ZiLOG

Z380™
Microprocessor Unit

Microprocessor Solutions for Datacommunications and Computer Peripheral Applications

User's Manual

Z380™ Microprocessor Unit

User's Manual

Q3/94 DC 8297-03

PREFACE

Thank you for your interest in the Z380™ CPU (Central Processing Unit) and its associated family of products. This Technical Manual describes programming and operation of the Z380™ Superintegration™ Core CPU, which is found in the Z380 MPU (Microprocessor Processing Unit), and future products built around Z380™ CPU core. For the external interface and detailed descriptions of the on-chip peripherals for each Superintegration device, please refer to individual product specifications.

This Technical manual consists of the following Sections:

1. **Z380™ Architectural Overview**
   Chapter 1 is an introductory section covering the key features and giving an overview of the architecture of the device.

2. **Address Spaces**
   Chapter 2 explains the address spaces the Z380 CPU can handle. Also, this chapter includes a brief description of the on-chip registers.

3. **Native/Extended Mode, Word/Long Word Mode of Operation, and Decoder Directives**
   This chapter provides a detailed explanation on the Z380's unique features, operation modes, and the Decoder Directives.

4. **Addressing Modes and Data Types**
   Chapter 4 describes the Addressing mode and data types which the Z380 can handle.

5. **Instruction Set**
   Chapter 5 contains an overview of the instruction set; as well as a detailed instruction-by-instruction description in alphabetical order.

6. **Interrupts and Traps**
   Chapter 6 explains the interrupts and traps features of the Z380.

7. **Reset**
   Chapter 7 describes the Reset function.

ZILOG

Table of Contents

TABLE OF CONTENTS

FIGURES

ZILOG

CHAPTER 1
Z380™ ARCHITECTURAL OVERVIEW

1.1 INTRODUCTION

The Z380 CPU incorporates advanced architectural features that allow fast and efficient throughput and increased memory addressing capabilities while maintaining Z80® CPU and Z180® MPU object-code compatibility. The Z380 CPU core provides a continuing growth path for present Z80- or Z180®-based designs and offers the following key features:

- Full Static CMOS Design with Low Power Standby Mode Support
- DC to 18 MHz Operating Frequency @ 5 Volts V_{CC}
- DC to 10 MHz Operating Frequency @ 33 Volts V_{CC}
- Enhanced Instruction Set that Maintains Object-Code Compatibility with Z80 and Z180 Microprocessors
- 16-Bit (64K) or 32-Bit (4G) Linear Address Space
- 16-Bit Internal Data Bus
- Two Clock Cycle Instruction Execution (Minimum)
- Multiple On-Chip Register Files (Z380 MPU has Four Banks)
- BC/DE/HL/IX/IY Registers are Augmented by 16-Bit Extended Registers (BCz/DEz/HLz/IXz/IYz), PC/SP/I Registers are Augmented by Extended Registers (PCz/SPz/Iz) for 32-Bit Addressing Capability.
- Newly Added IX' and IY' Registers with Extended Registers (IXz'/IYz')
- Enhanced Interrupt Capabilities, Including 16-Bit Vector
- Undefined Opcode Trap for Full Z380 CPU Instruction Set

The Z380 CPU, an enhanced version of the Z80 CPU, retains the Z80 CPU instruction set to maintain complete binary-code compatiblity with present Z80 and Z180 codes. The basic addressing modes of the Z80 microprocessor have been augmented with Stack Pointer Relative loads and stores, 16-bit and 24-bit Indexed offsets, and increased Indirect register addressing flexibility, with all of the addressing modes allowing access to the entire 32-bit address space. Significant additions have been made to the instruction set iincorporating16-bit arithmetic and logical operations, 16-bit I/O operations, multiply and divide, a complete set of register-to-register loads and exchanges, plus 32-bit load and exchange, and 32-bit arithmetic operation for address calculation.

The basic register file of the Z80 microprocessor is expanded to include alternate register versions of the IX and IY registers. There are four sets of this basic Z80 microprocessor register file present in the Z380 MPU, along with the necessary resources to manage switching between the different register sets. All of the register pairs and index registers in the basic Z80 microprocessor register file are expanded to 32 bits.

The Z380 CPU expands the basic 64 Kbyte Z80 and Z180 address space to a full 4 Gbyte (32-bit) address space. This address space is linear and completely accessible to the user program. The external I/O address space is similarly expanded to a full 4 Gbyte (32-bit) range, and 16-bit I/O, both simple and block move are included. A 256 byte-wide internal I/O space has been added. This space will be used to access on-chip I/O resources on future Superintegration implementation of this CPU core.

Figure 1-1 provides a detailed description of the basic register architecture of the Z380 CPU with the size of the register banks shown at four each, however, the Z380 CPU architecture allows future expansion of up to 128 sets of each.

1.2 CPU ARCHITECTURE

The Z380 CPU is a binary-compatible extension of the Z80 CPU and the Z180 CPU architecture. High throughput rates are achieved by a high clock rate, high bus bandwidth, and instruction fetch/execute overlap. Communicating to the external world through an 8-bit or 16-bit data bus, the Z380 CPU is a full 32-bit machine internally, with a 32-bit ALU and 32-bit registers.

1.2.1 Modes of Operation

To maintain compatibility with the Z80/Z180 CPU while having the capability to manipulate 4 Gbytes of memory address range, the Z380 CPU has two bits in the Select Register (SR) to control the modes of operation. One bit controls the address manipulation mode: Native mode or Extended mode; and the other bit controls the data manipulation mode: Word mode or Long Word mode. In result, the Z380 CPU has four modes of operation. On reset, the Z380 CPU is in Native/Word mode, which is compatible to the Z80/Z180's operation mode. For details on this subject, refer to Chapter 3, "Native/Extended Mode, Word/Long Word Mode of Operation, and Decoder Directive Instructions."

