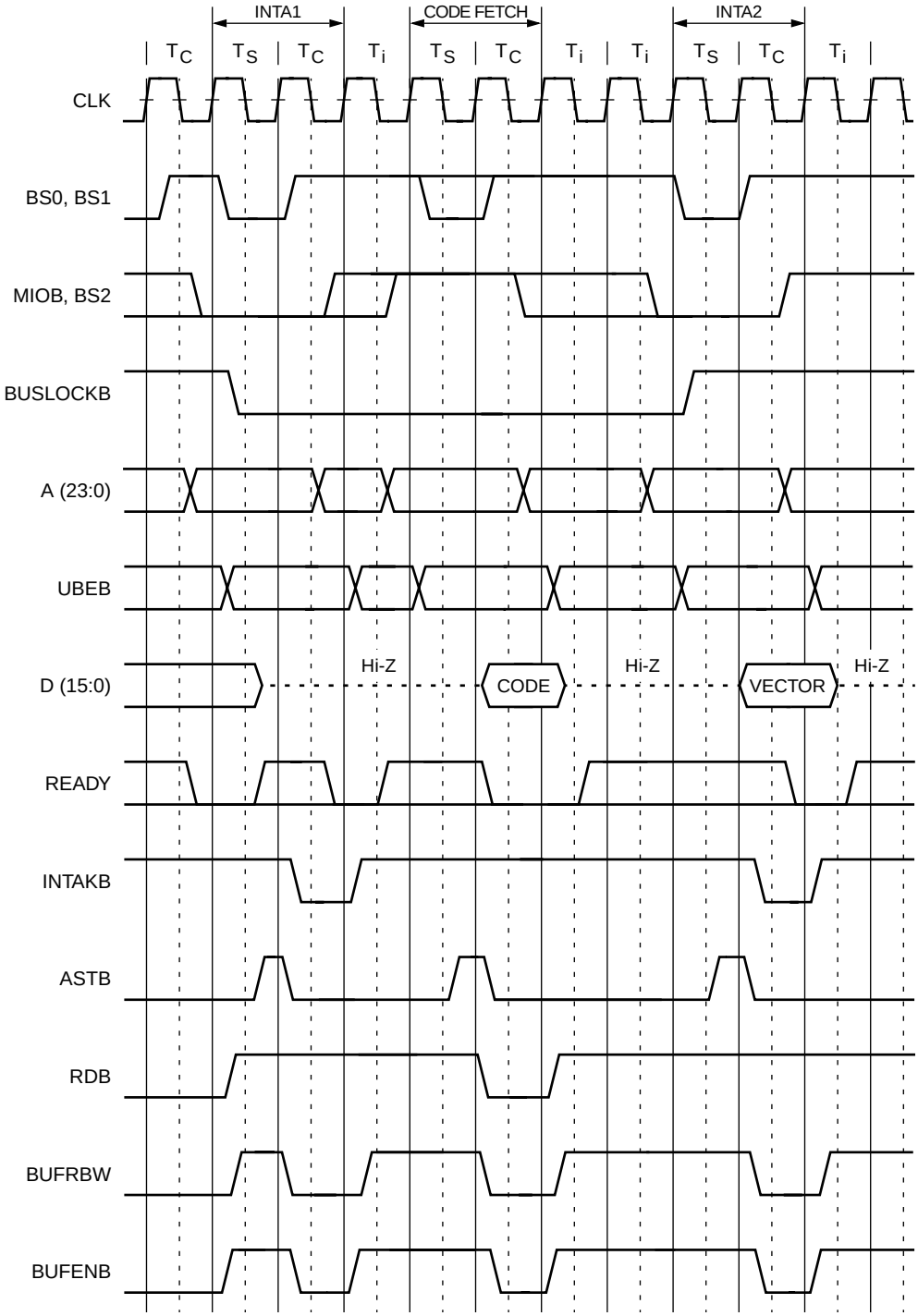


Figure 5-3. Interrupt Acknowledge Cycle (with code fetch)



5.2 Software Interrupts

Software interrupts take precedence over hardware 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 bit 8. However, it is manipulated with the PSW saved to the stack, not by an instruction which directly sets/resets the BRK flag and set/reset processing is indirectly performed by restoring it to the PSW.

When the BRK flag is set, 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 reset 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 reset 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 single step interrupt is acknowledged. Furthermore, between the EI instruction in (5) and the following one instruction only the INT interrupt is not acknowledged.

With the following 5 timings, no interrupt is acknowledged.

(1) Between each of MOV sreg, reg 16; MOV sreg, mem16; MOV reg16, sreg; MOV mem16, sreg; POP sreg instructions and the next instruction

(2) Between segment override prefix (PS:, SS:, DS0:, DS1:) and the next instruction

(3) Between repeat prefix (REPC, REPNC, REP, REPE, REPZ, REPNZ) and the next instruction

(4) Between BUSLOCK instruction and the next instruction

(5) Between EI instruction and the next instruction (only INT interrupt is not acknowledged)

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.

5.4 Interrupt Servicing in Execution of Block Processing Instruction

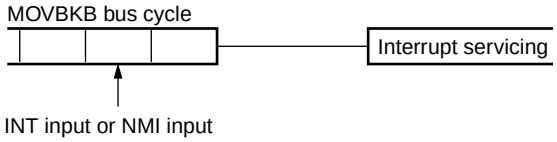
When a hardware interrupt is generated in execution of a primitive block transfer/comparison, input instruction, the V30MX acknowledges it and branches to the corresponding interrupt address.

