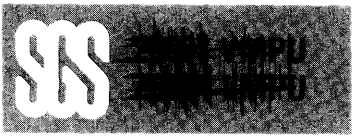
Virtual Memory Processing Unit



Features

- Regular, easy-to-use architecture.
- Instruction set more powerful than many minicomputers.
- Direct addressing capability of up to 8M bytes in each address space.
- Supports implementation of virtual memory systems.
- Eight user-selected addressing modes.
- Wide range of data types including bits, bytes, words, 32-bit long words, and byte and word strings.
- Binary-compatible with Z8001/2 CPUs.
- Separate System and Normal operating modes.
- Sophisticated interrupt structure.
- Resource-sharing capabilities for multiprocessing systems.
- Multi-programming support.
- 32-bit operations, including signed multiply and divide.
- Z-BUS compatible.
- Multiple clock rates: 4, 8, or 10 MHz.

General Description

The Virtual Memory Microprocessor Units (Z8003 and Z8004 VMPUs) accommodate applications that range from the simplest to the most complex.

The Z8003 VMPU uses both segmented and nonsegmented address spaces. It also provides facilities for the implementation of demand segment swapping or a demand paged virtual memory system.

The Z8004 VMPU uses only nonsegmented address spaces. It also provides facilities for the implementation of a demand paged virtual memory system.

Both VMPUs interface with the entire Z8000 Family of support components. Used alone or with Z8000 Family components, the advanced architecture of these LSI VMPUs permits the implementation of systems that have the flexibility and the sophisticated features usually associated with minicomputers or mainframe computers.

The Z8003/4 microprocessors are binary compatible with other Z8000 Family microprocessors. The features that distinguish these microprocessors from the Z8001 and Z8002 microprocessors are the abort capability and the Test and Set status.

An abort request function aids in the implementation of virtual memory systems. The abort function is initiated by memory management circuitry external to the VMPU when an address issued by the VMPU references information (data or instructions) that is not in main memory. After the abort interrupt function, a service routine must bring the page or segment containing the addressed data into main memory. The mainstream program is then restarted at the point of interruption. An abort interrupt differs from a standard interrupt in that the executing instruction is stopped immediately upon detection of the interrupt; this prevents the loss of information needed for a successful restart.

The Test and Set instruction (TSET), in addition to its semaphore test and set function, causes status code 1111 to be placed onto output lines ST_0-ST_3 during the data read bus transaction. It can be used by external circuitry to lock memory to prevent it from being accessed by any other device during the execution of the current TSET instruction.

The architectural features of the VMPU combine to produce a powerful and versatile microprocessor. These features result in the following benefits:

- High-density code
- Efficient compilation of programs
- Support for typical operating system operations
- Complex data structures
- Large-scale virtual memory systems

General Description (Continued)

The VMPU is designed so that a powerful memory management system can be used to improve the utilization fo the main memory either as a standard memory or as a virtual memory configuration. SGS produces Memory Management Units (MMUs) designed for use with the Z8003 VMPU to implement both virtual and nonvirtual memory systems.

The architectural resources of the VMPUs include sixteen 16-bit registers, seven data types (ranging from bits to 32-bit words, and byte and word strings), eight addressing modes, and a powerful instruction set.

A general mechanism has been provided for extending the basic instruction set through the use of external devices called Extended Processing Units (EPUs). In general, an EPU is dedicated to perfoming complex and time-consuming tasks (such as floating-point arithmetic) so as to unburden the VMPU. Figure 1 shows a simplified block diagram of the VMPU.

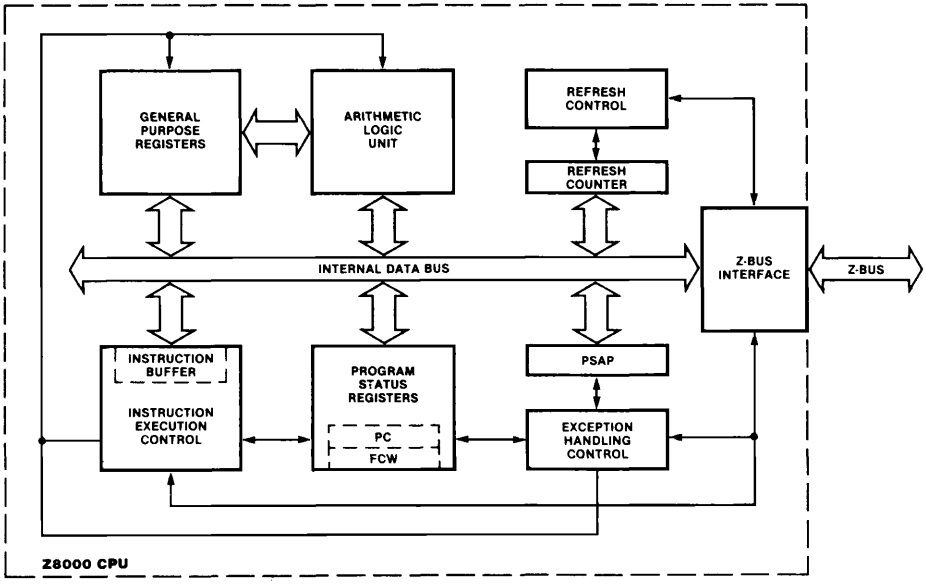


Figure 1. Block Diagram

Architecture

General-Purpose Registers. The VMPU is a register-oriented machine that contains sixteen 16-bit general-purpose registers. All general-purpose registers can be used as accumulators and all but one can be used as index registers or memory pointers.

Register flexibility is created by grouping and overlapping multiple register (Figure 2). For byte operations, the first eight 16-bit registers can be treated as sixteen 8-bit registers. The sixteen 16-bit registers can also be grouped in pairs to form eight 32-bit long-word registers. Similarly, the register set can be grouped in quadruples to form four 64-bit registers.

Stacks. VMPUs can use stacks located anywhere in main memory. Call and Return instructions, as well as interrupts and traps, use an implied stack. Two stack pointers are available, the System Stack Pointer and the Normal Stack Pointer. The two stacks separate operating system (System mode) information from application program (Normal mode) information. The user can manipulate the Stack Pointer with any instruction available for register operations because the Stack Pointer is part of the general-purpose register group.

In the Z8003 VMPU, register pair RR14 is the implied Stack Pointer for segmented operation. Register R14 contains the 7-bit segment number and R15 contains the 16-bit offset. Register R15 is used as the Stack Pointer during nonsegmented operation. Since the Z8004 runs only in the nonsegmented mode, register R15 is used as the Stack Pointer.

Special-Purpose Registers. The VMPUs also provide 16-bit special-purpose registers. These registers include Program Status registers, Program Status Area Pointer registers(s), and a Refresh Counter. The configurations of the special-purpose

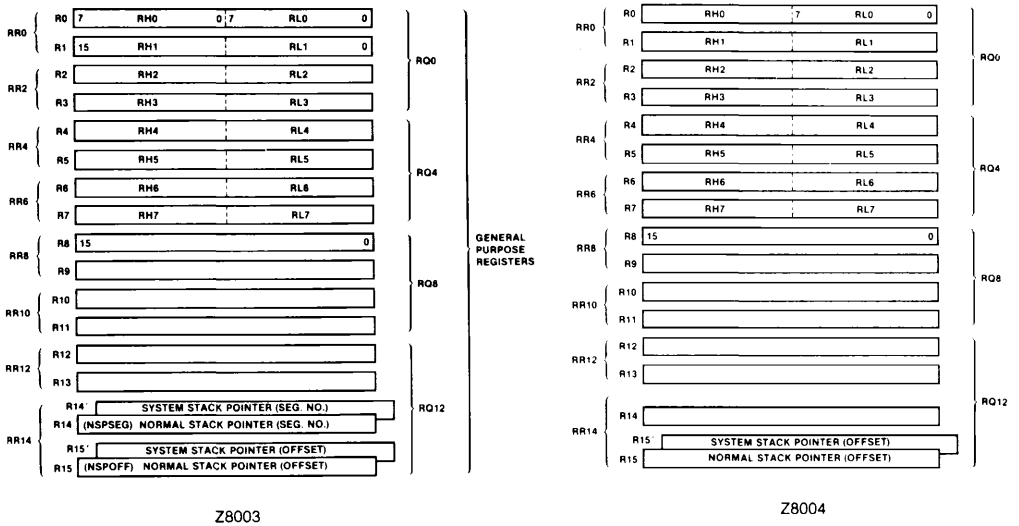


Figure 2. VMPU General-Purpose Registers

Architecture (Continued)

registers for the Z8003 and Z8004 VMPUs are shown in Figure 3.

Program Status Registers. This group of registers consists of the Program Counter (PC) register and the Flag and Control Word (FCW) register. The PC register contains the address of the next instruction to be loaded into the CPU. The low-order byte of the FCW register contains the following flags:

C, Carry flag, is used to indicate that a carry was made out of the high-order bit position of a register used as an accumulator.

Z, Zero flag, is generally used to indicate that the result of an operation was zero.

S, Sign flag, is generally used to indicate that the result of an operation was negative.

P/V, Parity/Overflow flag, is generally used to indicate either even parity (after logical operations on byte operands) or an overflow condition (after arithmetic operations).

D, Decimal-Adjust flag, is used in BCD arithmetic to indicate the type of instruction that was executed (addition or subtraction).

H, Half Carry flag, is used to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result.

The high-order byte of the FCW register contains control bits which are used to control the VMPU operating modes and to enable various types of interrupts. The following control bits are contained in the FCW:

NVIE, Nonvectored Interrupt Enable bit. This but must be 1 to enable the VMPU to accept non-vectored interrupts.

VIE, Vectored Interrupt Enable bit. This bit must be 1 to enable the VMPU to accept vectored interrupts.

S/$\overline{N}$, System/Normal bit. This bit indicates the current VMPU operating mode. When 0, S/N specifies Normal mode; When 1, S/$\overline{N}$ specifies System mode. The VMPU output N/S represents the complement of this bit.

EPA, Extended Processor Architecture mode bit. This bit, when 1, indicates that the system contains an Extended Processing Unit (EPU) and extended instructions are to be executed by the appropriate EPU. When 0, this bit specifies that extended instructions will be trapped for software emulation.

SEG, Segmentation mode bit (Z8003 only). When 1, this bit specifies that the VMPU is in segmented addressing mode; when 0 it specifies that the VMPU is in the nonsegmented addressing mode.

Program Status Area Pointer (PSAP) Register. A Program Status Area (PSA) array in main memory is used to store new program status information (i.e., sets of FCW and PC values). Each time an interrupt or trap occurs the current program status is saved and a new program status is loaded into the status registers from the Program Status Area. The address of the table that contains new program status values is contained in a Program Status Area Pointer (PSAP) register (Figure 4). The low order byte of the offset address is assumed to be all zeros; therefore, the Program Status Area must start on a 256-byte boundary.

Refresh Register. The VMPU contains a programmable counter that automatically refreshes dynamic memory. The Refresh Counter register consists of a 9-bit row counter, a 6-bit rate counter, and an Enable bit (Figure 5). The 9-bit row counter can address up to 256 rows and is incremented by two each time the rate counter reaches end-of-count. The rate counter determines the time between successive refreshes. It consists of a programmable, 6-bit modulo-n prescaler (n = 1 – 64), driven at one-fourth the VMPU clock rate. Refresh can be disabled by programming the refresh Enable/Disable bit. If this register is not needed for memory refresh, it can function as an on-board internal timer.

Architecture (Continued)

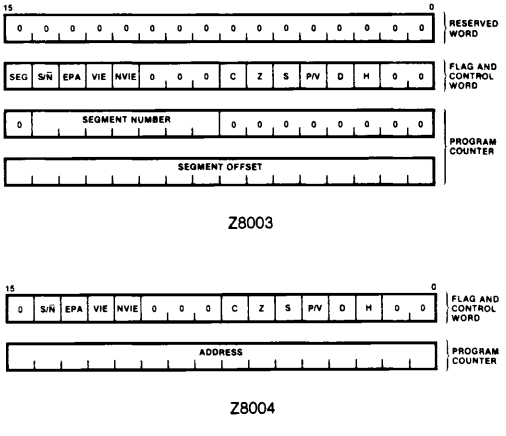


Figure 3. Program Status Registers

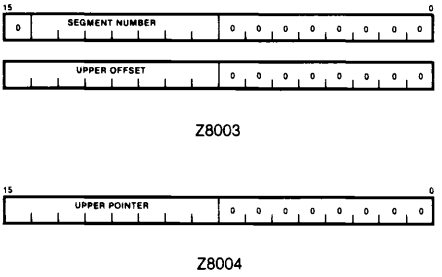


Figure 4. Z8003 Program Status Area Pointer (PSAP) Registers

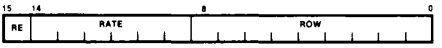


Figure 5. Refresh Register/Counter

System and Normal Modes

The VMPUs can run in either System or Normal mode. In System mode, all instructions can be executed and all VMPU control registers can be accessed. This mode is useful in programs that perform operating system functions.