1.2.1.1 Native Mode and Extended Mode

The Z380 CPU can operate in either Native or Extended mode, as controlled by a bit in the Select Register (SR). In Native mode (the Reset configuration), all address manipulations are performed modulo 65536 (2^{16}). In this mode, the Program Counter (PC) only increments across 16 bits, all address manipulation instructions (increment, decrement, add, subtract, indexed, stack relative, and PC relative) only operate on 16 bits, and the Stack Pointer (SP) only increments and decrements across 16 bits. The PC high-order word is left at all zeros, as the high-order words of the SP and the I register. Thus, Native mode is fully compatible with the Z80 CPU's 64 Kbyte address mode. It is still possible to address memory outside of 64 Kbyte address space for data storage and retrieval in Native mode, however, since direct addresses, indirect addresses, and the high-order word of the SP, I, and the IX and IY registers may be loaded with non-zero values. Executed code and interrupt service routines must reside in the lowest 64 Kbytes of the address space.

In Extended mode, however, all address manipulation instructions operate on 32 bits, allowing access to the entire 4 Gbyte address space of the Z380 CPU. In both Native and Extended modes, the Z380 drives all 32 bits of the address onto the external address bus; only the width of the manipulated addresses distinguishes Native from Extended mode. The Z380 CPU implements one instruction to allow switching from Native to Extended mode (SETC XM); however, once in Extended mode, only Reset will return the Z380 CPU to Native mode. This restriction applies because of the possibility of "misplacing" interrupt service routines or vector tables during the transition from Extended mode back to Native mode.

1.2.1.2 Word or Long Word Mode

In addition to Native and Extended mode, which are specific to memory space addressing, the Z380 CPU can operate in either Word or Long Word mode specific to data load and exchange operations. In Word mode (the Reset configuration), all word load and exchange operations manipulate 16-bit quantities. For example, only the low-order words of the source and destination are exchanged in an exchange operation, with the high-order words unaffected.

In the Long Word mode, all 32 bits of the source and destination are exchanged. The Z380 CPU implements two instructions plus decoder directives to allow switching between Word and Long Word mode; SETC LW (Set Control Long Word) and RESC LW (Reset Control Long Word) perform a global switch, while DDIR W, DDIR LW and their variants are decoder directives that select a particular mode only for the instruction that they precede.

Note that all word data arithmetic (as opposed to address manipulation arithmetic), rotate, shift, and logical operations are always in 16-bit quantities. They are not controlled by either the Native/Extended or Word/Long Word selections. The exceptions to the 16-bit quantities are, of course, those multiply and divide operations with 32-bit products or dividends.

All word Input/Output operations are performed on 16-bit values, regardless of Word/Long Word operation.

1.2.2 Address Spaces

Addressing spaces in the Z380 CPU include the CPU register, the CPU control register, the memory address, on-chip I/O address, and the external I/O address. The CPU register space is a superset of the Z80 CPU register set, and consists of all of the registers in the CPU register file. These CPU registers are used for data and address manipulation, and are an extension of the Z80 CPU register set, with four sets of this extended Z80 CPU register set present in the Z380 CPU. Access to these registers is specified in the instruction, with the active register set selected by bits in the Select Register (SR) in the CPU control register space.

are handled by a newly added interrupt handing mode, "Assigned Vectored Mode," which is a fixed vectored interrupt mode similar in interrupt handling to the Z180's interrupts from on-chip peripherals. For handling interrupt requests on the /INT0 line, there are four modes available:

- 8080 compatible (Mode 0), in which the interrupting device provides the first instruction of the interrupt routine.
- Dedicated interrupts (Mode 1), in which the CPU jumps to a dedicated address when an interrupt occurs.
- Vectored interrupt mode (Mode 2), in which the interrupting peripheral device provides a vector into a table of jump address.
- Enhanced vectored interrupt mode (Mode 3), wherein the CPU expects 16-bit vector, instead of 8-bit interrupt vectors in Mode 2.

The first three modes are compatible with Z80 interrupt modes; the fourth mode provides more flexibility.

Traps are synchronous events that trigger a special CPU response when an undefined instruction is executed. It can be used to increase system reliability, or used as a "software trap instruction."

Hardware resets occur when the /RESET line is activated and override all other conditions. A /RESET causes certain CPU control registers to be initialized.

For details on this subject, refer to Chapter 6, "Interrupts and Traps."

1.3 BENEFITS OF THE ARCHITECTURE

The Z380 CPU architecture provides several significant benefits, including increased program throughput achieved by higher bus bandwidth (16-bit wide bus), reduction to two clocks/basic machine cycle (vs four clocks/cycle on the Z80 CPU), prefetch cue, access to the larger linear addressing space, enhanced instructions/new addressing mode, data/address manipulation in 16/32 bits, and faster context switching by utilizing multiple register banks.

1.3.1 High Throughput

Very high throughput rates can be achieved with the Z380 CPU, due to the basic machine cycle's reduction to two clocks/cycle from four clocks/cycle on the Z80 CPU, fine tuned four staged pipeline with prefetch cue. This well designed pipeline and prefetch cue are both totally transparent to the user, thus maximizing the efficiency of the pipeline all the time. The Z380 CPU implemented onto the Z380 MPU is configured with a 16-bit wide data bus, which doubles the bus bandwidth. These architectural features result in two clocks/instructions execution minimum, three clocks/instruction on average. The high clock rates (up to 40 MHz) achievable with this processor. Make the overall performance of the Z380 CPU more than ten times that of the Z80.

1.3.2 Linear Memory Address Space

Z380 CPU architecture has 4 Gbytes of linear memory address space. The Z80 CPU architecture allows 64 Kbytes of memory addressing space. This was more than sufficient when the Z80 CPU was first developed. But as the technology improved over time, applications started to demand more complicated processing, multitasking, faster processing, etc., with the high level language needed to develop software. As a result, 64 Kbytes of memory addressing space is not enough for some Z80 CPU based applications. In order to handle more than 64 Kbytes of memory, the Z80 CPU requires a Memory Banking scheme, or MMU (Memory Management Unit), like the Z180 MPU or Z280 MPU. These provide the overhead to access more than 64 Kbytes of memory.