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 V30MX 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.

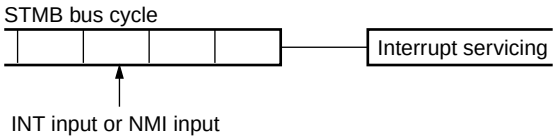
Table 5-2. Number of Bus Cycles Required until Interrupt is Acknowledged

Instruction	IX	IY	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 CW which is operating as a counter for the block data is saved to the stack and CW 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 no more than 3.

Example 1. Good example

In the following example, after returning from NMI interrupt servicing, BUSLOCK, REPC, SS: all function effectively.

```
    BUSLOCK
    REPC
NMI → CMPBKB SS: src-block, dst-block
```

Example 2. Bad example

In the following example, it is accepted that there are 3 kinds of prefix (all the repeat prefixes are of the same kind) and modification is carried out for returns and addresses corresponding to 3 addresses, but actually the prefixes occupy 4 addresses. Therefore, it is not possible to return to BUSLOCK from the interrupt service routine and the program returns to REP.

```
    BUSLOCK
    REP
    REPC
NMI → CMPBK SS: src-block, dst-block
```

CHAPTER 6 STANDBY FUNCTIONS

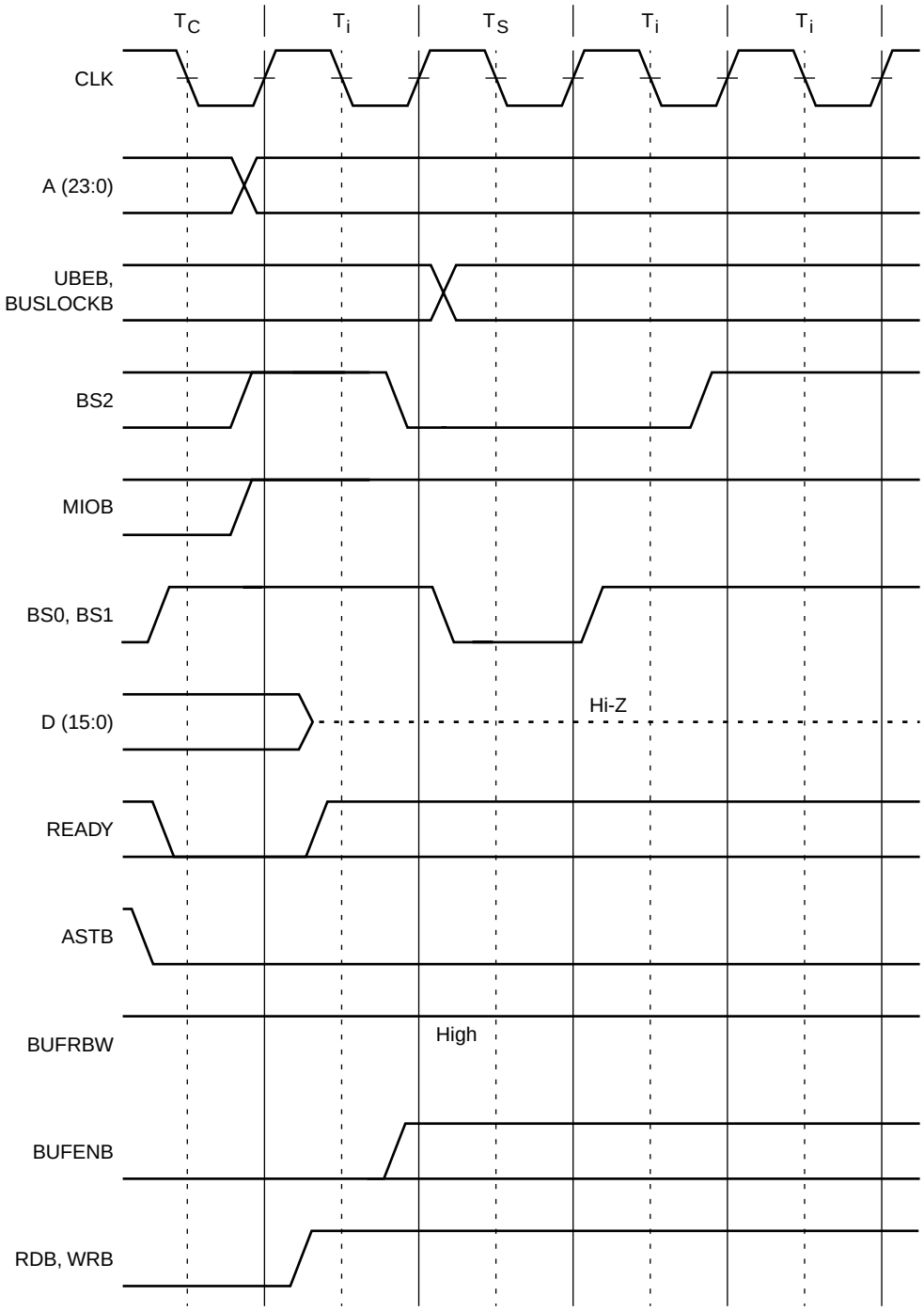
6.1 Setting of Standby Mode

Executing a HALT instruction sets the standby mode.