In Normal mode, some instructions, such as the I/O instructions, cannot be executed. In addition, the VMPU control registers cannot be accessed. This mode is intended for use by application (user) programs.

The use of separate VMPU System and Normal modes promotes the integrity of the system by preventing user programs from having access to the operating system and the control registers. The current operating mode is specified by the S/N bit of the FCW register. The complement of the state of this bit is output by the VMPU on line N/S. Output N/S can be used to separate System and Normal address spaces.

Address Spaces

Programs and data can be located in the main memory of the computer system or in peripheral devices. In either case, the location of the information must be specified by an address before that information can be accessed. A set of these addresses is called an address space.

The VMPUs support two different types of addresses and thus two categories of address space:

- Memory addresses, which specify locations in main memory.
- I/O addresses, which specify the ports through which peripheral devices are accessed.

Within the two general types of address spaces (memory and I/O), there are several

Address Spaces (Continued)

subcategories. Figure 6 shows the address spaces that are available on both types of VMPUs.

The difference between the Z8003 and the Z8004 VMPUs lies not in the number and type of address spaces, but rather in the organization and size of each space. For the Z8003, the memory address space contains 8M bytes of addresses grouped into 128 separate segments. For the Z8004, the memory space is a homogeneous collection of 64K bytes of addresses. In both the Z8003 and the Z8004, each I/O address space contains 32K byte port addresses and 64K word Port addresses.

When an address is used to access data, the address spaces can be distinguished by the state of the status lines (ST_0-ST_3) and by the value of the Normal/System line (N/S). The states of the four status lines are determined by the way the address was generated. The value of the N/S output line is the complement of the S/N control bit in the FCW register.

The 23-bit segmented addresses are divided into 7-bit segment identifiers (segment numbers) and 16-bit offsets to address locations relative to the beginning of the specified segment. In hardware, segmented addresses are contained in a register pair or in a long-word memory location. The segment number and offset of an address can be manipulated separately or together by all available word and long word operations.

In an instruction, a segmented address can have one or two representations; long-offset or short-offset. A long-offset address occupies two words, with the first word containing the 7-bit segment number and the second word containing the 16-bit offset. A short-offset address requires only one word, which combines the 7-bit segment number with an 8-bit offset (range 0-256). The short-offset mode allows very dense encoding of addresses and minimizes the need for long addresses to directly access each 8M byte address space.

Nonsegmented addresses are 16 bits and permit access of up to 64K of contiguous byte locations.

The Z8004 operates only in the nonsegmented address mode. The Z8003 can operate in either the segemented or nonsegmented address mode. When the Z8003 is in nonsegmented mode, all address representations assume implicitly the segment number contained in the 7-bit segment number field of the PC.

I/O Addresses. There is a set of I/O instructions that perform 8-or 16-bit transfers between a VMPU and its I/O devices. I/O devices are addressed with 16-bit I/O port addresses. An I/O port address is similar to a memory address; however, the I/O address space is not part of the memory address space. Memory-mapped I/O can be implemented by dedicating memory locations to I/O device registers. Two types of I/O instruction are available. Standard and Special. Each type has its own address space. Special I/O instructions are used for loading and unloading memory management units.

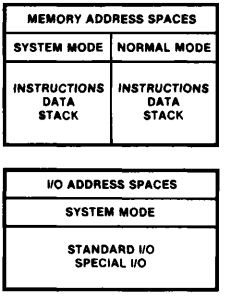


Figure 6. Address Spaces on the Z8003 and Z8004

Instruction Addressing Modes

The information included in VMPU instructions consists of the function to be performed, the type and size of data elements to be manipulated, and the locations of the data elements. Locations are designated by register addresses, memory addresses, or I/O addresses. The addressing mode of a given instruction defines the method used to compute the address. Addressing modes are explicitly specified or implied by the instruction. Locations are designated using one of the following addressing modes:

- **Register Mode (R).** The data element is located in one of the 16 general-purpose registers or a control register.
- **Immediate Mode (IM).** The data element is located in the instruction.
- **Indirect Register Mode (IR).** The data element can be found in the location whose address is given in a specified register.
- **Direct Address Mode (DA).** The data element can be found in the location whose address is given in the instruction.
- **Index Mode (X).** The data element can be found in the location whose address is the sum of the contents of an index value in a specified register and an address in the instruction.
- **Relative Address Mode (RA).** The data element can be found in the location whose address is the sum of the contents of the Program Counter and a diplacement given in the instruction.
- **Base Address Mode (BA).** The data element can be found in the location whose address is the sum of a base address in a specified register and a displacement given in the instruction.
- **Base Index Mode (BX).** The data element can be found in the location whose address is the sum of a base address in one specified register and an index value in a second specified register.

Instruction Set

Major Groups. The major groups of instructions provided by the VMPU are described in the following paragraphs. A detailed summary of the instructions is presented in Table 3 (located at the back of this document).

Load and Exchange. These instructions move data among registers or between registers and main memory.

Arithmetic. These instructions perform integer arithmetic. The basic instructions (e.g., add, subtract, multiply and divide) in this group use standard two's complement binary format. Support is also provided for implementing BCD arithmetic.

Logical. These instructions perform logical operations (i.e., AND, OR, XOR, and complementation) on the bits of specified operands. The operands can be bytes or words. The Test Long (TESTL) instruction, however, permits logical operations to be performed on 32-bit quantities.

Program Control. These instructions affect the Program Counter, thereby controlling program flow.

Bit Manipulation. These instructions manipulate individual bits in registers or main memory.

Rotate and Shift. These instructions shift and rotate the contents of registers.

Block Transfer and String Manipulation. These instructions perform string comparisons, string translations, and block transfer functions.

Input/Output. These instructions transfer bytes, words or blocks of data between peripheral devices and the VMPU registers or main memory.

Instruction Set (Continued)

VMPU Control. These instructions modify VMPU control and status registers or perform those functions that do not fit into any of the preceding instruction groups.

Extended. These instructions perform Extended Processor Unit (EPU) internal operations, data transfers between memory and EPU, data transfers between EPU and the VMPU, and data transfers between EPU flag registers and the VMPU Flag And Control Word (FCW).

Processor Flags. The processor flags contained by the program status registers provide a link between sequentially executed instructions. The link is provided in the sense that the result of executing one instruction may alter one or more flags. The new flag values (states) can then be used to determine the operation of a subsequent instruction (typically a conditional jump instruction). The following six flags are available for use by the programmer and the processor:

- Carry (C)
- Zero (Z)
- Sign (S)
- Parity/Overflow (P/V)
- Decimal-Adjust (D)
- Half Carry (H)

Condition Code. Flags C, Z, S, and P/V are used to control the operation of conditional instructions (such as Conditional Jump). The operations performed by this type of instruction depend on whether or not a specified Boolean condition exists on the four flags. Sixteen functions of the flag settings found to be frequently used are encoded in a 4-bit condition code (CC) field, which forms a part of all conditional instructions. These 16 codes are described in Table 1.

Code	Meaning	Flag Settings	CC Field Binary	CC Field Hex
F	Always false	—	0000	0
T	Always true	—	1000	8
Z	Zero	Z = 1	0110	6
NZ	Not zero	Z = 0	1110	E
C	Carry	C = 1	0111	7
NC	No carry	C = 0	1111	F
PL	Plus	S = 0	1101	D
MI	Minus	S = 1	0101	5
NE	Not equal	Z = 0	1110	E
EQ	Equal	Z = 1	0110	6
OV	Overflow	P/V = 1	0100	4
NOV	No overflow	P/V = 0	1100	C
PE	Parity is even	P/V = 1	0100	4
PO	Parity is odd	P/V = 0	1100	C
GE	Greater than or equal (signed)	(S XOR P/V) = 0	1001	9
LT	Less than (signed)	(S XOR P/V) = 1	0001	1
GT	Greater than (signed)	|Z OR (S XOR P/V)| = 0	1010	A
LE	Less than or equal (signed)	|Z OR (S XOR P/V)| = 1	0010	2
UGE	Unsigned greater than or equal	C = 0	1111	F
ULT	Unsigned less than	C = 1	0111	7
UGT	Unsigned greater than	|(C = 0) AND (Z = 0)| = 1	1011	B
ULE	Unsigned less than or equal	(C OR Z) = 1	0011	3

Note: Some condition codes have identical flag settings and binary fields in the instruction, i.e., Z = EQ, NZ = NE, C = ULT, NC = UGE, OV = PE, NOV = PO.

Table 1. Condition Codes

Multi-Microprocessor Resource Control

The Z8003 and Z8004 VMPUs include both hardware and software support for controlling access to shared resources in multi-microprocessor systems. VMPU pins $\overline{MI}$ (Multi-Micro In) and $\overline{MO}$ (Multi-Micro Out) and instructions MSET (Set $\overline{MO}$), MREQ (access request), MBIT (Test $\overline{MI}$), and MRES (reset $\overline{MO}$) can be used to form a prioritized resource access control system. Such a system would, for a VMPU, 1) issue requests for access to a shared resource, 2) test the access status for the resource (available/not available) and 3) when access is granted, exclude all other VMPUs in the system from the resource until use of the resource is complete.

Test and Set Instruction (TSET)

The TSET instruction implements synchronization mechanisms in multiprogramming and multiprocessing environments. TSET tests and sets semaphores that control access to shared resources. The testing and setting of a semaphore requires the semaphore to be read from memory, modified, then written back into the same memory location. To prevent other processors from requesting access to a resource during a test and set process, status code 1111 is placed onto status lines ST_0-ST_3 during the data read transaction to specify that an uninterruptable memory operation is taking place. Status code 1111 is particularly useful in a multiple microprocessor environment to permit external circuitry to preclude memory access by another device between the read transaction and the write transaction of the test and set operation. Request input BUSREQ is also disabled during a test and set operation to ensure that the test and set operation is not interrupted; this action is useful in a single-processor system.

Extended Processing Architecture

The VMPU has an Extended Processing Architecture (EPA) facility which extends the basic functions of the VMPU by using external devices called Extended Processing Units (EPUs). A special set of extended instructions controls the operations to be performed by each EPU. When a VMPU encounters an extended instruction, it either traps the instruction, or it performs the data transfer portion of the instruction. The data manipulation portion of the instruction is executed by the involved EPU. Whether the VMPU traps or transfers data depends on the setting of an EPA bit in its Flag and Control Word (FCW) status register.

Exceptions

The Z8003 and Z8004 VMPUs support four types of exceptions (conditions that alter the normal flow of program execution): interrupts, traps, instruction aborts, and reset.

Interrupt and Trap Structure. The Z8003 and Z8004 VMPUs have a flexible and powerful interrupt and trap structure. Interrupts are external events requiring VMPU attention and are generally triggered by peripherals needing service. Traps are synchronous events resulting from the execution of certain instructions.

Both Z8003 and Z8004 VMPUs support three interrupts: nonmaskable ($\overline{NMI}$),

Exceptions (Continued)

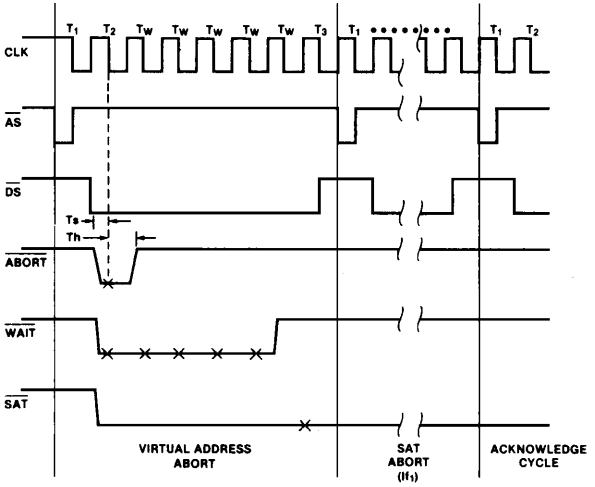
vectored ($\overline{VI}$), and nonvectored ($\overline{NVI}$).

Both VMPUs support several types of traps: System Call, EPU instruction, and privileged instruction. In addition, the Z8003 supports a Segment/Address Translation ($\overline{SAT}$) trap. Of the above traps, only the last is initiated by external events. Such events are normally generated by a memory management system. The remaining traps occur when instructions limited to the System mode are used in the Normal mode, when a System Call instruction is executed, or when an EPA instruction is encountered.

The descending order of priority for traps and interrupts is: internal traps, nonmaskable interrupts, segment/address translation traps, vectored interrupts, and nonvectored interrupts.

When an interrupt or trap occurs, the current program status information is automatically pushed onto the System stack. The new program status is then automatically loaded into the Program Status registers from the Program Status Area in System program memory. This area of memory is identified by the Program Status Area Pointer (PSAP).