The Z380 CPU architecture allows access to a full 4 Gbytes (2^{32}) of memory addressing space as well as 4 Gbytes of I/O addressing area, without using a Memory Banking scheme, or MMU.

1.3.3. Enhanced Instruction Set with 16-Bit and 32-Bit Manipulation Capability

The Z380 CPU instruction set is 100% upward compatible to the Z80 CPU instruction set; that is all the Z80 instructions have been preserved at the binary level. New instructions added to the Z380 CPU include:

- Less restricted operand source/destination combinations.
- More flexible register exchange instructions.
- Stack Pointer Relative addressing mode.

ZILOG

CHAPTER 2
ADDRESS SPACES

2.1 INTRODUCTION

The Z380 CPU supports five address spaces corresponding to the different types of locations that can be addressed and the method by which the logical addresses are formed. These five address spaces are:

- **CPU Register Space.** This consists of all the register addresses in the CPU register file.
- **CPU Control Register Space.** This consists of the Select Register (SR).
- **Memory Address Space.** This consists of the addresses of all locations in the main memory.
- **External I/O Address Space.** This consists of all external I/O ports addresses through which peripheral devices are accessed.
- **On-Chip I/O Address Space.** This consists of all internal I/O port addresses through which peripheral devices are accessed. Also, this addressing space contains registers to control the functionality of the device, giving status information.

2.2 CPU REGISTER SPACE

The Z380 register file is illustrated in Figure 2-1. Note that this figure shows the configuration of the register on the Z380 CPU, and the number of the register files may vary on future Superintegration devices. The Z380 CPU contains abundant register resources. At any given time, the program has immediate access to both primary and alternate registers in the selected register set. Changing register sets is a simple matter of an LDCTL instruction to program the Select Register (SR).

The CPU register file is divided into five groups of registers (an apostrophe indicates a register in the auxiliary registers).

- Four sets of Flag and Accumulator registers (F, A, F', A')
- Four sets of Primary and Working registers (B, C, D, E, H, L, B', C', D', E', H', L')
- Four sets of Index registers (IX, IY, IX', IY')
- Stack Pointer (SP)
- Program Counter, Interrupt register, Refresh register (PC, I, R)

Register addresses are either specified explicitly in the instruction or are implied by the semantics of the instruction.

2.2.1 Primary and Working Registers

The working register set is divided into two register files: the primary file and the alternate file (designated by prime ('')). Each file contains an 8-bit accumulator (A), a Flag register (F), and six 8-bit general-purpose registers (B, C, D, E, H, and L) with their Extended registers. Only one file can be active at any given time, although data in the inactive file can still be accessed by using EX R, R' instructions for the byte-wide registers, EX RR, RR' instructions for register pairs (either in 16-bit or 32-bit wide depending on the LW status). Exchange instructions allow the programmer to exchange the active file with the inactive file. The EX AF, AF', EXX, or EXALL instructions changes the register files in use. Upon reset, the primary register file in register set 0 is active. Changing register sets is a simple matter of an LDCTL instruction to program SR.

The accumulator is the destination register for 8-bit arithmetic and logical operations. The six general-purpose registers can be paired (BC, DE, and HL), and are extended to 32 bits by the extension to the register (with suffix "z"; BCz/DEz/HLz), to form three 32-bit general-purpose registers. The HL register serves as the 16-bit or 32-bit accumulator for word operations. Access to the Extended portion of the registers is possible using the SWAP instruction or word Load instructions in Long Word operation mode.

The Flag register contains eight status flags. Four can be individually used for control of program branching, two are used to support decimal arithmetic, and two are reserved. These flags are set or reset by various CPU operations. For details on Flag operations, refer to Section 5.2, "Flag Register."

2.2.2. Index Registers

The four index registers, IX, IX', IY, and IY', are extended to 32 bits by the extension to the register (with suffix "z"; IXz/IYz), to form 32-bit index registers. To access the Extended portion of the registers use the SWAP instruction or word Load instructions in Long Word operation mode. These Index registers hold a 32-bit base address that is used in the Index addressing mode.

Only one register of each can be active at any given time, although data in the inactive file can still be accessed by using EX IX, IX' and EX IY, IY' (either in 16-bit or 32-bit wide depending on the LW bit status). Index registers can also function as general-purpose registers with the upper and lower bytes of the lower 16 bits being accessed individually. These byte registers are called IXU, IXU', IXL, and IXL' for the IX and IX' registers, and IYU, IYU', IYL, and IYL' for the IY and IY' registers.

Selection of primary or auxiliary Index registers can be made by EXXX, EXXY, or EXALL instructions, or programming of SR. Upon reset, the primary registers in register set 0 is active. Changing register sets is a simple matter of an LDCTL instruction to program SR.

2.2.3. Interrupt Register

The Interrupt register (I) is used in interrupt modes 2 and 3 for /INT0 to generate a 32-bit indirect address to an interrupt service routine. The I register supplies the upper 24 or 16 bits of the indirect address and the interrupting peripheral supplies the lower eight or 16 bits. In Assigned Vectors mode for /INT3-/INT1, the upper 16 bits of the vector are supplied by the I register; bits 15-9 are supplied from the Assigned Vector Base register, and bits 8-0 are the assigned vector unique to each of /INT3-/INT1.

2.2.4. Program Counter

The Program Counter (PC) is used to sequence through instructions in the currently executing program and to generate relative addresses. The PC contains the 32-bit address of the current instruction being fetched from memory. In Native mode, the PC is effectively only 16 bits long, since the upper word [PC31-PC16] of the PC is forced to zero, and when carried from bit 15 to bit 16 (Lower word [PC15-PC0] to Upper word [PC31-PC16]) are inhibited in this mode. In Extended mode, the PC is allowed to increment across all 32 bits.