6.2 Standby Mode

When the standby mode is entered, BS2 to BS0 and MIOB output a HALT status. However, ASTB is not activated but remains low at this time.

Figure 6-1. Timing in Standby Mode



6.3 Release of Standby Mode

There are 2 ways of releasing the standby mode once set; release by a hardware interrupt and release by RESET input.

6.3.1 Release by hardware interrupt

The standby mode is released by a hardware interrupt (NMI input or INT input).

(1) Release by NMI input

Interrupt servicing (NMI routine) is started from the standby mode and the standby mode is released.

When an RETI instruction is executed at the end of the NMI routine, the program returns to the original mode and is restarted from the instruction following the HALT instruction.

(2) Release by INT input

The operation after a release differs depending on an interrupt enabled state (IE flag of PSW = 1) or interrupt disabled state (IE flag of PSW = 0).

(a) Interrupt enabled state

When an RETI instruction is executed at the end of the INT routine, the program returns to the original mode and is restarted from the instruction following a HALT instruction or HLT instruction.

(b) Interrupt disabled state

The program returns to the mode in which the standby mode was started and is restarted from a HALT instruction or the instruction following a HLT instruction.

Caution When releasing the standby mode by INT input with an interrupt disabled, keep the INT signal high until execution of a HALT instruction or the instruction following an HLT instruction is started, that is, for 15 clocks (when it is supposed that the queue is empty with the HALT instruction or HLT instruction executed).
Furthermore, if a wait state has been inserted, add the time corresponding to the wait states.

6.3.2 Release by RESET input

If RESET is input in standby mode, a normal reset operation starts unconditionally. Therefore, the state retained in standby mode becomes invalid and the program that has been stopped in standby mode cannot be restarted.

[MEMO]

CHAPTER 7 RESET FUNCTIONS

When a high level is input to the RESET pin for 4 clocks or more, the V30MX pins change to the statuses shown in Table 7-1. They retain those values while the high level is input.

Table 7-1. Pin Status after Reset

Pin	After Reset
A(23:0)	Hi-Z
D(15:0)	Hi-Z
UBEB	Hi-Z
RDB	Hi-Z
QS0, QS1	L
BUSLOCKB	Hi-Z
BS2 to BS0	Hi-Z
MIOB	Hi-Z
PS0	L
PS1	H
PS2	L
PS3	L
WRB	Hi-Z
INTAKB	H
ASTB	L
BUFRBW	Hi-Z
BUFENB	Hi-Z
HLDAK	L

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

Table 7-2. Initial Values of Registers after Reset

Register	Initial Value
PC	0000H
PFP	0000H
PS	FFFFH
SS, DS0, DS1	0000H
AW, BW, CW, DW	not decided
SP, BP	not decided
IX, IY	not decided
PSW	1111000000000010B
IDX	00H
EC	00H
EDL11 to EDL0, EDH11 to EDH0	00H
EA	00H

CHAPTER 8 TEST FUNCTIONS

The V30MX has a unit test function using the test bus like other mega-functions. CHAPTER 8 describes this test function.

The V30MX test function is almost the same as a mega-function, but note that it is different only in the following points.

- In unit test mode and standby test mode, the 3-state output normal pins are not set to high impedance but output mode.

8.1 Test Pins

In addition to the normal pins (A (23:0), D (15:0), RDB, etc.) the V30MX has the following test pins.

8.1.1 Test bus pins (TBI (27:0), TBO (71:0))

The test bus pins are used in place of the normal pins in unit test mode.
Refer to the **User's Manual – Design** of each family for details.

8.1.2 BUNRI, TEST

Used for selection of normal/unit test/standby test modes.

Table 8-1. Test Mode Settings

BUNRI	TEST	Mode
0	X	Normal mode
1	0	Standby test mode
1	1	Unit test mode

8.2 Normal Mode

This is a mode that the customers normally use.

When the BUNRI pin is low, the normal pin is valid and normal mode is entered. At this time, input from TBI (27:0) is ignored and TBO (71:0) go high impedance.

8.3 Unit Test Mode and Standby Test Mode

When the BUNRI pin is high, input to the normal pins is ignored (invalid) and the test mode is entered. There are two test modes; unit test mode and standby test mode.

In either case, A (23:0) and D (15:0) go to high impedance.

8.3.1 Unit test mode

When both the BUNRI pin and TEST pin are high, the unit test mode is entered. In unit test mode, input from the normal pins is ignored and instead input from the test bus input pins TBI (27:0) is valid. Output values corresponding to each normal pin appear at test bus output pins TBO (71:0).

Of the normal pins, A (23:0) and D (15:0) go to high impedance.

The normal output pins including other 3-state output pins are undefined. Note that in unit test mode, this undefined state changes depending on the internal state or goes to high impedance. Furthermore, DC, CNT1 and CNT2 do not correctly express the status of the 3-state pin. When controlling the high impedance state, for example, BUNRI should be ORed with DC, CNT1 and CNT2.

Caution This unit test mode is used by NEC to facilitate testing and not available to customers.