Instruction Abort Function. The VMPU monitors its $\overline{ABORT}$ input during each bus transaction it generates. The timing for an Instruction Abort operation is shown in Figure 7. If the $\overline{ABORT}$ input is asserted during clock cycle T_2 of a memory access, the currently executing instruction is automatically aborted. If no abort is indicated but input $\overline{WAIT}$ is asserted, input $\overline{ABORT}$ is also tested during each wait cycle (T_W). When an Instruction Abort condition is indicated (ABORT is asserted) the $\overline{WAIT}$ input must also be asserted for five cycles to permit the VMPU internal control mechanism to abort the current instruction. When the $\overline{WAIT}$ input is deasserted, the VMPU acknowledges any pending interrupt request. Therefore, the memory management circuitry that caused the interrupt to be aborted should also request an interrupt to the software routine that restores the VMPU registers and the main memory so that the aborted instruction can be reissued.



NOTE: * = Clock Sample Points

Figure 7. Instruction Abort Timing

Virtual Memory Systems

Virtual memory systems permit programs to reference an address space that exceeds the main (physical) memory. In virtual memory systems, high-speed main memory is supported by medium- and low-speed storage devices (secondary memory) such as hard disks or floppy disks. When a VMPU in a virtual system issues an address that references information not in main memory, a software swap operation must be initiated.

The swap retrieves the block containing the referenced location, loads it into main memory, and restarts the aborted mainstream program at the point of interruption.

The swap operation is transparent to the user and to the executing program; therefore, the system appears to have a memory that is not constrained by physical size.

The maximum size of a virtual memory is determined by the address structure used and by the capabilities of the system memory management hardware and software.

Segmented and Paged Virtual Memories. External circuitry can be used to implement either a segmented virtual memory or a paged virtual memory. In a segmented virtual memory, information is transferred between main memory and secondary storage devices on a segment-by-segment basis. The Z8003 VMPU permits use of variable-length segments of up to 64K bytes.

In a paged virtual memory system, each segment is divided into fixed-size pages (standard size is 2048 bytes). Main memory is divided into page "frames." Information is then transferred between main memory and the secondary storage devices on a page-by-page basis.

The Z8003 VMPU can support both segmented or paged virtual memory systems. The Z8004 supports only the paged virtual memory approach.

External Hardware Support. The detection of a logical address that references a location outside main memory (i.e., an addressing fault) and the initiation of the required swap operation must be performed by memory management circuitry external to the VMPU.

A swap operation is started by the initiation of a Segment/Address Translation ($\overline{SAT}$) trap request function in the VMPU. Since the Z8004 does not have a $\overline{SAT}$ input, one of the $\overline{NMI}$, $\overline{VI}$ or $\overline{NVI}$ inputs must be used instead. Low levels on VMPU inputs $\overline{ABORT}$, $\overline{SAT}$ and $\overline{WAIT}$ initiate $\overline{SAT}$ requests.

These inputs are sampled at the falling clock of the second clock cycle of a bus transaction. Input $\overline{WAIT}$ must be asserted for at least five clock cycles. Input $\overline{ABORT}$ must be deasserted on or before the rising edge of the $\overline{WAIT}$ signal. The same timing can be used for both $\overline{WAIT}$ and $\overline{ABORT}$. Input $\overline{SAT}$ should be asserted until the trap acknowledge bus transaction is indicated by Z8003 VMPU status code 0100.

External circuitry is needed to record the information for instruction restart. The following assumptions about the operating system must also be true:

- The fault handler does not generate a fault until all critical data is saved.
- Accessing the System stack never causes a fault. (Either the segment is in memory or a memory management mechanism warns of a potential stack overflow).
- I/O buffers are always in main memory, so I/O instructions never cause a fault.
- The Program Status Area is always in main memory.

The following information must be saved by external circuitry to restart the instruction interrupted by the addressing fault:

- The value of the Program Counter during the initial instruction fetch cycle (cycle identified by status code 1101).
- The address that caused the fault.
- The code that was on the status lines during the aborted cycle.
- For paged memories, the number of successful data accesses made by the instruction.

Virtual Memory Systems (Continued)

Software Support. The software required for virtual memory operation normally consists of a fault handler and a restart routine. The fault handler is started during each VMPU abort request operation. The fault handler is responsible for saving information about the aborted instruction and for the initiation of a request which brings the segment (or page) containing the referenced location in main memory. The state of the aborted program (Flag and Control Word (FCW), Program Counter (PC), and the register file must be saved and another process dispatched while the missing segment (or page) is being fetched from secondary memory.

When the page or segment containing the referenced location is loaded into main memory, an instruction restart routine must be executed. This instruction restart routine must restore the operating environment that existed when the instruction/program abort was initiated.

This routine must establish the PC value that points to the aborted instruction. It must also decode the instruction's opcode to determine whether or not any of the VMPU's registers were modified before the instruction execution cycle in which the abort occurred. If registers were modified, the instruction restart routine must return these registers to a state in which the restarted instruction behaves as if no abort had occurred. The flow chart in Figure 8 illustrates a possible control sequence for a software restart routine. The instructions requiring remodification of system registers and the manner in which these registers must be modified depend upon the type (segmented or paged) of virtual memory system implemented.

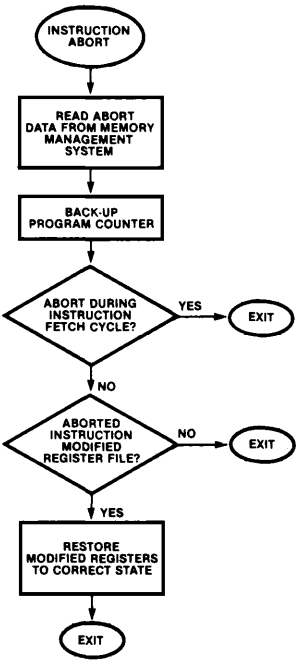


Figure 8. Flow Chart of an Instruction Restart Routine

Bus Transactions

Status Outputs. The VMPUs provide output that specifies the type of transaction on the Address/Data bus. Output line R/$\overline{W}$ specifies whether a read or write operation is involved. Output line B/$\overline{W}$ specifies whether the transaction involves byte or word data. Output line N/$\overline{S}$ specifies the mode of operation, Normal or System. In addition to these lines, output lines ST_0-ST_3 encode additional characteristics of the current bus transaction. These lines can present any of sixteen 4-bit status codes which define specific characteristics of the current bus transaction. The available status codes are listed and defined in Table 2.

Bus Transactions (Continued)

ST_3-ST_0 Binary	Definition
0 0 0 0	Internal Operation
0 0 0 1	Memory Refresh
0 0 1 0	I/O Reference
0 0 1 1	Special I/O Reference (e.g., to an MMU)
0 1 0 0	Segment/Address Translation Trap Acknowledge
0 1 0 1	Nonmaskable Interrupt Acknowledge
0 1 1 0	Nonvectored Interrupt Acknowledge
0 1 1 1	Vectored Interrupt Acknowledge
1 0 0 0	Data Memory Request
1 0 0 1	Stack Memory Request
1 0 1 0	Data Memory Request (Extended Processing Architecture)
1 0 1 1	Stack Memory Request (Extended Processing Architecture)
1 1 0 0	Instruction Space Access
1 1 0 1	Instruction Fetch, First Word
1 1 1 0	Extended Processing Unit-VMPU Transfer
1 1 1 1	Bus Lock, Data Memory Request

Table 2. Status Codes

Memory Read and Write. Memory read and instruction fetch cycles are identical, except for the status code on the $\overline{ST}_0$-$\overline{ST}_3$ outputs.

Memory write is similar to memory read except for the R/$\overline{W}$ status and the timing of $\overline{DS}$ and data valid true. During a memory cycle, a 16-bit offset address is placed on the AD_0-AD_{15} outputs early in the first clock period (Figura 9). In the Z8003, a 7-bit segment number is also output on SN_0-SN_6 one clock period earlier than the 16-bit address offset. Issuing the segment number early minimizes address translation overhead by enabling the memory management circuitry to overlap its operations with the VMPU instruction execution cycle.

A valid address is indicated by the rising edge of Address Strobe ($\overline{AS}$). Status and mode information becomes valid early in the memory access cycle and remains stable throughout it. The access cycle can be extended in length by the addition of wait cycles.

The Read/Write line (R/$\overline{W}$) indicates the direction of the data transfer. R/$\overline{W}$ is High for transfers to the VMPU. R/$\overline{W}$ is Low for transfers from the VMPU.

Word data (B/$\overline{W}$ is Low) to or from the VMPU is transmitted on lines AD_0-AD_{15}. Byte data to the VMPU is transmitted in AD_{10}-AD_7, from odd addresses ($AD_0 = 1$) and in AD_8-AD_{15} from even addresses ($AD_0 = 0$). Byte data from the VMPU is replicated in AD_0-AD_7 and AD_8-AD_{15}, regardless of address.

I/O Transactions. I/O transactions, which are generated by the execution of I/O instructions, move data to or from peripherals or VMPU support devices. As shown in the timing diagram presented in Figure 10, I/O transactions have a minimum length of four clock cycles; wait cycles can be added to lengthen transaction periods to meet the needs of slow peripherals. Status line outputs indicate whether access is to the Standard I/O (0010) or Special I/O (0011) address spaces.

I/O transactions are always performed with the VMPU in System mode (N/$\overline{S}$ = Low). The rising edge of $\overline{AS}$ indicates that a valid address is present on lines AD_0-AD_{15}. Since the I/O address is always 16 bits long, the segment number lines in Z8003 are undefined.

For byte transfers (B/$\overline{W}$ = High) in Standard I/O space, addresses must be odd; for byte transfers in Special I/O space, addresses must be even.

Word data (B/$\overline{W}$ = Low) to or from the CPU is transmitted on AD_0-AD_{15}. Byte data (B/$\overline{W}$ = High) is transmitted on AD_0-AD_{15} for Special I/O. This allows peripheral devices or CPU support devices to attach to only eight of the 16 AD_0-AD_{16} lines. The Read/Write line (R/$\overline{W}$) indicates the direction of the data transfer: peripheral-to-CPU (Read: R/$\overline{W}$ = High) or CPU-to-peripheral (Write: R/$\overline{W}$ = Low).

Bus Transactions (Continued)

Wait Add-On Cycles. As shown in Figures 9 and 10, the $\overline{WAIT}$ input line is sampled on a falling edge of CLK one cycle before data is sampled ($\overline{DS}$ is Low for a read or write operation). If the $\overline{WAIT}$ input line is Low when sampled, another cycle is added to the transaction before data is sampled or $\overline{DS}$ is deasserted (goes High). During an added wait cycle. Input $\overline{WAIT}$ is sampled again on the falling clock edge: if it is Low, another wait cycle is added to the transaction.

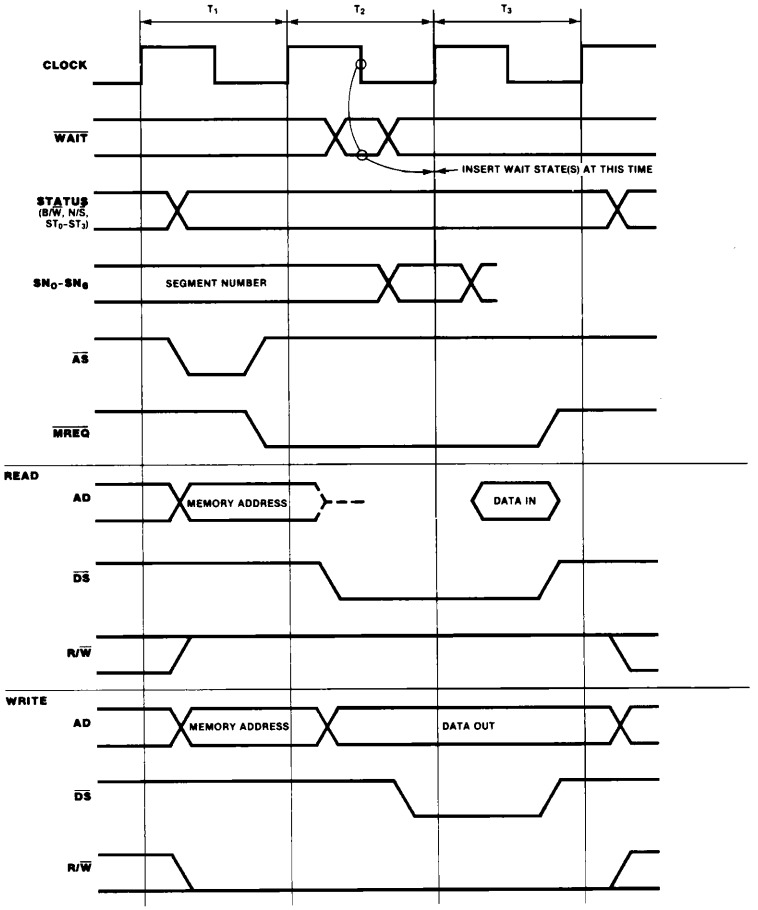


Figure 9. Memory Read and Write Timing

Bus Transactions (Continued)

This use of the $\overline{\text{WAIT}}$ input permits transactions to be extended arbitrarily to accommodate, for example, slow memories or I/O devices that are not yet ready for data transfer.