2.2.5. R Register

The R register can be used as a general-purpose 8-bit read/write register. The R register is not associated with the refresh controller and its contents are changed only by the user.

2.2.6. Stack Pointer

The Stack Pointer (SP) is used for saving information when an interrupt or trap occurs and for supporting subroutine calls and returns. Stack Pointer relative addressing allows parameter passing using the SP. The SP is 16 bits wide, but is extended by the SPz register to 32 bits wide.

Bits within a byte:

7	6	5	4	3	2	1	0

16-bit word at address n:

Least Significant Byte	Address n
Most Significant Byte	Address n+1

32-bit word at address n:

D7-0 (Least Significant Byte)	Address n
D15-8	Address n+1
D23-16	Address n+2
D31-24 (Most Significant Byte)	Address n+3

Memory addresses:

Even address (A0=0)	Odd address (A0=1)
Least Significant Byte	Most Significant Byte

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 2-2. Bit/Byte Ordering Conventions

CHAPTER 3

NATIVE EXTENDED MODE, WORD/LONG WORD MODE OF OPERATIONS AND DECODER DIRECTIONS

3.1 INTRODUCTION

The Z380™ CPU architecture allows access to 4 Gbytes (2^{32}) of memory addressing space, and 4G locations of I/O. It offers 16/32-bit manipulation capability while maintaining object-code compatibility with the Z80 CPU. In order to implement these capabilities and new instruction sets, it has two modes of operation for address manipulation (Native or Extended mode), two modes of operation for data manipulation (Word or Long Word mode), and a special set of new Decoder Directives.

On Reset, the Z380 CPU defaults in Native mode and Word mode. In this condition, it behaves exactly the same as the Z80 CPU, even though it has access to the entire 4 Gbytes of memory for data access and 4G locations of I/O space, access to the newly added registers which includes Extended registers and register banks, and the capability of executing all the Z380 instructions.

As described below, the Z380 CPU can be switched between Word mode and Long Word mode during operation through the SETC LW and RESC LW instructions, or Decoder Directives. The Native and Extended modes are a key exception— it defaults up in Native mode, and can be set to Extended mode by the instruction. Only Reset can return it to Native mode. Figure 3-1 illustrates the relationship between these modes of operation.

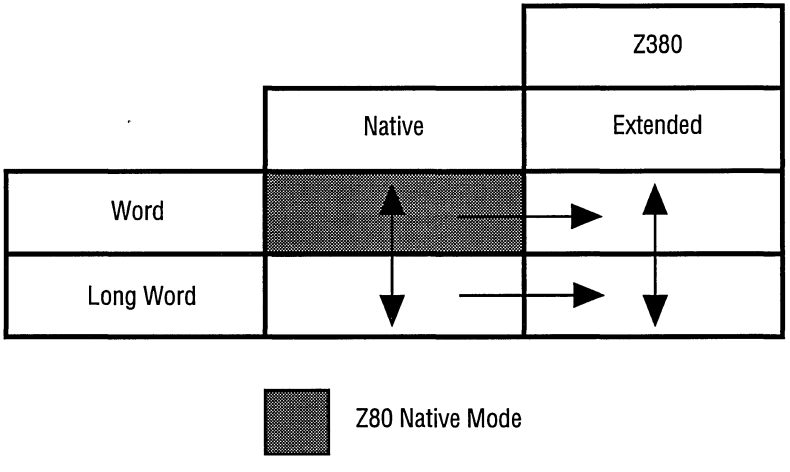


Figure 3-1. Z380™ CPU Operation Modes

For the instructions which work with the DDIR instructions, refer to Appendix D and E.

The Z380 CPU implements one instruction to switch to Extended mode from Native mode; SETC XM (set Extended mode) places the Z380 CPU in Extended mode. Once in Extended mode, only Reset can return it to Native mode. On Reset, the Z380 is in Native mode. Refer to Sections 4 and 5 for more examples.

3.4 WORD AND LONG WORD MODE OF OPERATION

The Z380 CPU can operate in either Word or Long Word mode. In Word mode (the Reset configuration), all word operations manipulate 16-bit quantities, and are compatible with the Z80 CPU 16-bit operations. In the Long Word mode, all word operations can manipulate 32-bit quantities. Note that the Native/Extended and Word/Long Word selections are independent of one another, as Word/Long Word pertains to data and operand address manipulation only. The Z380 CPU implements two instructions and two decoder directives to allow switching between these two modes; SETC LW (Set Long Word) and RESC LW (Reset Long Word) perform a global switch, while DDIR LW and DDIR W are decoder directives that select a particular mode only for the instruction that they precede.

Examples:

1. Effect of Word mode and Long Word mode

```
DDIR W
LD      BC, (HL)
```

Loads BC15-BC0 from the location (HL) and (HL+1), and BCz (BC31-BC16) remains unchanged.

```
DDIR LW
LD      BC, (HL)
```

Loads BC31-BC0 from the locations (HL) to (HL+3).

2. Immediate data load with DDIR instructions

```
DDIR IW,LW
LD      HL,12345678H
```

Loads 12345678H into HL31-HL0.

```
DDIR IB,LW
LD      HL,123456H
```

Loads 00123456H into HL31-HL0.
00H is appended as the Most significant byte as HL31-HL24.

```
DDIR LW
LD      HL,1234H
```

Loads 00001234H into HL31-HL0.
0000H is appended as the HL31-HL16 portion.

CHAPTER 4
ADDRESSING MODES AND DATA TYPES

4.1 INSTRUCTION

An instruction is a consecutive list of one or more bytes in memory. Most instructions act upon some data; the term operand refers to the data to be operated upon. For Z380™ CPU instructions, operands can reside in CPU registers, memory locations, or I/O ports (internal or external). The method used to designate the location of the operands for an instruction are called addressing modes. The Z380 CPU supports seven addressing modes; Register, Immediate, Indirect Register, Direct Address, Indexed, Program Counter Relative Address, and Stack Pointer Relative. A wide variety of data types can be accessed using these addressing modes.