8.3.2 Standby test mode

When the BUNRI pin is high and TEST pin is low, the standby test mode is entered. This mode is used for the mega-function that is not tested yet for test circuit check simulation or user logic separation simulation.

Input from the normal pins is ignored and A (23:0) and D (15:0) of the normal output pins go to high impedance and other output pins retain their previous statuses.

Input from test bus input pins TBI (27:0) is also ignored and test bus output pins TBO (71:0) go to high impedance.

APPENDIX LIST OF NUMBER OF INSTRUCTION EXECUTION CLOCKS

List of Number of Instruction Execution Clocks (1/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
ADD	reg, reg'	0	–		2	2	2
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	2	0	–	18	18	14
			1	Odd	26	26	18
				Even		18	14
	acc, imm	0	–		4	4	4
ADD4S[Note]	[DS1-spec:] dst-string, [Seg-spec:] src-string	0	–		19×m+7	19×m+7	19×m+7
	None	0	–		19×m+7	19×m+7	19×m+7
ADDC	reg, reg'	0	–		2	2	2
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	2	0	–	18	18	14
			1	Odd	26	26	18
				Even		18	14
	acc, imm	0	–		4	4	4
ADJ4A	None	0	–		3	3	3
ADJ4S	None	0	–		3	3	3
ADJBA	None	0	–		7	7	7
ADJBS	None	0	–		7	7	7

Note m: Number of BCD digits × 1/2

List of Number of Instruction Execution Clocks (2/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
AND	reg, reg'	0	–		2	2	2
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	2	0	–	18	18	14
			1	Odd	26	26	18
				Even		18	14
	acc, imm	0	–		4	4	4
BC	short-label	0	CY=1		14	14	14
			CY=0		4	4	4
BCWZ	short-label	0	CW=0		5	5	5
			CW=0		13	13	13
BE	short-label	0	Z=1		14	14	14
			Z=0		4	4	4
BGE	short-label	0	S⊽V=1		4	4	4
			S⊽V=0		14	14	14
BGT	short-label	0	(S⊽V) ∨Z=1		4	4	4
			(S⊽V) ∨Z=0		14	14	14
BH	short-label	0	CY∨Z=1		4	4	4
			CY∨Z=0		14	14	14
BL	short-label	0	CY=1		14	14	14
			CY=0		4	4	4
BLE	short-label	0	(S⊽V) ∨Z=1		14	14	14
			(S⊽V) ∨Z=0		4	4	4
BLT	short-label	0	S⊽V=1		14	14	14
			S⊽V=0		4	4	4
BN	short-label	0	S=1		14	14	14
			S=0		4	4	4
BNC	short-label	0	CY=1		4	4	4
			CY=0		14	14	14
BNE	short-label	0	Z=1		4	4	4
			Z=0		14	14	14

List of Number of Instruction Execution Clocks (3/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
BNH	short-label	0	CY∨Z=1		14	14	14
			CY∨Z=0		4	4	4
BNL	short-label	0	CY=1		4	4	4
			CY=0		14	14	14
BNV	short-label	0	V=1		4	4	4
			V=0		14	14	14
BNZ	short-label	0	Z=1		4	4	4
			Z=0		14	14	14
BP	short-label	0	S=1		4	4	4
			S=0		14	14	14
BPE	short-label	0	P=1		14	14	4
			P=0		4	4	4
BPO	short-label	0	P=1		4	4	14
			P=0		14	14	13
BR	near-label	0	–		13	13	13
	short-label	0	–		12	12	12
	regptr16	0	–		11	11	11
	memptr16	1	–	Odd	24	24	20
				Even		20	18
	far-label	0	–		15	15	15
	memptr32	2	–	Odd	35	35	27
				Even		27	23
BRK	3	5	–	Odd	50	50	30
				Even		38	28
	imm8 (≠3)	5	–	Odd	50	50	30
				Even		38	28
BRKEM	imm8	5	–	Odd	50	50	30
				Even		38	28
BRKV	None (when V=1)	5	–	Odd	52	52	32
				Even		40	30
	None (when V=0)	5	–		3	3	3
BRKXA	imm8	2	–		–	–	–
BUSLOCK	None	0	–		2	2	2
BV	short-label	0	V=1		14	14	14
			V=0		4	4	4
BZ	short-label	0	Z=1		14	14	14
			Z=0		4	4	4

List of Number of Instruction Execution Clocks (4/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
CALL	near-proc	1	–	Odd	20	20	16
				Even		16	14
	regptr16	1	–	Odd	18	18	14
				Even		14	12
	memptr16	2	–	Odd	31	31	23
				Even		23	19
	far-proc	2	–	Odd	29	29	21
				Even		21	17
	memptr32	4	–	Odd	47	47	31
				Even		31	23
CALLN	imm8	5	–	Odd	58	58	38
				Even		3	28
CHKIND	reg16, mem32[Note] (when interrupt condition is established)	7	–	Odd	73 to 76	73 to 76	45 to 48
				Even		53 to 56	39 to 42
	reg16, mem32 (when interrupt condition is not established)	2	–	Odd	26	26	18
				Even		18	14
CLR1	reg8, CL	0	–		5	5	5
	mem8, CL	0	–		14	14	14
	reg16, CL	0	–		5	5	5
	mem16, CL	2	–	Odd	22	22	14
				Even		14	10
	reg8, imm3	0	–		6	6	6
	mem8, imm3	0	–		15	15	15
	reg16, imm4	0	–		6	6	6
	mem16, imm4	2	–	Odd	23	23	15
				Even		15	11
	CY	0	–		2	2	2
	DIR	0	–		2	2	2

Note The number of clocks varies depending on the timing at which an interrupt request is acknowledged.