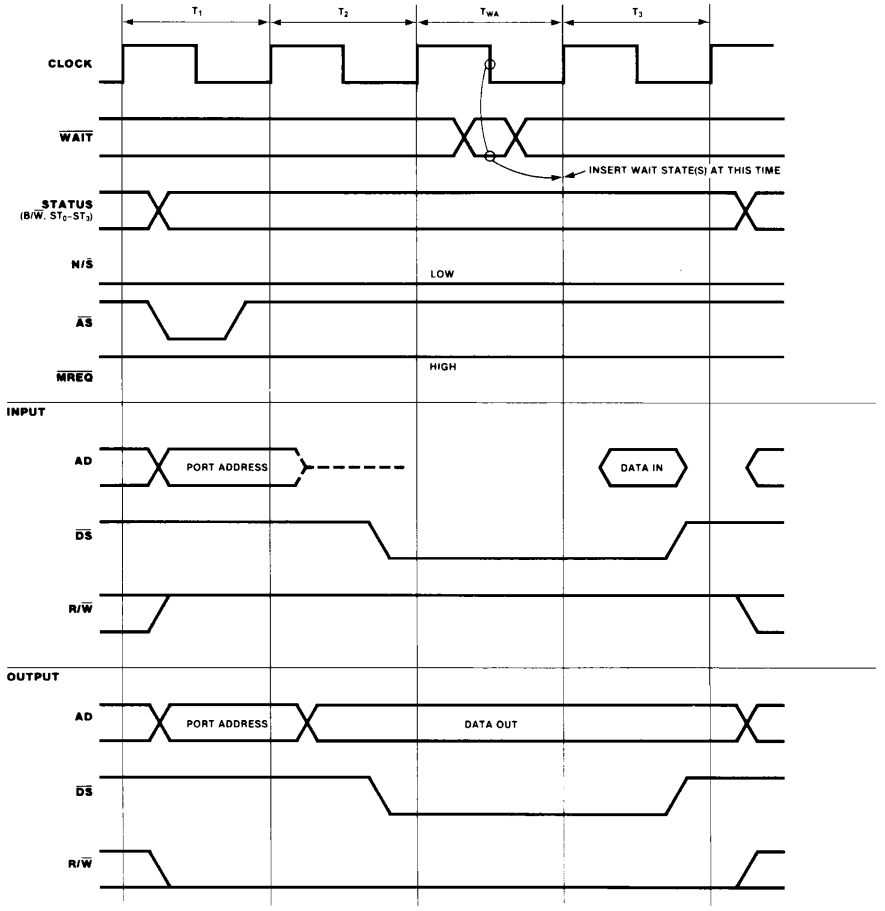


Figure 10. Input/Output Transaction

Bus Transactions (Continued)

Memory Refresh Timing. When the 6-bit prescaler in the refresh counter has been decremented to zero, a refresh cycle is started (Figure 11). The 9-bit refresh counter value is put on AD_0-AD_8; lines AD_9-AD_{15} are undefined. Unless disabled, the presettable prescaler runs continuously, therefore any delay in starting a refresh cycle is not cumulative.

While the $\overline{STOP}$ input is Low, a continous stream of memory refresh cycles is executed without using the refresh prescaler. The refresh count, however, is incremented.

Internal Operation Timing. Certain instructions, such as multiply and divide, need additional time to execute internal operations. In these cases, the VMPU goes through a sequence of internal operation machine cycles, each threee to eight clock cycles long (Figure 12). This allows fast response to bus and refresh requests because a bus request or a refresh cycle can be inserted at the end of any internal machine cycle.

Although the address outputs during clock cycle T_1 are undefined. Address Strobe ($\overline{AS}$) is generated to satisfy the requirements of Z-BUS-compatible peripherals and self-refresh dynamic memories.

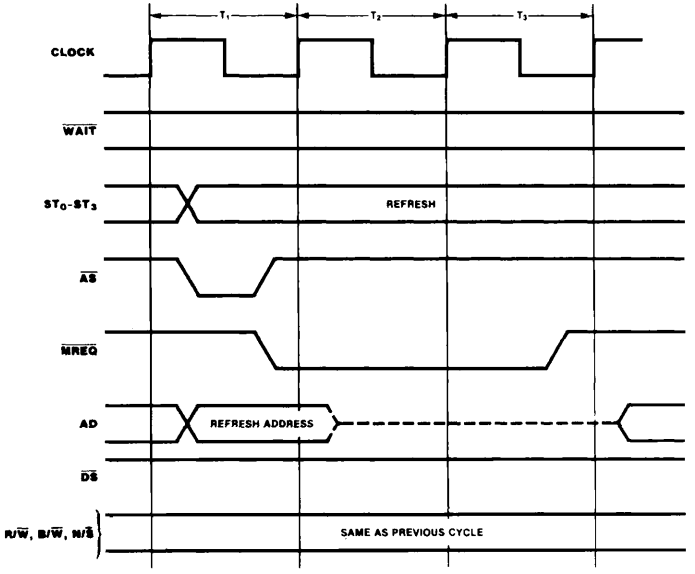


Figure 11. Memory Refresh Timing

Bus Transactions (Continued)

Reset Function. A Low on the $\overline{\text{RESET}}$ input causes the following results within five clock cycles (Figure 13):

1. AD_0-AD_{15} are 3-stated.
2. $\overline{AS}$, $\overline{DS}$, $\overline{MREQ}$, $\overline{BUSACK}$, $\overline{MO}$, and ST_0-ST_3 are forced High.
3. SN_0-SN_6 are forced Low.
4. Refresh is disabled.
5. R/$\overline{W}$, B/$\overline{W}$ and N/$\overline{S}$ are undefined.

When $\overline{\text{RESET}}$ is again High, the Z8003 VMPU executes three memory read cycles in a System mode of operation. During these three word read cycles, the VMPU reads, in sequence, the following information from segment 0:

1. The flag and control word (FCW) from offset location 0002.
2. The Program Counter segment number from location 0004 and offset from location 0006.

In the Z8004 VMPU, only two read cycles are performed. During the first cycle, the FCW is read from location 0002. During the second cycle, the 16-bit PC value is read from location 0004. The program is started during the following machine cycle.

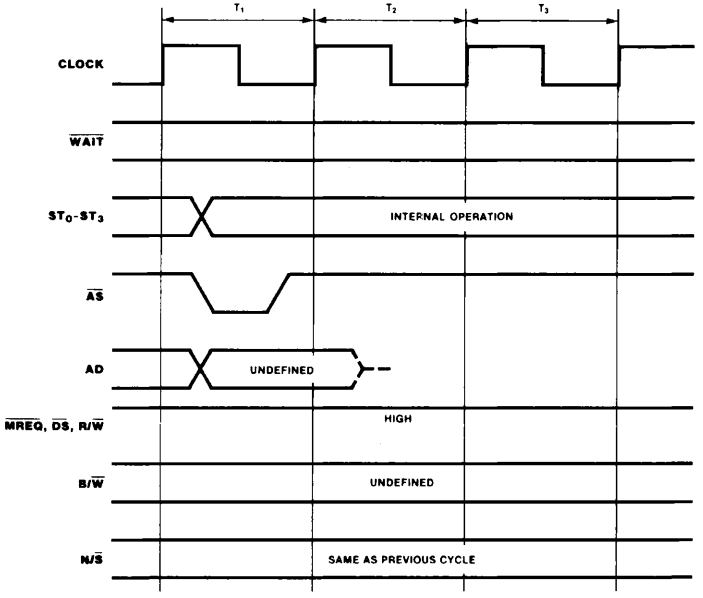


Figure 12. Internal Operating Timing

Bus Transactions (Continued)

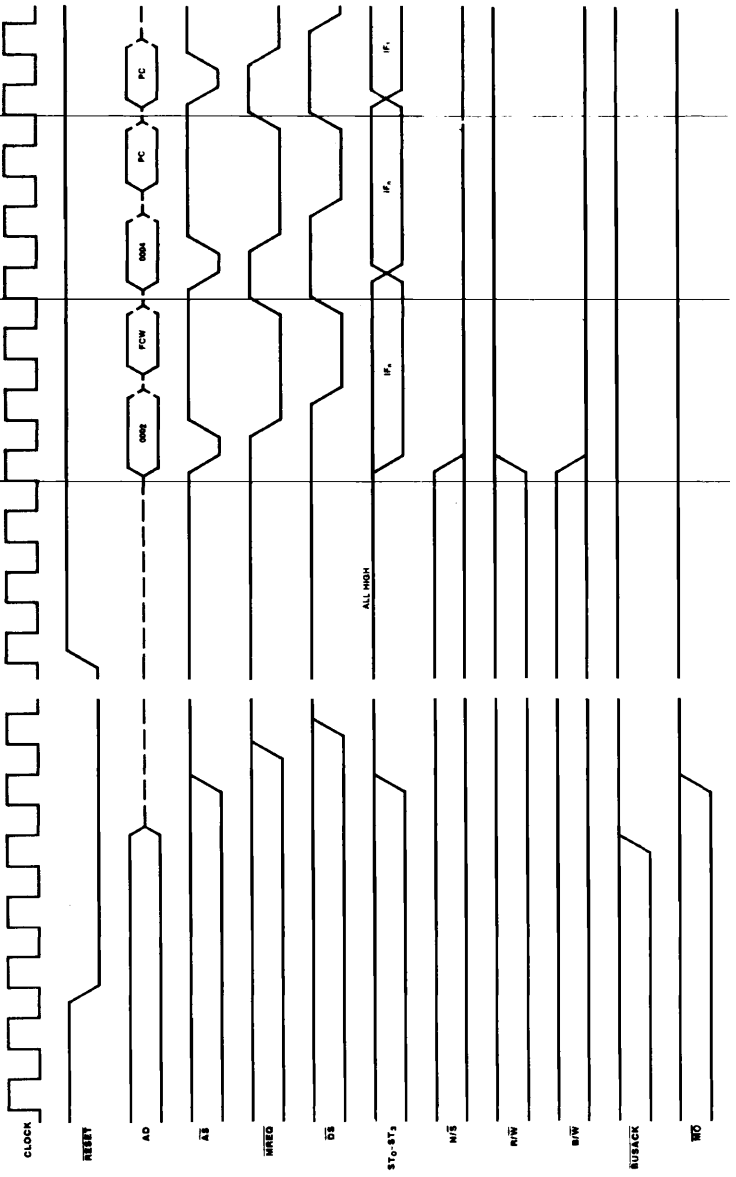


Figure 13. Reset Timing

Bus Request, Interrupt and Acknowledge

A low on the $\overline{\text{BUSREQ}}$ input indicates to the VMPU that another device is requesting the address/data and control lines. The asynchronous $\overline{\text{BUSREQ}}$ input is synchronized at the beginning of any machine cycle (Figure 14). If $\overline{\text{BUSREQ}}$ is Low, an internal synchronous $\overline{\text{BUSREQ}}$ signal is generated, which, after completion of the current machine cycle, causes the $\overline{\text{BUSACK}}$ output to go Low and all bus outputs to go into the high-impedance state. The requesting device (typically a DMA) can the controll the bus.

When $\overline{\text{BUSREQ}}$ is released, it is synchronized with the rising clock edge. The $\overline{\text{BUSACK}}$ output goes High one clock period later to indicate that the VMPU will take control of the bus.

Interrupt and Segment/Address Translation Trap Request and Acknowledge. Any High-to-Low transition on the VMPU's $\overline{\text{NMI}}$ input (Figure 15) is asynchronously edge-detected and sets the internal $\overline{\text{NMI}}$ latch. The $\overline{\text{VI}}$, $\overline{\text{NVI}}$, and $\overline{\text{SAT}}$ inputs, as well as the state of the internal $\overline{\text{NMI}}$ latch, are sampled at the beginning of T_3.

In response to an interrupt or trap, the subsequent IF_1 cycle is exercised. The Program Counter, however, is not updated, but the System Stack Pointer is decremented in preparation for storing status information on the System stack.

The next machine cycle is the interrupt acknowledge cycle. This cycle has five automatic wait states and additional wait states are possible.