4.2 ADDRESSING MODE DESCRIPTIONS

The following pages contain descriptions of the addressing modes for the Z380 CPU. Each description explains how the operand's location is calculated, indicates which address spaces can be accessed with that particular addressing mode, and gives an example of an instruction using that mode, illustrating the assembly language format for the addressing modes.

4.2.1 Register (R, RX)

When this addressing mode is used, the instruction processes data taken from one of the 8-bit registers A, B, C, D, E, H, L, IXU, IXL, IYU, IYL, one of the 16-bit registers BC, DE, HL, IX, IY, SP, or one of the special byte registers I or R.

Storing data in a register allows shorter instructions and faster execution that occur with instructions that access memory.

Instruction
OPERATION REGISTER → OPERAND

The operand value is the contents of the register.

The operand is always in the register address space. The register length (byte or word) is specified by the instruction opcode. In the case of Long Word register operation, it is specified either through the SETC LW instruction or the DDIR LW decoder directive.

Example of R mode:

1. **Load register in Word mode.**

```
DDIR W    ;Next instruction in Word mode
LD BC,HL  ;Load the contents of HL into BC
```

	BCz	BC	HLz	HL
Before instruction execution	1234	5678	9ABC	DEF0
After instruction execution	1234	DEF0	9ABC	DEF0

2. **Load register in Long Word mode.**

```
DDIR LW   ;Next instruction in Long Word mode
LD BC,HL  ;Load the contents of HL into BC
```

	BCz	BC	HLz	HL
Before instruction execution	1234	5678	9ABC	DEF0
After instruction execution	9ABC	DEF0	9ABC	DEF0

4.2.2 Immediate (IM)

When the Immediate addressing mode is used, the data processed is in the instruction.

The Immediate addressing mode is the only mode that does not indicate a register or memory address as the source operand.

4.2.4 Direct Address (DA)

When Direct Address mode is used, the data processed is at the location whose memory or I/O port address is in the instruction.

Instruction		Memory or
OPERATION		I/O Port
ADDRESS	→	OPERAND

The operand value is the contents of the location whose address is in the instruction.

Depending on the instruction, the operand specified by DA mode is either in the I/O address space (I/O instruction) or memory address space (all other instructions).

This mode is also used by Jump and Call instructions to specify the address of the next instruction to be executed. (The address serves as an immediate value that is loaded into the program counter.)

Also, DDIR Immediate Data Directives are used to expand the direct address to 24 or 32 bits. Operand width is affected by LW bit status for the load and exchange instructions.

Example of DA mode:

1. **Load BC register from memory location 00005E22H in Word mode**

```
LD BC, (5E22H)    ;Load BC with the data in address
                  ;00005E22H
```

	BC	
Before instruction execution	1234	
After instruction execution	0301	
Memory location	00005E22	01
	00005E23	03

2. **Load BC register from memory location 12345E22H in Word mode**

```
DDIR IW                ;extend direct address by one word
LD BC, (12345E22H)     ;Load BC with the data in address
                       ;12345E22H
```

	BC	
Before instruction execution	1234	
After instruction execution	0301	
Memory location	12345E22	01
	12345E23	03

3. **Load BC register from memory location 12345E22H in Long Word mode**

```
DDIR IW,LW             ;extend direct address by one word,
                       ;and operation in Long Word
LD BC, (12345E22H)     ;Load BC with the data in address
                       ;12345E22H
```

	BCz	BC
Before instruction execution	1234	5678
After instruction execution	0705	0301
Memory location	12345E22	01
	12345E23	03
	12345E24	05
	12345E25	07

2. Load accumulator from location (IX-1) in Extended mode

```
SETC  XM          ;Set Extended mode
LD    A, (IX-1)   ;Load into the accumulator the
                  ;contents of the memory location
                  ;whose address is one less than
                  ;the contents of IX
```

	A	IXz	IX
Before instruction execution	01	0001	0000
After instruction execution	23	0001	0000
Memory location	0000FFFF	23	

Address calculation: In Extended mode, 0FFH encoding in the instruction is sign extended to a 32-bit value before the address calculation, but calculation is done in modulo 2^{32} and takes into account the index register's extended portion.

```
  00010000
+ FFFFFFFF
  0000FFFF
```

4.2.6 Program Counter Relative Mode (RA)

The Program Counter Relative Addressing mode is used by certain program control instructions to specify the address of the next instruction to be executed (specifically, the sum of the Program Counter value and the displacement value is loaded into the Program Counter). Relative addressing allows reference forward or backward from the current Program Counter value; it is used for program control instructions such as Jumps and Calls that access constants in the memory.

As a displacement, an 8-bit, 16-bit, or 24-bit value can be used. The address to be loaded into the Program Counter is computed by adding the two's complement signed displacement specified in the instruction to the current Program Counter.

Also, in Native mode,

```
Instruction     PC                MEMORY
OPERATION       ADDRESS     →+    OPERAND
DISPLACEMENT                —↑
```

Example of RA mode:

1. Jump relative in Native mode, 8-bit displacement

```
JR    $-2    ;Jumps to the location
             ;(Current PC value) – 2
             ;'$' represents for current PC value
             ;This instruction jumps to itself.
             ;since after the execution of this instruction,
             ;PC points to the next instruction.
```

Note that computation of the effective address is affected by the mode of operation (Native or Extended). In Native mode, address computation is done in modulo 2^{16}, and the PC Extend (PC31-PC16) is forced to 0 and will not affect this portion. In Extended mode, address computation is done is modulo 2^{32}, and will affect the contents of PC extend if there is a carry or borrow operation.

4.2.7 Stack Pointer Relative Mode (SR)

For Stack Pointer Relative addressing mode, the data processed is at the location whose address is the contents of the Stack Pointer, offset by an 8-bit displacement in the instruction.