List of Number of Instruction Execution Clocks (5/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
CMP	reg, reg'	0	–		2	2	2
	mem, reg	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	1	0	–	13	13	11
			1	Odd	17	17	13
				Even		13	11
	acc, imm	0	–		4	4	4
CMP4S[Note 1]	[DS1-spec:] dst-string, [Seg-spec:] stc-string	0	–		19×m+7	19×m+7	19×m+7
	None	0	–		19×m+7	19×m+7	19×m+7
CMPBK[Note 2]	[Seg-spec:] src-block, [DS1-spec:] dst-block	2×rep(2)	0	–	7+14×rep(13)	7+14×rep(13)	7+10×rep(9)
			1	Odd, Odd	7+22×rep(21)	7+22×rep(21)	7+14×rep(13)
				Odd, Even		7+18×rep(17)	7+12×rep(11)
				Even, Even		7+14×rep(13)	7+10×rep(9)
CMPBKB[Note 2]	None	2×rep(2)	0	–	7+14×rep(13)	7+14×rep(13)	7+10×rep(9)
			1	Odd, Odd	7+22×rep(21)	7+22×rep(21)	7+12×rep(13)
				Odd, Even		7+18×rep(17)	7+16×rep(11)
				Even, Even		7+14×rep(13)	7+10×rep(9)
CMPBKW[Note 2]	None	2×rep(2)	0	–	7+14×rep(13)	7+14×rep(13)	7+10×rep(9)
			1	Odd, Odd	7+22×rep(21)	7+22×rep(21)	7+14×rep(13)
				Odd, Even		7+18×rep(17)	7+12×rep(11)
				Even, Even		7+14×rep(13)	7+10×rep(9)
CMPM[Note 2]	[DS1-spec:] dst-block	1×rep(1)	0	–	7+10×rep(7)	7+10×rep(7)	7+8×rep(5)
			1	Odd	7+14×rep(11)	7+14×rep(11)	7+10×rep(7)
				Even		7+10×rep(7)	7+8×rep(5)
CMPMB[Note 2]	None	1×rep(1)	0	–	7+10×rep(7)	7+10×rep(7)	7+8×rep(5)
			1	Odd	7+14×rep(11)	7+14×rep(11)	7+10×rep(7)
				Even		7+10×rep(7)	7+8×rep(5)

Notes 1. m : Number of BCD digits x 1/2
2. ( ) : Applies to only one-time processing

List of Number of Instruction Execution Clocks (6/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
CMPMW[Note 1]	None	1×rep(1)	0	–	7+10×rep(7)	7+10×rep(7)	7+8×rep(5)
			1	Odd	7+14×rep(11)	7+14×rep(11)	7+10×rep(7)
				Even		7+10×rep(7)	7+8×rep(5)
CVTBD	None	0	–		15	15	15
CVTBW	None	0	–		2	2	2
CVTDB	None	0	–		7	7	7
CVTWL[Note 2]	None	0	–		4, 5	4, 5	4, 5
DBNZ	short-label	0	CW≠0		13	13	13
			CW=0		5	5	5
DBNZE	short-label	0	When CW ≠ 0 and Z = 1		14	14	14
			Other than above		5	5	5
DBNZNE	short-label	0	When CW ≠ 0 and Z = 0		14	14	14
			Other than above		5	5	5
DEC	reg8	0	–		2	2	2
	mem	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg16	0	–		2	2	2
DI	None	0	–		2	2	2
DISPOSE	None	1	–	Odd	10	10	6
				Even		6	4
DIV[Note 2]	reg8	0	–		29 to 34	29 to 34	29 to 34
	mem8	0	–		34 to 39	34 to 39	34 to 39
	reg16	0	–		38 to 43	38 to 43	38 to 43
	mem16	1	–	Odd	47 to 52	47 to 52	43 to 48
				Even		43 to 48	41 to 46
DIVU	reg8	0	–		19	19	19
	mem8	0	–		25	25	25
	reg16	0	–		25	25	25
	mem16	1	–	Odd	34	34	30
				Even		30	28
DS0:	None	0	–		2	2	2
DS1:	None	0	–		2	2	2

Notes 1. ( ) : Applies to only one-time processing.
2. The number of clocks varies depending on the data value.