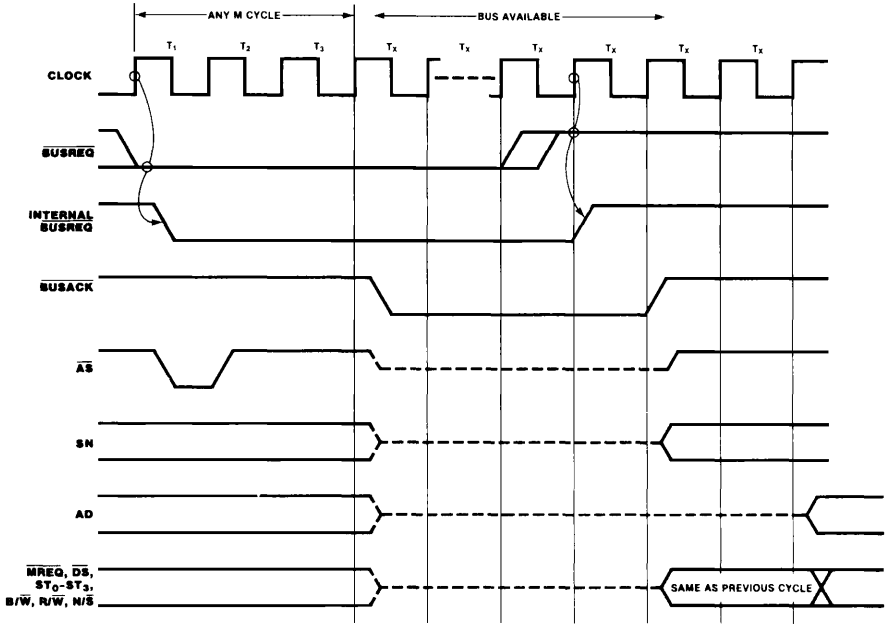


Figure 14. Bus Request/Acknowledge Timing

Bus Request, Interrupt and Acknowledge (Continued)

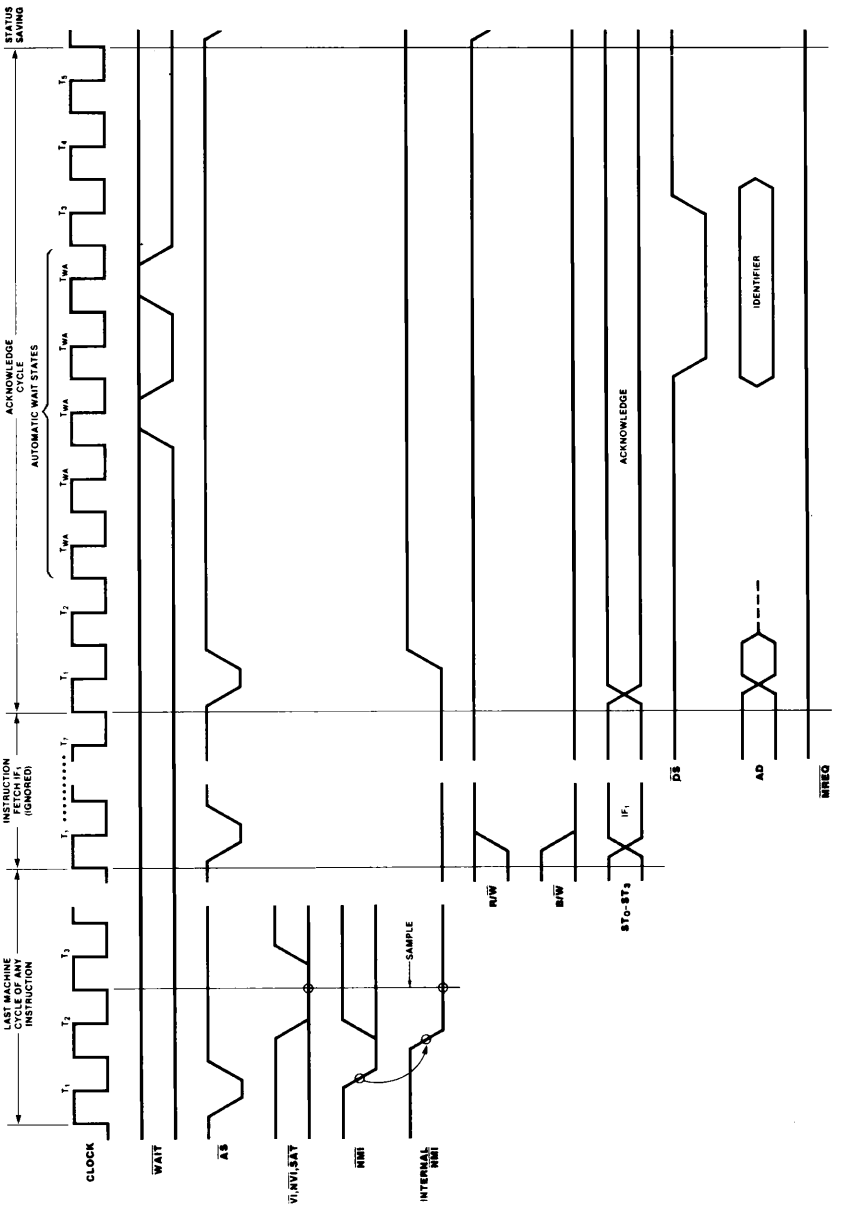


Figure 15. Interrupt and Segment/Address Translation Trap, Request/Acknowledge Timing

Bus Request, Interrupt and Acknowledge (Continued)

After the last wait state, the VMPU reads the information on AD_0-AD_{15} and stores it temporarily, to be saved on the stack later in the acknowledge sequence. This word identifies the source of the interrupt or trap. For internal traps, the identifier is the first word of the trapped instruction. For external events, the identifier is the contents of the Data bus as sampled during T_3 of the acknowledge cycle. During nonvectored and non-maskable interrupts, all 16 bits can represent peripheral device status information. For the vectored interrupt, the low byte is the jump vector, and the high byte can be used for extra status. For a $\overline{SAT}$ trap (assuming that a Z8010 MMU Memory Management Unit is used) the high byte is the memory management unit identifier and the low byte is undefined.

After the acknowledge cycle, the $N/\overline{S}$ output indicates the automatic change to System mode.

Pin Descriptions

The Z8003 VMPU is produced in a 48-pin package; the Z8004 VMPU is produced in a 40-pin package. The pin functions of both the Z8003 and Z8004 are illustrated in Figure 16; the pin assignments are illustrated in Figure 17. The signal names assigned to the VMPU I/O pins are listed alphabetically and are described in the following paragraphs.

$\overline{ABORT}$. *Abort Request* (input, active Low). This input is used to implement virtual memory. It is asserted by external circuitry when an address does not correspond to a location in main memory.

When $\overline{ABORT}$ is asserted with input $\overline{SAT}$ in the Z8003, or with input $\overline{NMI}$, $\overline{VI}$, or $\overline{NVI}$ in the Z8004, it initiates an Abort interrupt in the VMPU.

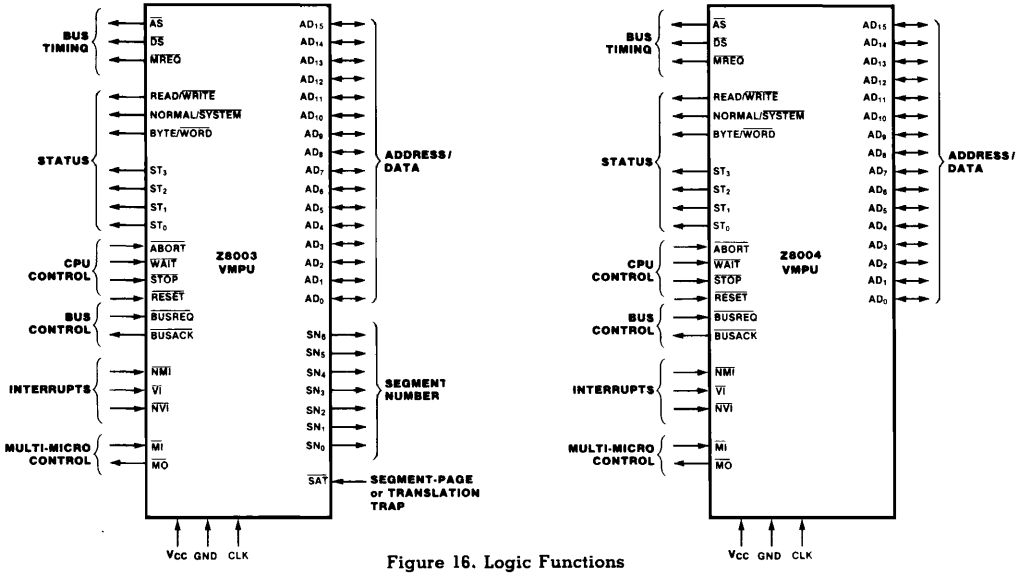


Figure 16. Logic Functions

Pin Descriptions (Continued)

AD_0-AD_{15}. *Address/Data* (inputs/outputs, active High, 3-state). These multiplexed address and data lines are used both for I/O and memory.

$\overline{AS}$. *Address Strobe* (output, active Low, 3-state). The rising edge of $\overline{AS}$ indicates that addresses are valid.

$\overline{BUSACK}$. *Bus Acknowledge* (output, active Low). A Low on this line indicates that the VMPU has relinquished control of the bus.

$\overline{BUSREQ}$. *Bus Request* (input, active Low). This line must be driven Low to request the bus from the VMPU.

B/$\overline{W}$. *Byte/Word* (output, Low = Word, 3-state). This line defines the size of the data being transferred.

CLK. *System Clock* (input). CLK is a + 5 V signle-phase, time-base input.

$\overline{DS}$. *Data Strobe* (output, active Low, 3-state). This line strobe data in and out of the VMPU.

$\overline{MI}$, $\overline{MO}$. *Multi-Micro In, Multi-Micro Out* (input and output, active Low). These two lines form a resource-request daisy chain that allows only one VMPU in a multi-microprocessor system to access a shared resource at the same time.

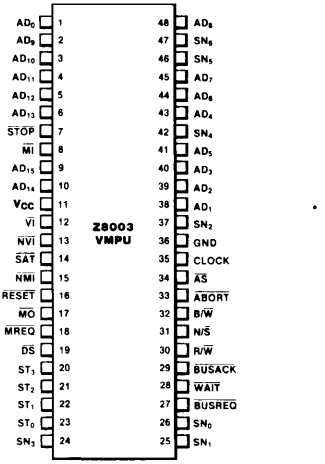
$\overline{MREQ}$. Memory Request (output, active Low, 3-state). A Low on this line indicates that a memory reference is in progress.

$\overline{NMI}$. Nonmaskable Interrupt (edge-triggered, input, active Low). A High-to-Low transition on $\overline{NMI}$ requests a nonmaskable interrupt.

N/$\overline{S}$. *Normal/System Mode* (output, Low = System mode, 3-state). N/$\overline{S}$ indicates the current VMPU operating mode (System or Normal).

$\overline{NVI}$. *Nonvectored Interrupt* (input, active Low). A Low on this line requests a nonvectored interrupt.

$\overline{RESET}$. *Reset* (input, active Low). A Low on this line resets the VMPU.



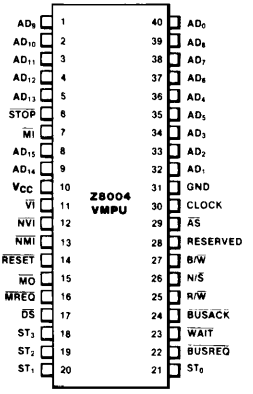


Figure 17. Pin Configuration

Instruction Set Summary

The Z8003/04 instruction set is presented in the instruction set summary. This summary lists the mnemonics, operands, addressing modes, timing, and operation for each instruction.

Timing is given as the number of CPU clock cycles required for instruction execution. Timing requirements are given for the three possible addressing representations used in word, byte and long word operations:

- NS nonsegmented addresses
- SS segmented short-offset adddresses
- SL segmented long-offset addresses

The SS and SL address representations apply only to those instructions for which the address of the operand is contained within the instruction itself. The only instructions of this type are those using the DA and X addressing modes.

With few exceptions, timing requirements are the same for all instructions in either segmented or nonsegmented mode, except for those instructions that employ the SS and SL addresses. The timing for these instructions will differ since the number of fetches needed to load the address, one word or two words, will vary.

Note

Timing values are given in the SS and SL columns of the instruction set summary for all addressing modes, even where the address representation does not apply. These values are given to indicate that the time requirements are the same for both segmented and nonsegmented modes.