The Stack Pointer Relative address is computed by adding the 8-bit two's complement signed displacement specified in the instruction to the contents of the SP, also specified by the instruction. Stack Pointer Relative addressing mode is used to specify data items to be found in the stack, such as parameters passed to procedures.

Offset portion can be expanded to 16 or 24 bits by using DDIR immediate instructions (DDIR IB for a 16-bit offset, DDIR IW for a 24-bit offset).

Note that computation of the effective address is affected by the operation mode (Native or Extended). In Native mode, address computation is done in modulo 2^{16}, meaning computation is done in 16-bit and does not affect upper half of the SP portion for calculation (wrap around within the 16-bit). In Extended mode, address computation is done in modulo 2^{32}.

Also, the size of the data transfer is affected by the LW mode bit. In Word mode, transfer is done in 16 bits, and in Long Word mode, transfer is done in 32 bits.

Instruction	SP		
OPERATION	ADDRESS	—I	MEMORY
DISPLACEMENT		—+	OPERAND

Example of SR mode:

1. Load HL from location (SP – 4) in Native mode, Word mode

```
LD HL, (SP–4)    ;Load into the HL from the
                 ;contents of the memory location
                 ;whose address is four less than
                 ;the contents of SP.
                 ;Assume it is in Native/Word mode.
```

	HLz	HL	SPz	SP
Before instruction execution	1234	5678	07FF	7F00
After instruction execution	EFCD	AB89	07FF	7F00
Memory location	07FF7EFC		89	
	07FF7EFD		AB	

Address calculation: In Native mode, FCH (–4 in Decimal) encoding in the instruction is sign extended to a 16-bit value before the address calculation. Calculation is done in modulo 2^{16} and does not take into account the Stack Pointer's extended portion.

```
  7F00
+ FFFC
  7EFC
```

4.3 DATA TYPES

The Z380 CPU can operate on bits, binary-coded decimal (BCD) digits (four bits), bytes (eight bits), words (16 bits or 32 bits), byte strings, and word strings. Bits in registers can be set, cleared, and tested.

The basic data type is a byte, which is also the basic accessible element in the register, memory, and I/O address space. The 8-bit load, arithmetic, logical, shift, and rotate instructions operate on bytes in registers or memory. Bytes can be treated as logical, signed numeric, or unsigned numeric value.

Words are operated on in a similar manner by the word load, arithmetic, logical, and shift and rotate instructions.

Operation on 2-byte words is also supported. Sixteen-bit load and arithmetic instructions operate on words in registers or memory; words can be treated as signed or unsigned numeric values. I/O reads and writes can be 8-bit or 16-bit operations. Also, the Z380 CPU architecture supports operation in Long Word mode to handle a 32-bit address manipulation. For that purpose, 16-bit wide registers originally on the Z80 have been expanded to 32 bits wide, along with the support of the arithmetic instruction needed for a 32-bit address manipulation.

Bits are fully supported and addressed by number within a byte (see Figure 2-2). Bits within byte registers or memory locations can be tested, set, or cleared.

Operation on binary-coded decimal (BCD) digits are supported by Decimal Adjust Accumulator (DAA) and Rotate Digit (RLD and RRD) instructions. BCD digits are stored in byte registers or memory locations, two per byte. The DAA instruction is used after a binary addition or subtraction of BCD numbers. Rotate Digit instructions are used to shift BCD digit strings in memory.

Strings of up to 65536 (64K) bytes of Byte data or Word data can be manipulated by the Z380 CPU's block move, block search, and block I/O instructions. The block move instructions allow strings of bytes/words in memory to be moved from one location to another. Block search instructions provide for scanning strings of bytes/words in memory to locate a particular value. Block I/O instructions allow strings of bytes or words to be transferred between memory and a peripheral device.

Arrays are supported by Indexed mode (with 8-bit, 16-bit, or 24-bit displacement). Stack is supported by the Indexed and the Stack Pointer Relative addressing modes, and by special instructions such as Call, Return, Push, and Pop.

ZiLOG

CHAPTER 5
INSTRUCTION SET

5.1 INTRODUCTION

The Z380™ CPU instruction set is a superset of the Z80 CPU and the Z180 MPU; the Z380 CPU is opcode compatible with the Z80 CPU/Z180 MPU. Thus, a Z80/Z180 program can be executed on a Z380 CPU without modification. The instruction set is divided into 12 groups by function:

- 8-Bit Load/Exchange Group
- 16/32-Bit Load, Exchange, SWAP and Push/Pop Group
- Block Transfers, and Search Group
- 8-Bit Arithmetic and Logic Operations
- 16/32-Bit Arithmetic Operations
- 8-Bit Bit Manipulation, Rotate and Shift Group
- 16-Bit Rotates and Shifts
- Program Control Group
- Input and Output Operations for External I/O Space
- Input and Output Operations for Internal I/O Space
- CPU Control Group
- Decoder Directives

This chapter describes the instruction set of the Z380 CPU. Flags and condition codes are discussed in relation to the instruction set. Then, the interpretability of instructions and trap are discussed. The last part of this chapter is a detailed description of each instruction, listed in alphabetical order by mnemonic. This section is intended as a reference for Z380 CPU programmers. The entry for each instruction contains a complete description of the instruction, including addressing modes, assembly language mnemonics, and instruction opcode formats.

5.2 PROCESSOR FLAGS

The Flag register contains six bits of status information that are set or cleared by CPU operations (Figure 5-1). Four of these bits are testable (C, P/V, Z, and S) for use with conditional jump, call, or return instructions. Two flags are not testable (H and N) and are used for binary-coded decimal (BCD) arithmetic.