List of Number of Instruction Execution Clocks (7/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
EI	None	0	–		2	2	2
EXT[Note 1]	reg8, reg8'	1 or 2	–	Odd	34 to 59	34 to 59	32 to 55
				Even		26 to 55	22 to 47
	reg8, imm4	1 or 2	–	Odd	34 to 59	34 to 59	32 to 55
				Even		26 to 55	22 to 47
FPO1	fp-op	0	–		2	2	2
	fp-op, mem	1	–	Odd	15	15	11
				Even		11	9
FPO2	fp-op	0	–		2	2	2
	fp-op, mem	1	–	Odd	15	15	11
				Even		11	9
HALT	None	0	–		2	2	2
IN	acc, imm8	1	0	–	9	9	7
			1	Odd	13	13	9
				Even		9	7
	acc, DW	1	0	–	8	8	6
			1	Odd	12	12	8
				Even		8	6
INC	reg8	0	–		2	2	2
	mem	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg16	0	–		2	2	2
INM[Note 2]	[DS1-spec:] dst-block, DW	2×rep(2)	0	–	9+8×rep(10)	9+8×rep(10)	9+4×rep(6)
			1	Odd, Odd	9+16×rep(18)	9+16×rep(18)	9+8×rep(10)
				Odd, Even		9+12×rep(14)	9+6×rep(8)
				Even, Even		9+8×rep(10)	9+4×rep(6)
INS[Note 1]	reg8, reg8'	2 or 4	–	Odd	35 to 133	35 to 133	27 to 117
				Even		31 to 117	27 to 109
	reg8, imm4	2 or 4	–	Odd	35 to 133	35 to 133	27 to 117
				Even		31 to 117	27 to 109
LDEA	reg16, mem16	0	–		4	4	4
LDM[Note 2]	[Seg-spec:] src-block	1×rep(1)	0	–	7+9×rep(7)	7+9×rep(7)	7+7×rep(5)
			1	Odd	7+13×rep(11)	7+13×rep(11)	7+9×rep(7)
				Even		7+9×rep(7)	7+7×rep(5)

Notes 1. The number of clocks varies depending on the data value.
2. ( ) : Applies to only one-time processing.

List of Number of Instruction Execution Clocks (8/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
LDMB[Note]	None	1×rep(1)	0	–	7+9×rep(7)	7+9×rep(7)	7+7×rep(5)
			1	Odd	7+13×rep(11)	7+13×rep(11)	7+9×rep(7)
				Even		7+9×rep(7)	7+7×rep(5)
LDMW[Note]	None	1×rep(1)	0	–	7+9×rep(7)	7+9×rep(7)	7+7×rep(5)
			1	Odd	7+13×rep(11)	7+13×rep(11)	7+9×rep(7)
				Even		7+9×rep(7)	7+7×rep(5)
MOV	reg, reg'	0	–		2	2	2
	mem, reg	1	0	–	9	9	7
			1	Odd	13	13	9
				Even		9	7
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	mem, imm	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	acc, dmem	1	0	–	10	10	8
			1	Odd	14	14	10
				Even		10	8
	dmem, acc	1	0	–	9	9	7
			1	Odd	13	13	9
				Even		9	7
	sreg, reg16	0	–		2	2	2
	sreg, mem16	1	–	Odd	15	15	11
				Even		11	9
	reg16, sreg	0	–		2	2	2
	mem16, sreg	1	–	Odd	14	14	10
				Even		10	8
	DS0, reg16, mem32	2	–	Odd	26	26	18
				Even		18	14
	DS1, reg16, mem32	2	–	Odd	26	26	18
				Even		18	14
	AH, PSW	0	–		2	2	2
	PSW, AH	0	–		3	3	3

Note ( ) : Applies to only one-time processing.

List of Number of Instruction Execution Clocks (9/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
MOVBK[Note 1]	[DS1-spec:] dst-block, [Seg-spec:] src-block	2×rep(2)	0	–	11+8×rep(11)	11+8×rep(11)	11+4×rep(7)
			1	Odd, Odd	11+16×rep(19)	11+16×rep(19)	11+8×rep(11)
				Odd, Even		11+12×rep(15)	11+6×rep(9)
				Even, Even		11+8×rep(11)	11+4×rep(7)
MOVBKB[Note 1]	None	2×rep(2)	0	–	11+8×rep(11)	11+8×rep(11)	11+4×rep(7)
			1	Odd, Odd	11+16×rep(19)	11+16×rep(19)	11+8×rep(11)
				Odd, Even		11+12×rep(15)	11+6×rep(9)
				Even, Even		11+8×rep(11)	11+4×rep(7)
MOVBKW[Note 1]	None	2×rep(2)	0	–	11+8×rep(11)	11+8×rep(11)	11+4×rep(7)
			1	Odd, Odd	11+16×rep(19)	11+16×rep(19)	11+8×rep(11)
				Odd, Even		11+12×rep(15)	11+6×rep(9)
				Even, Even		11+8×rep(11)	11+4×rep(7)
MUL[Note 2]	reg8	0	–		33 to 39	33 to 39	33 to 39
	mem8	0	–		39 to 45	39 to 45	39 to 45
	reg16	0	–		41 to 47	41 to 47	41 to 47
	mem16	1	–	Odd	51 to 57	51 to 57	47 to 53
				Even		47 to 53	45 to 51
	reg16, imm8	0	–		28 to 34	28 to 34	28 to 34
	reg16, imm16	0	–		36 to 42	36 to 42	36 to 42
	reg16, reg16', imm8	0	–		28 to 34	28 to 34	28 to 34
	reg16, mem16, imm8	1	–	Odd	38 to 44	38 to 44	34 to 40
				Even		34 to 40	32 to 38
	reg16, reg16', imm16	0	–		36 to 42	36 to 42	36 to 42
	reg16, mem16, imm16	1	–	Odd	46 to 52	46 to 52	42 to 48
				Even		42 to 48	40 to 46
MULU[Note 2]	reg8	0	–		21, 22	21, 22	21, 22
	mem8	1	–		27, 28	27, 28	25, 26
	reg16	0	–		29, 30	29, 30	29, 30
	mem16	1	–	Odd	39, 40	39, 40	35, 36
				Even		35, 36	33, 34
NEG	reg	0	–		2	2	2
	mem	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12

Notes 1. ( ) : Applies to only one-time processing.
2. The number of clocks varies depending on the data value.