Instruction Set Summary

The Z8003/4 provides the following types of instructions:

- Load and Exchange
- Arithmetic
- Logical
- Program Control
- Bit Manipulation
- Rotate and Shift
- Block Transfer and String Manipulation
- Input/Output
- CPU Control

Load and Exchange

Mnemonics	Operands	Addr. Mode	Clock Cycles * Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
CLR **CLRB**	dst	R	7	7	7				**Clear** dst ← 0
		IR	8	8	8				
		DA	11	12	14				
		X	12	12	15				
EX **EXB**	R. src	R	6	6	6				**Exchange** R ↔ src
		IR	12	12	12				
		DA	15	16	18				
		X	16	16	19				
LD **LDB** **LDL**	R. src	R	3	3	3	5	5	5	**Load into Register** R ← src
		IM	7	7	7	11	11	11	
		IM	5 (byte only)						
		IR	7	7	7	11	11	11	
		DA	9	10	12	12	13	15	
		X	10	10	13	13	13	16	
		BA	14	14	14	17	17	17	
		BX	14	14	14	17	17	17	
LD **LDB** **LDL**	dst. R	IR	8	8	8	11	11	11	**Load into Memory** (Store) dst ← R
		DA	11	12	14	14	15	17	
		X	12	12	15	15	15	18	
		BA	14	14	14	17	17	17	
		BX	14	14	14	17	17	17	
LD **LDB**	dst. IM	IR	11	11	11				**Load Immediate into Memory** dst ← IM
		DA	14	15	17				
		X	15	15	18				
LDA	R.src	DA	12	13	15				**Load Address** R ← source address
		X	13	13	16				
		BA	15	15	15				
		BX	15	15	15				
LDAR	R. src	RA	15	15	15				**Load Address Relative** R → source address
LDK	R. src	IM	5	5	5				**Load Constant** R ← n (n = 0 ... 15)
LDM	R.src.n	IR	11	11	11	+3 n			**Load Multiple** R ← src (n consecutive words) (n = 1 ... 16)
		DA	14	15	17				
		X	15	15	18				
LDM	dst.R.n	IR	11	11	11	+3 n			**Load Multiple** (Store Multiple) dst ← R (n consecutive words) (n = 1 ... 16)
		DA	14	15	17				
		X	15	15	18				
LDR **LDRB** **LDRL**	R.src	RA	14	14	14	17	17	17	**Load Relative** R ← src (range - 32768... +32767)
LDR **LDRB** **LDRL**	dst.R	RA	14	14	14	17	17	17	**Load Relative** (Store Relative) dst ← R (range −32768... +32767)

*** NS = Non-Segmented SS = Segmented Short Offset SL = Segmented Long Offset**

Load and Exchange (Continued)

Mnemonics	Operands	Addr. Mode	Clock Cycles: Word, Byte NS	Word, Byte SS	Word, Byte SL	Long Word NS	Long Word SS	Long Word SL	Operation
POP **POPL**	dst,IR	R	8	8	8	12	12	12	**Pop** dst ← IR Autoincrement contents of R
		IR	12	12	12	19	19	19	
		DA	16	16	18	23	23	25	
		X	16	16	19	23	23	26	
PUSH **PUSHL**	IR,src	R	9	9	9	12	12	12	**Push** Autodecrement contents of R IR ← src
		IM	12	12	12				
		IR	13	13	13	20	20	20	
		DA	13	14	16	21	21	23	
		X	14	14	17	21	21	24	

Arithmetic

ADC **ADCB**	R, src	R	5	5	5				**Add with Carry** R ← R + carry + src
ADD **ADDB** **ADDL**	R, src	R	4	4	4	8	8	8	**Add** R ← R + src
		IM	7	7	7	14	14	14	
		IR	7	7	7	14	14	14	
		DA	9	10	12	15	16	18	
		X	10	10	13	16	16	19	
CP **CPB** **CPL**	R, src	R	4	4	4	8	8	8	**Compare with Register** R ← src
		IM	7	7	7	14	14	14	
		IR	7	7	7	14	14	14	
		DA	9	10	12	15	16	18	
		X	10	10	13	16	16	19	
CP **CPB**	dst IM	IR	11	11	11				**Compare with Immediate** dst ← IM
		DA	14	15	17				
		X	15	15	18				
DAB	dst	R	5	5	5				**Decimal Adjust**
DEC **DECB**	dst,n	R	4	4	4				**Decrement by n** dst ← dst - n (n = 1 ... 16)
		IR	11	11	11				
		DA	13	14	16				
		X	14	14	17				
DIV **DIVL**	R, src	R	107	107	107	744	744	744	**Divide** (signed) Word: $R_{n+1} \leftarrow R_{n,\,n+1} \div$ src $R_n \leftarrow$ remainder Long Word: $R_{n+2,n+3} \leftarrow R_{n\ldots n+3} \div$ src $R_{n,n+1} \leftarrow$ remainder
		IM	107	107	107	744	744	744	
		IR	107	107	107	744	744	744	
		DA	108	109	111	745	746	748	
		X	109	109	112	746	746	749	
EXTS **EXTSB** **EXTSL**	dst	R	11	11	11	11	11	11	**Extend Sign** Extend sign of low order half of dst through high order half of dst
INC **INCB**	dst,n	R	4	4	4				**Increment by n** dst ← dst + n (n = 1 ... 16)
		IR	11	11	11				
		DA	13	14	16				
		X	14	14	17				

Arithmetic (Continued)

Mnemonics	Operands	Addr. Mode	Clock Cycles: Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
MULT **MULTL**	R, src	R IM IR DA X	70 70 70 71 72	70 70 70 72 72	70 70 70 74 75	282* 282* 282* 283* 284*	282* 282* 282* 284* 284*	282* 282* 282* 286* 287*	**Multiply** (signed) Word: $R_{n,\,n+1} \leftarrow R_{n+1} \cdot$ src Long Word: $R_{n\ldots+3} \leftarrow R_{n+2,\,n+3} \cdot$ src * Plus seven cycles for each 1 in the absolute value of the low order word of the multiplicand
NEG **NEGB**	dst	R IR DA X	7 12 15 16	7 12 16 16	7 12 18 19				**Negate** dst ← -dst
SBC **SBCB**	R,src	R	5	5	5				**Subtract with Carry** R ← R - src - carry
SUB **SUBB** **SUBL**	R,src	R IM IR DA X	4 7 7 9 10	4 7 7 10 10	4 7 7 12 13	8 14 14 15 16	8 14 14 16 16	8 14 14 18 19	**Subtract** R ← R - src

Logical

AND **ANDB**	R, src	R IM IR DA X	4 7 7 9 10	4 7 7 10 10	4 7 7 12 13				**And** R ← R AND src
COM **COMB**	dst	R IR DA X	7 12 15 16	7 12 16 16	7 12 18 19				**Complement** dst ← NOT dst
OR **ORB**	R, src	R IM IR DA X	4 7 7 9 10	4 7 7 10 10	4 7 7 12 13				**OR** R ← R OR src
TEST **TESTB** **TESTL**	dst	R IR DA X	7 8 11 12	7 8 12 12	7 8 14 15	13 13 16 17	13 13 17 17	13 13 19 20	**Test** dst OR 0
TCC **TCCB**	cc, dst	R	5	5	5				**Test Condition Code** Set LSB if cc is true
XOR **XORB**	R, src	R IM IR DA X	4 7 7 9 10	4 7 7 10 10	4 7 7 12 13				**Exclusive OR** R ← R XOR src

Program Control

Mnemonics	Operands	Addr. Mode	Clock Cycles: Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
CALL	dst	IR	10	15	15				**Call Subroutine** Autodecrement SP @ SP ← PC PC ← dst
		DA	12	18	20				
		X	13	18	21				
CALR	dst	RA	10	15	15				**Call Relative** Autodecrement SP @ SP ← PC PC ← PC + dst (range − 4094 to + 4096)
DJNZ **DBJNZ**	R, dst	RA	11	11	11				**Decrement and Jump if Non-Zero** R ← R − 1 If R ≠ 0: PC ← PC + dst (range − 254 to 0)
IRET*	–		13	16	16				**Interrupt Return** PS ← @ SP Autoincrement SP
JP	cc, dst	IR	10	15	15	(taken)			**Jump Conditional** If cc is true: PC ← dst
		IR	7	7	7	(not taken)			
		DA	7	8	10				
		X	8	8	11				
RET	cc		10	13	13	(taken)			**Return Conditional** If cc is true: PC ← @ SP Autoincrement SP
			7	7	7	(not taken)			
SC	src	IM	33	39	39				**System Call** Autoincrement SP @ SP ← Old PS Push instruction PS ← System Call PS
BIT **BITB**	dst. b	R	4	4	4				**Test Bit Static** Z flag ← NOT dst bit specified by b
		IR	8	8	8				
		DA	10	11	13				
		X	11	11	14				
BIT **BITB**	dst, R	R	10	10	10				**Test Bit Dynamic** Z flag ← NOT dst bit specified by contents of R

* Privileged instructions. Executed in system mode only.

Bit Manipulation

RES **RESB**	dst, b	R	4	4	4				**Reset Bit Static** Reset dst bit specified by b
		IR	11	11	11				
		DA	13	14	16				
		X	14	14	17				

Bit Manipulation (Continued)

Mnemonics	Operands	Addr. Mode	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
RES **RESB**	dst, R	R	10	10	10				**Reset Bit Dynamic** Reset dst bit specified by contents R
SET **SETB**	dst, b	R IR	4 11	4 11	4 11				**Set Bit Static** Set dst bit specified by
SET **SETB**	dst, R	R	10	10	10				**Set Bit Dynamic** Set dst bit specified by contents of R
TSET **TSETB**	dst	R IR DA X	7 11 14 15	7 11 15 15	7 11 17 18				**Test and Set** S flag ← MSB of dst dst ← all 1s

Rotate and Shift

RLDB	R, src	R	9	9	9				**Rotate Left Digit**
RRDB	R, src	R	9	9	9				**Rotate Right Digit**
RL **RLB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Left** Rotate dst by n bits (n = 1,2)
RLC **RLCB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Left through Carry** Rotate dst by n bits (n = 1,2)
RR **RRB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Right** Rotate dst by n bits (n = 1,2)
RRC **RRCB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Right through Carry** Rotate dst by bits (n = 1,2)
SDA **SDAB** **SDAL**	dst, R	R	(15 + 3n)			(15 + 3n)			**Shift Dynamic Arithmetic** Shift dst left or right by contents of R
SDL **SDLB** **SDLL**	dst, R	R	(15 + 3n)			(15 + 3n)			**Shift Dynamic Logical** Shift dst left or right by contents of R
SLA **SLAB** **SLAL**	dst, n	R	(13 + 3n)			(13 + 3n)			**Shift Left Arithmetic** Shift dst left by n bits
SLL **SLLB** **SLLL**	dst, n	R	(13 + 3n)			(13 + 3n)			**Shift Left Logical** Shift dst left by n bits
SRA **SRAB** **SRAL**	dst, n	R	(13 + 3n)			(13 + 3 n)			**Shift Right Arithmetic** Shift dst right by n bits
SRL **SRLB** **SRLL**	dst, n	R	(13 + 3n)			(13 + 3n)			**Shift Right Logical** Shift dst right by n bits

Block Transfer and String Manipulation

Mnemonics	Operands	Addr. Mode	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
CPD **CPDB**	R_X, src, R_Y, cc	IR	20	20	20				**Compare and Decrement** R_x - src Autodecrement src address $R_Y \leftarrow R_Y \leftarrow 1$
CPDR **CPDRB**	R_X, src, R_Y, cc	IR	(11 + 9n)						**Compare, Decrement and Repeat** R_X - src Autodecrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_y = 0$
CPI **CPDRB**	R_X, src, R_Y, cc	IR	20	20	20				**Compare, Decrement and Repeat** R_X – src Autodecrement src address $R_Y \leftarrow R_Y - 1$
CPIR **CPIRB**	R_x, src, R_Y, cc	IR	(11 + 9n)						**Compare, Increment and Repeat** R_x, src Autoincrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_Y = 0$
CPSD **CPSDB**	dst, src, R, cc	IR	25	25	25				**Compare String and Decrement** dst ← src Autodecrement dst and src addresses R ← R – 1
CPSDR **CPSDRB**	dst, src, R, cc	IR	(11 + 14n)						**Compare String, Decrement and Repeat** dst ← src Autodecrement dst and src addresses R ← R – 1 Repeat until cc is true or R = 0
CPSI **CPSIB**	dst,src, R,cc	IR	25	25	25				**Compare String and Increment** dst - src Autoincrement dst and src addresses R ← R – 1
CPSIR **CPSIRB**	dst,src, R,cc	IR	11 + 14n)						**Compare String, Increment and Repeat** dst – src Autoincrement dst and src addresses R ← R – 1 Repeat until cc is true or R = 0
LDD **LDDB**	dst,src, R	IR	20	20	20				**Load and Decrement** dst ← src Autodecrement dst and src addresses R ← R – 1
LDDR **LDDRB**	dst,src, R	IR	(11 + 9n)						**Load, Decrement and Repeat** dst ← src Autodecrement dst and src addresses R ← R – 1 Repeat until R = 0

Block Transfer and String Manipulation (Continued)

Mnemonics	Operands	Addr. Mode	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
LDI **LDIB**	dst,src, R	IR	20	20	20				**Load and Increment** dst ← src Autoincrement dst and src addresses R ← R − 1
LDIR **LDIRB**	dst,src, R	IR	(11 + 9n)						**Load, Increment and Repeat** dst ← src Autoincrement dst and src addresses R ← R − 1 Repeat until R = 0
TRDB	dst,src, R	IR	25	25	25				**Translate and Decrement** dst ← src (dst) Autodecrement dst address R ← R − 1
TRDRB	dst,src, R	IR	(11 + 14n)						**Translate, Decrement and Repeat** dst − src (dst) Autodecrement dst address R ← R − 1 Repeat until R = 0
TRIB	dst,src, R	IR	25	25	25				**Translate and Increment** dst ← src (dst) Autoincrement dst address R ← R − 1
TRIRB	dst,src, R	IR	(11 + 14n)						**Translate, Increment and Repeat** dst ← src (dst) Autoincrement dst address R ← R − 1 Repeat until R = 0
TRTDB	src1,src2,R	IR	25	25	25				**Translate and Test, Decrement** RH1 ← src 2 (src1) Autodecrement src 1 address R ← R − 1
TRTDRB	src1,src2,R	IR	(11 + 14n)						**Translate and Test, Decrement and Repeat** RH1 ← src2 (src1) Autodecrement src1 address R ← R − 1 Repeat until R = 0 or RH1 = 0
TRTIB	src1,src2,R	IR	25	25	25				**Translate and Test, Increment** RH1 ← src2 (src1) Autoincrement src 1 address R ← R − 1