S	Z	X	H	X	P/V	N	C
7	6	5	4	3	2	1	0

Figure 5-1. Flag Register

The Flag register provides a link between sequentially executed instructions, in that the result of executing one instruction may alter the flags, and the resulting value of the flags can be used to determine the operation of a subsequent instruction. The program control instructions, whose operation depends on the state of the flags, are the Jump, Jump Relative, subroutine Call, Call Relative, and subroutine Return instructions; these instructions are referred to as conditional instructions.

5.2.7 Condition Codes

The Carry, Zero, Sign, and Parity/Overflow flags are used to control the operation of the conditional instructions. The operation of these instructions is a function of the state of one of the flags. Special mnemonics called condition codes are used to specify the flag setting to be tested during execution of a conditional instruction; the condition codes are encoded into a 3-bit field in the instruction opcode itself.

Table 5-1 lists the condition code mnemonic, the flag setting it represents, and the binary encoding for each condition code.

Table 5-1. Condition codes

Condition Codes for Jump, Call, and Return Instructions

Mnemonic	Meaning	Flag Setting	Binary Code
NZ	Not Zero*	Z = 0	000
Z	Zero*	Z = 1	001
NC	No Carry*	C = 0	010
C	Carry*	C = 1	011
NV	No Overflow	V = 0	100
PO	Parity Odd	V = 0	100
V	Overflow	V = 1	101
PE	Parity Even	V = 1	101
NS	No Sign	S = 0	110
P	Plus	S = 0	110
S	Sign	S = 1	111
M	Minus	S = 1	111

*Abbreviated set

Condition Codes for Jump Relative and Call Relative Instructions

Mnemonic	Meaning	Flag Setting	Binary Code
NZ	Not Zero	Z = 0	100
Z	Zero	Z = 1	101
NC	No Carry	C = 0	110
C	Carry	C = 1	111

5.3.8. Long Word Mode (LW)

This bit controls the Long Word/Word mode selection for the Z380 CPU. This bit is set by the SETC LW instruction and cleared by the RESC LW instruction. When this bit is set, the Z380 CPU is in Long Word mode; when this bit is cleared the Z380 CPU is in Word mode. Reset clears this bit. Note that individual Word load and exchange instructions may be executed in either Word or Long Word mode using the DDIR W and DDIR LW decoder directives.

5.3.9. Interrupt Enable Flag (IEF)

This bit is the master Interrupt Enable for the Z380 CPU. This bit is set by the EI instruction and cleared by the DI instruction, or on acknowledgment of an interrupt request. When this bit is set, interrupts are enabled; when this bit is cleared, interrupts are disabled. Reset clears this bit.

5.3.10. Interrupt Mode (IM)

This 2-bit field controls the interrupt mode for the /INT0 interrupt request. These bits are controlled by the IM instructions (00 = IM 0, 01 = IM 1, 10 = IM 2, 11 = IM 3). Reset clears both of these bits, selecting Interrupt Mode 0.

5.3.11. Lock (LCK)

This bit controls the Lock/Unlock status of the Z380 CPU. This bit is set by the SETC LCK instruction and cleared by the RESC LCK instruction. When this bit is set, no bus requests will be accepted, providing exclusive access to the bus by the Z380 CPU. When this bit is cleared, the Z380 CPU will grant bus requests in the normal fashion. Reset clears this bit.

5.3.12. AF or AF' Register Select (AF')

This bit controls and reports whether AF or AF' is the currently active pair of registers. AF is selected when this bit is cleared, and AF' is selected when this bit is set. Reset clears this bit, selecting AF.

5.4 INSTRUCTION EXECUTION AND EXCEPTIONS

Three types of exception conditions—interrupts, trap, and Reset—can alter the normal flow of program execution. Interrupts are asynchronous events generated by a device external to the CPU; peripheral devices use interrupts to request service from the CPU. Trap is a synchronous event generated internally in the CPU by executing undefined instructions. Reset is an asynchronous event generated by outside circuits. It terminates all current activities and puts the CPU into a known state. Interrupts and Traps are discussed in detail in Chapter 6, and Reset is discussed in detail in Chapter 7. This section examines the relationship between instructions and the exception conditions.

5.4.1 Instruction Execution and Interrupts

When the CPU receives an interrupt request, and it is enabled for interrupts of that class, the interrupt is normally processed at the end of the current instruction. However, the block transfer and search instructions are designed to be interruptible so as to minimize the length of time it takes the CPU to respond to an interrupt. If an interrupt request is received during a block move, block search, or block I/O instruction, the instruction is suspended after the current iteration. The address of the instruction itself, rather than the address of the following instruction, is saved on the stack, so that the same instruction is executed again when the interrupt handler executes an interrupt return instruction. The contents of the repetition counter and the registers that index into the block operands are such that, after each iteration, when the instruction is reissued upon returning from an interrupt, the effect is the same as if the instruction were not interrupted. This assumes, of course, that the interrupt handler preserves the registers.

5.4.2 Instruction Execution and Trap

The Z380 MPU generates a Trap when an undefined opcode is encountered. The action of the CPU in response to Trap is to jump to address 00000000H with the status bit(s) set. This response is similar to the Z180 MPU's action on execution of an undefined instruction. The Trap is enabled immediately after reset, and it is not maskable. This feature can be used to increase software reliability or to implement "extended" instructions. An undefined opcode can be fetched from the instruction stream, or it can be returned as a vector in an interrupt acknowledge transaction in Interrupt mode 0.

Since it jumps to address 00000000H, it is necessary to have a Trap handling routine at the beginning of the program if processing is to proceed. Otherwise, it behaves just like a reset for the CPU. For a detailed description, refer to Chapter 6.

5.5.2 16-Bit and 32-Bit Load, Exchange, SWAP, and PUSH/POP Group

This group of load, exchange, and PUSH/POP instructions (Table 5-4) allows one or two words of data (two bytes equal one word) to be transferred between registers and memory.