List of Number of Instruction Execution Clocks (10/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
NOP	None	0	–		3	3	3
NOT	reg	0	–		2	2	2
	mem	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
NOT1	reg8, CL	0	–		4	4	4
	mem8, CL	0	–		13	13	13
	reg16, CL	0	–		4	4	4
	mem16, CL	2	–	Odd	21	21	13
				Even		13	9
	reg8, imm3	0	–		5	5	5
	mem8, imm3	0	–		14	14	14
	reg16, imm4	0	–		5	5	5
	mem16, imm4	2	–	Odd	22	22	14
				Even		14	10
	CY	0	–		2	2	2
OR	reg, reg'	0	–		2	2	2
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	2	0	–	18	18	14
			1	Odd	26	26	18
				Even		18	14
	acc, imm	0	–		4	4	4
OUT	imm8, acc	1	0	–	8	8	6
			1	Odd	12	12	8
				Even		8	6
	DW, acc	1	0	–	8	8	6
			1	Odd	12	12	8
				Even		8	6

List of Number of Instruction Execution Clocks (11/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
OUTM[Note 1]	DW, [Seg-spec:] src-block	2×rep(2)	0	–	9+8×rep(10)	9+8×rep(10)	9+4×rep(6)
			1	Odd, Odd	9+16×rep(18)	9+16×rep(18)	9+8×rep(10)
				Odd, Even		9+12×rep(14)	9+6×rep(8)
				Even, Even		9+8×rep(10)	9+4×rep(6)
POLL[Note 2]	None	0	–		2+5×poll	2+5×poll	2+5×poll
POP	mem16	2	–	Odd	25	25	17
				Even		17	13
	reg16	1	–	Odd	12	12	8
				Even		8	6
	sreg	1	–	Odd	12	12	8
				Even		8	6
	PSW	1	–	Odd	12	12	8
				Even		8	6
	R	7	–	Odd	75	75	47
				Even		43	29
PREPARE	imm16, imm8 (when imm8=0)	1	–	Odd	16	16	12
				Even		12	10
	imm16, imm8 (when imm8≥1)	2×imm8	–	Odd	23+16(imm8–1)	23+16(imm8–1)	23+8(imm8–1)
				Even		19+8(imm8–1)	19+4(imm8–1)
PS:	None	0	–		2	2	2
PUSH	mem16	2	–	Odd	26	26	18
				Even		18	14
	reg16	1	–	Odd	12	12	8
				Even		8	6
	sreg	1	–	Odd	12	12	8
				Even		8	6
	PSW	1	–	Odd	12	12	8
				Even		8	6
	R	8	–	Odd	67	67	35
				Even		35	19
	imm8	1	–	Odd	11	11	7
				Even		7	5
	imm16	1	–	Odd	12	12	8
				Even		8	6

Notes 1. ( ) : Applies to only one-time processing.
2. poll : Number of POLLB pin sampling times

List of Number of Instruction Execution Clocks (12/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
REP	None	0	–		2	2	2
REPC	None	0	–		2	2	2
REPE	None	0	–		2	2	2
REPNC	None	0	–		2	2	2
REPNE	None	0	–		2	2	2
REPNZ	None	0	–		2	2	2
REPZ	None	0	–		2	2	2
RET	None (segment internal call)	1	–	Odd	19	19	15
				Even		15	13
	None (segment external call)	2	–	Odd	29	29	21
				Even		21	17
	pop-value (segment internal call)	1	–	Odd	24	24	20
				Even		20	18
	pop-value (segment external call)	2	–	Odd	32	32	24
				Even		24	20
RETEM	None	3	–	Odd	39	39	27
				Even		27	21
RETI	None	3	–	Odd	39	39	27
				Even		27	21
RETXA	imm8	2	–		–	–	–
ROL[Note]	reg, 1	0	–		6	6	6
	mem, 1	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, CL	0	–		7+n	7+n	7+n
	mem, CL	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
	reg, imm8	0	–		7+n	7+n	7+n
	mem, imm8	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
ROL4	reg8	0	–		13	13	13
	mem8	0	–		28	28	28

Note n: Number of shifts

List of Number of Instruction Execution Clocks (13/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
ROLC[Note]	reg, 1	0	–		6	6	6
	mem, 1	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, CL	0	–		7+n	7+n	7+n
	mem, CL	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
	reg, imm8	0	–		7+n	7+n	7+n
	mem, imm8	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
ROR[Note]	reg, 1	0	–		6	6	6
	mem, 1	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, CL	0	–		7+n	7+n	7+n
	mem, CL	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
	reg, imm8	0	–		7+n	7+n	7+n
	mem, imm8	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
ROR4	reg8	0	–		17	17	17
	mem8	0	–		32	32	32