Block Transfer and String Manipulation (Continued)

Mnemonics	Operands	Addr. Mode	Clock Cycles: Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
TRTIRB	src1,src2,R	IR	(11 + 14n)						**Translate and Test, Increment and Repeat** RH1 ← src 2 (src1) Autoincrement src1 address R ← R − 1 Repeat until R = 0 or RH1 = 0

Input/Output

Mnemonics	Operands	Addr. Mode	Clock Cycles: Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
IN*	R,src	IR	10	10	10				**Input**
INB*		DA	12	12	12				R ← src
IND* **INDB***	dst,src,R	IR	21	21	21				**Input and Decrement** dst ← src Autodecrement dst address R ← R − 1
INDR* **INDRB***	dst,src,R	IR	(11 + 10n)						**Input, Decrement and Repeat** dst ← src Autodecrement dst address R ← R − 1 Repeat until R = 0
INI* **INIB***	dst,src,R	IR	21	21	21				**Input and Increment** dst ← src Autoincrement dst address R ← R − 1
INIR* **INIRB***	dst,src,R	IR	(11 + 10n)						**Input, Increment and Repeat** dst ← src Autoincrement dst address R ← R − 1 Repeat until R = 0
OUT*	dst,R	IR	10	10	10				**Output**
OUTB*		DA	12	12	12				dst ← R
OUTD* **OUTDB***	dst,src,R	IR	21	21	21				**Output and Decrement** dst ← src Autodecrement src address R ← R − 1
OTDR* **OTDRB***	dst,src,R	IR	(11 + 10n)						**Output, Decrement and Repeat** dst ← src Autodecrement src address R ← R − 1 Repeat until R = 0

* Privileged instructions. Executed in system mode only.

Input/Output (Continued)

Mnemonics	Operands	Addr. Mode	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
OUTI* **OUTIB***	dst,src,R	IR	21	21	21				**Output and Increment** dst ← src Autoincrement src address R ← R − 1
OTIR* **OTIRB***	dst,src,R	IR	(11 + 10n)						**Output, Increment and Repeat** dst ← src Autoincrement src address R ← R − 1 Repeat until R = 0
SIN* **SINB***	R,src	DA	12	12	12				**Special Input** R ← src
SIND* **SINB***	dst,src,R	IR	21	21	21				**Special Input and Decrement** dst ← src Autodecrement dst address R ← R − 1
SINDR* **SINDRB***	dst,src,R	IR	(11 + 10n)						**Special Input, Decrement and Repeat** dst ← src Autodecrement dst address R ← R − 1 Repeat until R = 0
SINI* **SINIB***	dst,src,R	IR	21	21	21				**Special Input and Increment** dst ← src Autoincrement dst address R ← R − 1
SINIR* **SINIRB***	dst,src,R	IR	(11 + 10n)						**Special Input, Increment and Repeat** dst ← src Autoincrement dst address R ← R − 1 Repeat until R = 0
SOUT* **SOUTB***	dst,src	DA	12	12	12				**Special Output** dst ← src
SOUTD* **SOUTDB***	dst,src,R	IR	21	21	21				**Special Output and Decrement** dst ← src Autodecrement src address R ← R − 1
SOTDR* **SOTDRB***	dst,src,R	IR	(11 + 10n)						**Special Output, Decrement and Repeat** dst ← src Autodecrement src address R ← R − 1 Repeat until R = 0
SOUTI* **SOUTIB***	dst,src,R	IR	21	21	21				**Special Output and Increment** dst ← src Autoincrement src address R ← R − 1

Input/Output (Continued)

Mnemonics	Operands	Addr. Mode	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
SOTIR* **SOTIRB***	dst,src,R	R	(11 + 10n)						**Special Output, Increment and Repeat** dst ← src Autoincrement src address R ← R − 1 Repeat until R = 0

CPU Control

COMFLG	flags		7	7	7				**Complement Flag** (Any combination of C,Z,S,P/V)
DI*	int		7	7	7				**Disable Interrupt** (Any combination of NVI, VI)
EI*	int		7	7	7				**Enable Interrupt** (Any combination of NVI, VI)
HALT*			(8 + 3n)						**HALT**
LDCTL*	CTLR,src	R	7	7	7				**Load into Control Register** CTLR ← src
LDCTL*	dst,CTLR	R	7	7	7				**Load from ControL Register** dst ← CTLR
LDCTLB	FLGR,src	R	7	7	7				**Load into Flag Byte Register** FLGR ← src
LDCTLB	dst,FLGR	R	7	7	7				**Load from Flag Byte Register** dst ← FLGR
LDPS*	src	IR DA X	12 16 17	16 20 20	16 22 23				**Load Program Status** PS ← src
MBIT*			7	7	7				**Test Multi-Micro Bit** Set S if MI is Low; clear S if MI is High
MREQ*	dst	R	(12 + 7n)						**Multi-Micro Request**
MRES*			5	5	5				**Multi-Micro Reset**
MSET*			5	5	5				**Multi-Micro Set**
NOP			7	7	7				**No Operation**
RESFLG	flag		7	7	7				**Reset Flag** (Any combination of C.Z.S.P/V)
SETFLG	flag		7	7	7				**Set Flag** (Any combination of C.Z.S.P/V)

* Privileged instructions. Executed in system mode only.

Extended Instructions

Function	Addr. Mode	Clock Cycles NS	SS	SL	Operation
Memory ← EPU	IR	(11 + 3n)	(11 + 3n)	(11 + 3n)	**Load Memory from EPU**
	X	(15 + 3n)	(15 + 3n)	(18 + 3n)	Write n words from EPU into memory
	DA	(14 + 3n)	(15 + 3n)	(17 + 3n)	
EPU ← Memory	IR	(11 + 3n)	(11 + 3n)	(11 + 3n)	**Load EPU from Memory**
	X	(15 + 3n)	(15 + 3n)	(18 + 3n)	Read n words from memory into EPU
	DA	(14 + 3n)	(15 + 3n)	(17 + 3n)	
CPU ← EPU Registers		(11 + 4n)	(11 + 4n)	(11 + 4n)	**Load VMPU from EPU** Transfer n words from EPU to VMPU registers
EPU ← CPU Registers		(11 + 4n)	(11 + 4n)	(11 + 4n)	**Load EPU from VMPU** Transfer n words from VMPU register to EPU
Flags ← EPU		15	15	15	**Load FCW from EPU** Load information from EPU into flags of the VMPU's Flag and Control Word
EPU ← Flags		15	15	15	**Load EPU from FCW** Transfer information from VMPU's Flag and Control Word to EPU
EPU Internal Operations		(11 + 4n)	(11 + 4n)	(11 + 4n)	**Internal EPU Operations** VMPU treats this template as a "no-operations"; it is typically used to initiate an internal EPU operation. The character is a field in the instruction.

Absolute Maximum Ratings

Voltages on all inputs and outputs with respect to GND . . −0.3 V to + 7.0 V
Operating Ambient Temperature . . . . . 0°C to + 70°C
Storage Temperature . . −65°C to + 150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition beyond those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Standard Test Conditions

Standard test temperature/operating voltage ranges are presented below. All voltages are referenced to GND. Positive current flows into the referenced pin.

- 0°C to +70°C, +4.75 V ≤ V_{CC} ≤ +5.25 V
- −40°C to +85°C, +4.75V ≤ V_{CC} ≤ +5.25V
- ⇐55°C to +125°C, +4.5V ≤ V_{CC} ≤ +5.5V

All ac parameters assume a load capacitance of 100 pF max, except for parameter 6, which has a load capacitance of 50 pF max. Timing reference between two output signals assume a load difference of 50 pF max.

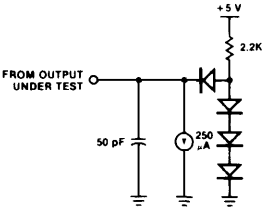


Figure 18. Standard Test Load

DC Characteristics

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	V_{CC} − 0.4	V_{CC} + 0.3	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3	0.45	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	I_{IO} = − 250 A
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = + 2.0 mA
I_{IL}	Input Leakage		±10	A	0.4 V_{IN} + 2.4 V
I_{OL}	Output Leakage		±10	A	0.4 V_{OUT} + 2.4 V
I_{CC}	V_{CC} Supply Current		300	mA	

AC Characteristics †

No.	Symbol	Parameter	Z8003/Z8004 (4 MHz) Min	Z8003/Z8004 (4 MHz) Max	Z8003A/Z8004A (6 MHz) Min	Z8003A/Z8004A (6 MHz) Max	Z8003B/Z8004B (10 MHz) Min	Z8003B/Z8004B (10 MHz) Max
1	TcC	Clock Cycle Time	250	2000	165	2000	100	2000
2	TwCh	Clock Width (High)	105	2000	70	2000	40	
3	TwCl	Clock Width (Low)	105	2000	70	2000	40	
4	TfC	Clock Fall Time		20		10		10
5	TrC	Clock Rise Time		20		15		10
6	TdC(SNv)	Clock ↑ to Segment Number Valid (50 pF laod)		130		110		70
7	TdC(SNn)	Clock ↑ to Segment Number Not Valid	20		10		5	
8	TdC(Bz)	Clock ↑ to Bus Float		65		55		40
9	TdC(A)	Clock ↑ to Address Valid		100		75		50
10	TdC(Az)	Clock ↑ to Address Float		65		55		40
11	TdA(DR)	Address Valid to Read Data Required Valid		475*		305*		180*
12	TsDR(C)	Read Data to Clock ↓ Setup Time	30		20		10	
13	TdDS(A)	$\overline{DS}$ ↑ to Address Active	80*		45*		20*	
14	TdC(DW)	Clock ↑ to Write Data Valid		100		75		50
15	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0		0		50	
16	TdDW(DS)	Write Data Valid to $\overline{DS}$ ↑ Delay	295*		195*		110*	
17	TdA(MR)	Address Valid to $\overline{MREQ}$ ↓ Delay	55		35*		20*	
18	TdC(MR)	Clock ↓ to $\overline{MREQ}$ ↓ Delay		80		70		40
19	TwMRh	$\overline{MREQ}$ Width (High)	210*		135*		80*	
20	TdMR(A)	$\overline{MREQ}$ ↓ to Address Not Active	70*		35*		20*	
21	TdDW(DSW)	Write Data Valid to $\overline{DS}$ ↓ (Write) Delay	55*		35*		15*	
22	TdMR(DR)	$\overline{MREQ}$ ↓ to Read Data Required Valid	375*		230*		140*	
23	TdC(MR)	Clock ↓ $\overline{MREQ}$ ↑ Delay		80		60		45
24	TdC(ASf)	Clock ↑ to $\overline{AS}$ ↓ Delay		80		60		40
25	TdA(AS)	Address Valid to $\overline{AS}$ ↑ Delay	55*		35*		20*	
26	TdC(ASr)	Clock ↓ to $\overline{AS}$ ↑ Delay		90		80		40
27	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid	360*		220*		140*	
28	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	70*		35*		15*	
29	TwAS	$\overline{AS}$ Width (Low)	85*		55*		30*	
30	TdAS(A)	$\overline{AS}$ ↑ to Address Not Active Delay	70*		45*		20*	
31	TdAz(DSR)	Address Float to $\overline{DS}$ (Read) ↓ Delay	0		0		0	
32	TdAS(DSR)	$\overline{AS}$ ↑ to $\overline{DS}$ (Read) ↓ Delay	80*		55*		30*	
33	TdDSR(DR)	$\overline{DS}$ (Read) ↓ to Read Data Required Valid	205*		130*		70*	
34	TdC(DSr)	Clock ↓ to $\overline{DS}$ ↑ Delay		70		65		45
35	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid	75*		45*		25*	
36	TdA(DSR)	Address Valid to $\overline{DS}$ (Read) ↓ Delay	180*		110*		65*	
37	TdC(DSR)	Clock ↑ to $\overline{DS}$ (Read) ↓ Delay		120		85		60
38	TwDSR	$\overline{DS}$ (Read) Width (Low)	275*		185*		110*	
39	TdC(DSW)	Clock ↓ to $\overline{DS}$ (Write) ↓ Delay		95		80		60
40	TwDSW	$\overline{DS}$ (Write) Width (Low)	185*		110*		75*	
41	TdDSI(DR)	$\overline{DS}$ (I/O) ↓ to Read Data Required Valid	330*		210*		120*	

* Clock-cycle-time-dependent characteristics. See table on following page.