Note n: Number of shifts

List of Number of Instruction Execution Clocks (14/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
RORC[Note]	reg, 1	0	–		6	6	6
	mem, 1	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, CL	0	–		7+n	7+n	7+n
	mem, CL	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
	reg, imm8	0	–		7+n	7+n	7+n
	mem, imm8	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
SET1	reg8, CL	0	–		4	4	4
	mem8, CL	0	–		13	13	13
	reg16, CL	0	–		4	4	4
	mem16, CL	2	–	Odd	21	21	15
				Even		13	9
	reg8, imm3	0	–		5	5	5
	mem8, imm3	0	–		14	14	14
	reg16, imm4	0	–		5	5	5
	mem16, imm4	2	–	Odd	22	22	16
				Even		14	10
	CY	0	–		2	2	2
	DIR	0	–		2	2	2
SHL[Note]	reg, 1	0	–		6	6	6
	mem, 1	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, CL	0	–		7+n	7+n	7+n
	mem, CL	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
	reg, imm8	0	–		7+n	7+n	7+n
	mem, imm8	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n

Note n: Number of shifts

List of Number of Instruction Execution Clocks (15/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
SHR[Note 1]	reg, 1	0	–		6	6	6
	mem, 1	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, CL	0	–		7+n	7+n	7+n
	mem, CL	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
	reg, imm8	0	–		7+n	7+n	7+n
	mem, imm8	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
SHRA[Note 1]	reg, 1	0	–		6	6	6
	mem, 1	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, CL	0	–		7+n	7+n	7+n
	mem, CL	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
	reg, imm8	0	–		7+n	7+n	7+n
	mem, imm8	2	0	–	19+n	19+n	15+n
			1	Odd	27+n	27+n	19+n
				Even		19+n	15+n
SS:	None	0	–		2	2	2
STM[Note 2]	[DS1-spec:] dst-block	1×rep(1)	0	–	7+4×rep(7)	7+4×rep(7)	7+2×rep(5)
			1	Odd	7+8×rep(11)	7+8×rep(11)	7+4×rep(7)
				Even		7+4×rep(7)	7+2×rep(5)
STMB[Note 2]	None	1×rep(1)	0	–	7+4×rep(7)	7+4×rep(7)	7+2×rep(5)
			1	Odd	7+8×rep(11)	7+8×rep(11)	7+4×rep(7)
				Even		7+4×rep(7)	7+2×rep(5)
STMW[Note 2]	None	1×rep(1)	0	–	7+4×rep(7)	7+4×rep(7)	7+2×rep(5)
			1	Odd	7+8×rep(11)	7+8×rep(11)	7+4×rep(7)
				Even		7+4×rep(7)	7+2×rep(5)

Notes 1. n : Number of shifts
2. ( ) : Applies to only one-time processing.

List of Number of Instruction Execution Clocks (16/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
SUB	reg, reg'	0	–		2	2	2
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	2	0	–	18	18	14
			1	Odd	26	26	18
				Even		18	14
	acc, imm	0	–		4	4	4
SUB4S[Note]	[DS1-spec:] dst-string, [Seg-spec:] src-string	0	–		19×m+7	19×m+7	19×m+7
	None	0	–		19×m+7	19×m+7	19×m+7
SUBC	reg, reg'	0	–		2	2	2
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	2	0	–	18	18	14
			1	Odd	26	26	18
				Even		18	14
	acc, imm	0	–		4	4	4

Note m: Number of BCD digits × 1/2

List of Number of Instruction Execution Clocks (17/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
TEST	reg, reg'	0	–		2	2	2
	mem, reg	1	0	–	10	10	8
			1	Odd	14	14	10
				Even		10	8
	reg, mem	1	0	–	10	10	8
			1	Odd	14	14	10
				Even		10	8
	reg, imm	0	–		4	4	4
	mem, imm	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	acc, imm	0	–		4	4	4
TEST1	reg8, CL	0	–		3	3	3
	mem8, CL	0	–		8	8	8
	reg16, CL	0	–		3	3	3
	mem16, CL	1	–	Odd	12	12	8
				Even		8	6
	reg8, imm3	0	–		4	4	4
	mem8, imm3	0	–		9	9	9
	reg16, imm4	0	–		4	4	4
	mem16, imm4	1	–	Odd	13	13	9
				Even		9	7
TRANS	src-table	1	–		9	9	7
	None	1	–		9	9	7
TRANSB	None	1	–		9	9	7
XCH	reg, reg'	0	–		3	3	3
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	AW, reg16	0	–		3	3	3
	reg16, AW	0	–		3	3	3

List of Number of Instruction Execution Clocks (18/18)

Mnemonic	Operand	Number of Word Transfers	Condition		Clocks		
			W	Address	V20HL	V30HL	V30MX
XOR	reg, reg'	0	–		2	2	2
	mem, reg	2	0	–	16	16	12
			1	Odd	24	24	16
				Even		16	12
	reg, mem	1	0	–	11	11	9
			1	Odd	15	15	11
				Even		11	9
	reg, imm	0	–		4	4	4
	mem, imm	2	0	–	18	18	14
			1	Odd	26	26	18
				Even		18	14
	acc, imm	0	–		4	4	4

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: +55-11-889-1689

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: 02-719-5951

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

Japan
NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
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	❑	❑	❑	❑