† Timings are preliminary and subject to change. Units in nanoseconds (ns).

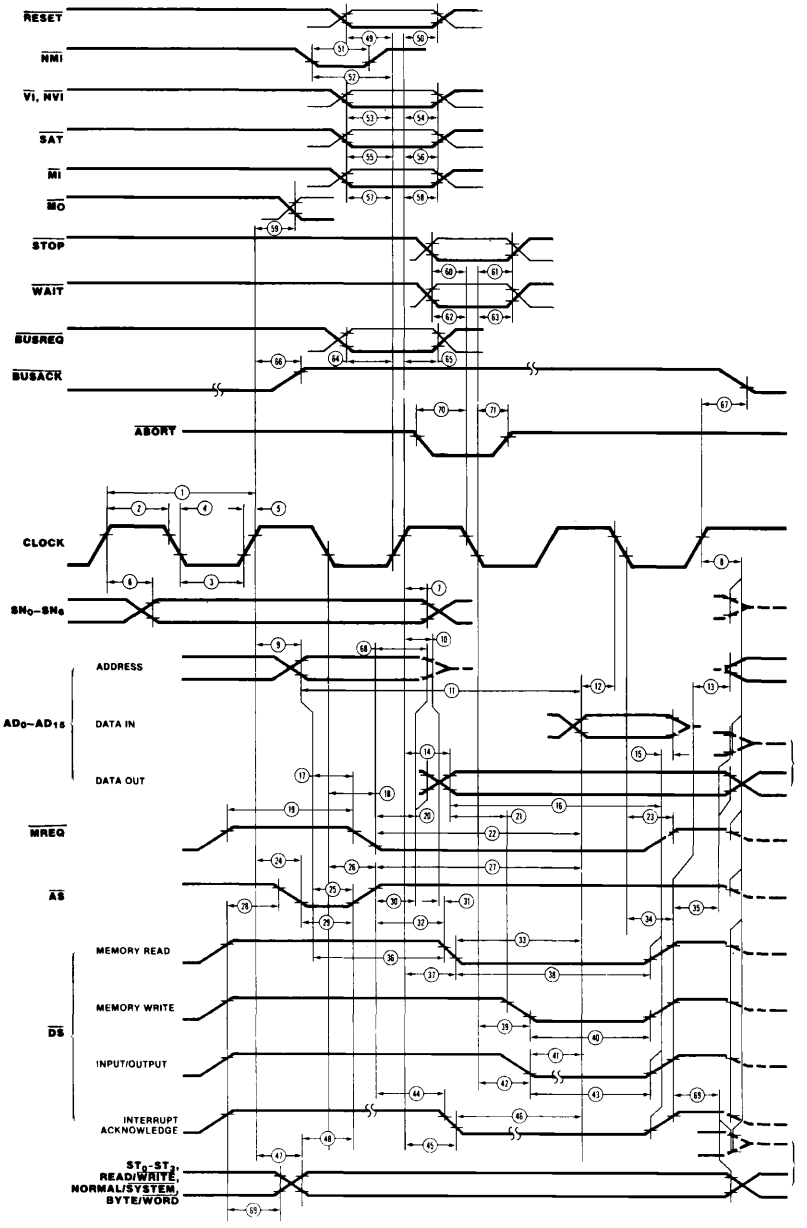
AC Characteristics † (Continued)

Number	Symbol	Parameter	Z8003/Z8004 (4 MHz) Min	Z8003/Z8004 (4 MHz) Max	Z8003A/Z8004A (6 MHz) Min	Z8003A/Z8004A (6 MHz) Max	Z8003B/Z8004B (10 MHz) Min	Z8003B/Z8004B (10 MHz) Max
42	TdC(DSf)	Clock ↓ to $\overline{DS}$ (I/O) ↓ Delay		120		90		60
43	TwDS	$\overline{DS}$ (I/O) Width (Low)	410*		255*		160*	
44	TdAS(DSA)	$\overline{AS}$ ↑ to $\overline{DS}$ (Acknowledge) ↓ Delay	1065*		690*		410*	
45	TdC(DSA)	Clock ↑ to $\overline{DS}$ (Acknowledge) ↓ Delay		120		85		65
46	TdDSA(DR)	$\overline{DS}$ (Acknowledge) ↓ to Read Data Required Delay	455*		295*		165*	
47	TdC(S)	Clock ↑ to Status Valid Delay		110		85		60
48	TdS(AS)	Status Valid to $\overline{AS}$ ↑ Delay	50		30*		10*	
49	TsR(C)	$\overline{RESET}$ to Clock ↑ Setup Time	180		70		50	
50	ThR(C)	$\overline{RESET}$ to Clock ↑ Hold Time	0		0		0	
51	TwNMI	$\overline{NMI}$ Width (Low)	100		70		50	
52	TsNMI(C)	$\overline{NMI}$ to Clock ↑ Setup Time	140		70		50	
53	TsVI(C)	$\overline{VI}$, $\overline{NVI}$ to Clock ↑ Setup Time	110		50		40	
54	ThVI(C)	$\overline{VI}$, $\overline{NVI}$ to Clock ↑ Hold Time	20		20		10	
55	TsSGT(C)	$\overline{SAT}$ to Clock ↑ Setup Time	70		55		40	
56	ThSGT(C)	$\overline{SAT}$ to Clock ↑ Hold Time	0		0		0	
57	TsMI(C)	$\overline{MI}$ to Clock ↑ Setup Time	180		110		80	
58	ThMI(C)	$\overline{MI}$ to Clock ↑ Hold Time	0		0		0	
59	TdC(MO)	Clock ↑ to $\overline{MO}$ Delay		120		85		70
60	TsSTP(C)	$\overline{STOP}$ to Clock ↓ Setup Time	140		80		50	
61	ThSTP(C)	$\overline{STOP}$ to Clock ↓ Hold Time	0		0		0	
62	TsW(C)	$\overline{WAIT}$ to Clock ↓ Setup Time	50		30		20	
63	ThW(C)	$\overline{WAIT}$ to Clock ↓ Hold Time	10		10		5	
64	TsBRQ(C)	$\overline{BUSREQ}$ to Clock ↑ Setup Time	90		80		60	
65	ThBRQ(C)	$\overline{BUSREQ}$ to Clock ↑ Hold Time	10		10		5	
66	TdC(BAKr)	Clock ↑ to $\overline{BUSACK}$ ↑ Delay		100		75		60
67	TdC(BAKf)	Clock ↑ to $\overline{BUSACK}$ ↓ Delay		100		75		60
68	TwA	Address Valid Width	150*		95*		50*	
69	TdDS(S)	$\overline{DS}$ ↑ to STATUS Not Valid	80*		55*		30*	
70	TsABT(C)	$\overline{ABORT}$ ↓ to Clock ↑ Setup Time	50		30		25	
71	ThABT(C)	$\overline{ABORT}$ ↓ to Clock ↓ Hold Time	0		0		0	

* Clock-cycle-time-dependent characteristics. See table on following page.

† Timings are preliminary and subject to change. Units in nanoseconds (ns).

Composite AC Timing Diagram



Clock-Cycle-Time-Dependent Characteristics

Number	Symbol	Z8003 Equation	Z8003A Equation	Z8003B Equation
11	TdA(DR)	2TcC + TwCh – 130 ns	2TcC + TwCh – 95 ns	2TcC + TwCh – 60 ns
13	TdDS(A)	TwCl – 25 ns	TwCl – 25 ns	TwCl – 20 ns
16	TdDW(DS)	TcC + TwCh – 60 ns	TcC + TwCh – 40 ns	TcC + TwCh – 30 ns
17	TdA(MR)	TwCh – 50 ns	TwCh – 35 ns	TwCh – 20 ns
19	TwMRh	TcC – 40 ns	TcC – 30 ns	TcC – 20 ns
20	TdMR(A)	TwCl – 35 ns	TwCl – 35 ns	TwCl – 20 ns
21	TdDW(DSW)	TwCh – 50 ns	TwCh – 35 ns	TwCh – 25 ns
22	TdMR(DR)	2TcC – 130 ns	2TcC – 100 ns	2TcC – 60 ns
25	TdA(AS)	TwCh – 50 ns	TwCh – 35 ns	TwCh – 20 ns
27	TdAS(DR)	2TcC – 140 ns	2TcC – 110 ns	2TcC – 60 ns
28	TdDS(AS)	TwCl – 35 ns	TwCl – 35 ns	TwCl – 25 ns
29	TwAS	TwCh – 20ns	TwCh – 15 ns	TwCh – 10 ns
30	TdAS(A)	TwCl – 35 ns	TwCl – 25 ns	TwCl – 20 ns
32	TdAS(DSR)	TwCl – 25 ns	TwCl – 15 ns	TwCl – 10 ns
33	TdDSR(DR)	TcC + TwCh – 150 ns	TcC + TwCh – 105 ns	TcC + TwCh – 70 ns
35	TdDS(DW)	TwCl – 30 ns	TwCl – 25 ns	TwCl – 15 ns
36	TdA(DSR)	TcC – 70 ns	TcC – 55 ns	TcC – 35 ns
38	TwDSR	TcC + TwCh – 80 ns	TcC + TwCh – 50 ns	TcC + TwCh – 30 ns
40	TwDSW	TcC – 65 ns	TcC – 55 ns	TcC – 25 ns
41	TdDSI(DR)	2TcC – 170 ns	2TcC – 120 ns	2TcC – 80 ns
43	TwDS	2TcC – 90 ns	2TcC – 75 ns	2TcC – 40 ns
44	TdAS(DSA)	4TcC + TwCl – 40 ns	4TcC + TwCl – 40 ns	4TcC + TwCl – 30 ns
46	TdDSA(DR)	2TcC + TwCh – 150 ns	2TcC + TwCh – 105 ns	2TcC + TwCh – 75 ns
48	TdS(AS)	TwCh – 55 ns	TwCh – 40 ns	TwCh – 30 ns
68	TwA	TcC – 90 ns	TcC – 70 ns	TcC – 50 ns
69	TdDS(S)	TwCl – 25 ns	TwCl – 15 ns	TwCl – 10 ns

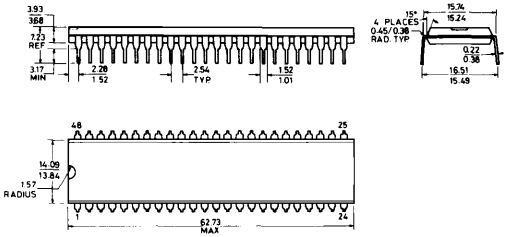
Ordering Information

Type	Package	Temp.	Clock	Description
Z8003 B1	Plastic 48 pin	0/+70°C	4 MHz	Z8003 Virtual Memory Processing Unit
B6	Plastic 48 pin	−40/+85°C		
D1	Ceramic 48 pin	0/+70°C		
D2	Ceramic 48 pin	−55/+125°C		
D6	Ceramic 48 pin	−40/+ 85°C		
Z8003A B1	Plastic 48 pin	0/+70°C	6 MHz	
B6	Plastic 48 pin	−40/+85°C		
D1	Ceramic 48 pin	0/+70°C		
D6	Ceramic 48 pin	−40/+85°C		
Z8003B B1	Plastic 48 pin	0/+70°C	10 MHz	
B6	Plastic 48 pin	−40/+85°C		
D1	Ceramic 48 pin	0/+70°C		
D6	Ceramic 48 pin	−40/+85°C		
Z8004 B1	Plastic 40 pin	0/+70°C	4 MHz	Z8004 Virtual Memory Processing Unit
B6	Plastic 40 pin	−40/+85°C		
D1	Ceramic 40 pin	0/+70°C		
D2	Ceramic 40 pin	−55/+125°C		
D6	Ceramic 40 pin	−40/+85°C		
Z8004A B1	Plastic 40 pin	0/+70°C	6 MHz	
B6	Plastic 40 pin	−40/+85°C		
D1	Ceramic 40 pin	0/+70°C		
D6	Ceramic 40 pin	−40/+85°C		
Z8004B B1	Plastic 40 pin	0/+70°C	10 MHz	
B6	Plastic 40 pin	−40/+85°C		
D1	Ceramic 40 pin	0/+70°C		
D6	Ceramic 40 pin	−40/+85°C		

Packages (dimensions in mm)

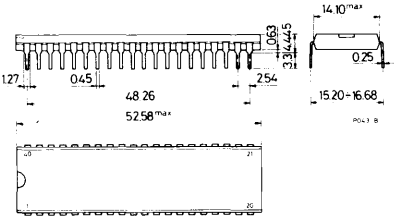
Plastic

Z8003



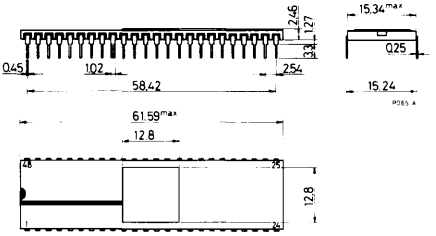
Plastic

Z8004



Ceramic

Z8003



Ceramic

Z8004