The exchange instructions (Table 5-5) allow for switching between the primary and alternate register files, exchanging the contents of two register files, exchanging the contents of an addressing register with the top word on the stack. For possible combinations of the word exchange instructions, refer to Table 5-5. The 16-bit and 32-bit loads include transfer between registers and memory and immediate loads of registers or memory. The Push and Pop stack instructions are also included in this group. None of these instructions affect the CPU flags, except for EX AF, AF'.

Table 5-6 has the supported source/destination combination for the 16-bit and 32-bit load instructions. The transfer size, 16-bit or 32-bit, is determined by the status of LW bit in SR, or by DDIR Decoder Directives.

PUSH/POP instructions are used to save/restore the contents of a register onto the stack. It can be used to exchange data between procedures, save the current register file on context switching, or manipulate data on the stack, such as return addresses. Supported sources are listed in Table 5-7.

Swap instructions allows swapping of the contents of the Word wide register (BC, DE, HL, IX, or IY) with its Extended portion. These instructions are useful to manipulate the upper word of the register to be set in Word mode. For example, when doing data accesses, other than 00000000H-0000FFFFH address range, use this instruction to set "data frame" addresses.

This group of instructions is affected by the status of the LW bit in SR (Select Register), and Decoder Directives which specifies the operation mode in Word or Long Word.

Table 5-4. 16-Bit and 32-Bit Load, Exchange, PUSH/POP Group Instructions

Instruction Name	Format	Note
Exchange Word/Long Word Registers	EX dst,src	See Table 5-5
Exchange Byte/Word Registers with Alternate Bank	EXX	
Exchange Register Pair with Alternate Bank	EX RR,RR'	RR = AF, BC, DE, or HL
Exchange Index Register with Alternate Bank	EXXX EXXY	
Exchange All Registers with Alternate Bank	EXALL	
Load Word/Long Word Registers	LD dst,src	See Table 5-6
	LDW dst,src	See Table 5-6
POP	POP dst	See Table 5-7
PUSH	PUSH src	See Table 5-7
Swap Contents of D31-D16 and D15-D0	SWAP dst	dst = BC, DE, HL, IX, or IY

Table 5-5. Supported Source and Destination Combination for 16-Bit and 32-Bit Exchange Instructions

	Source				
Destination	**BC**	**DE**	**HL**	**IX**	**IY**
BC	√	√	√	√	
DE		√	√	√	
HL			√	√	
IX				√	
(SP)		√	√	√	

Note: √ are supported combinations. The exchange instructions which designate IY register as destination are covered by the other combinations. These Exchange Word instructions are affected by Long Word mode.

has to be an even number (D0 = 0) in Word mode transfer, and a multiple of four in Long Word mode (D1 and D0 are both 0). Also, in Word or Long Word Block transfer, memory pointer values are recommended to be even numbers so the number of the transactions will be minimized.

Note that regardless of the Z380's operation mode, Native or Extended, memory pointer increment/decrement will be done in modulo 2^{32}. For example, if the operation is LDI and HL31-HL0 (HLz and HL) hold 0000FFFF, after the operation the value in the HL31-HL0 will be 0010000.

Table 5-8. Block Transfer and Search Group

Instruction Name	Format
Compare and Decrement	CPD
Compare, Decrement and Repeat	CPDR
Compare and Increment	CPI
Compare, Increment and Repeat	CPIR
Load and Decrement	LDD
Load , Decrement and Repeat	LDDI
Load and Increment	LDI
Load, Increment and Repeat	LDIR
Load and Decrement in Word/Long Word	LDDW
Load, Decrement and Repeat in Word/Long Word	LDDRW
Load and Increment in Word/Long Word	LDIW
Load, Increment and Repeat in Word/Long Word	LDIRW

5.5.4 8-bit Arithmetic and Logical Group

This group of instructions (Table 5-9) perform 8-bit arithmetic and logical operations. The Add, Add with Carry, Subtract, Subtract with Carry, AND, OR, Exclusive OR, and Compare takes one input operand from the accumulator and the other from a register, from immediate data in the instruction itself, or from memory. For memory addressing modes, follows are supported—Indirect Register, Indexed, and Direct Address—except multiplies, which returns the 16-bit result to the same register by multiplying the upper and lower bytes of one of the register pair (BC, DE, HL, or SP).

The Increment and Decrement instructions operate on data in a register or in memory; all memory addressing modes are supported. These instructions operate only on the accumulator—Decimal Adjust, Complement, and Negate. The final instruction in this group, Extend Sign, sets the CPU flags according to the computed result.

The EXTS instruction extends the sign bit and leaves the result in the HL register. If it is in Long Word mode, HLz (HL31-HL16) portion is also affected.

The TST instruction is a nondestructive AND instruction. It ANDs "A" register and source, and changes flags according to the result of operation. Both source and destination values will be preserved.

5

Table 5-9. Supported Source/Destination for 8-Bit Arithmetic and Logic Group

Instruction Name	Format	src/ dst	A	B	C	D	E	H	L	IXH	IXL	IYH	IYL	n	(HL)	(IX+d)	(IY+x)
Add With Carry (Byte)	ADC A,src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Add (Byte)	ADD A,src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
AND	AND [A,]src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Compare (Byte)	CP [A,]src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Complement Accumulator	CPL [A]	dst	√														
Decimal Adjust Accumulator	DAA	dst	√														
Decrement (Byte)	DEC dst	dst	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Extend Sign (Byte)	EXTS [A]	dst	√														
Increment (Byte)	INC dst	dst	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Multiply (Byte)	MLT src	Note 1															
Negate Accumulator	NEG [A]	dst	√														
OR	OR [A,]src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Subtract with Carry (Byte)	SBC A,src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Subtract (Byte)	SUB [A,]src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Nondestructive Test	TST dst	src	√	√	√	√	√	√	√					√	√		
Exclusive OR	XOR [A,]src	src	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

Note 1: dst = BC, DE, HL, or SP.

5.5.6 8-Bit Manipulation, Rotate and Shift Group

Instructions in this group (Table 5-11) test, set, and reset bits within bytes, and rotate and shift byte data one bit position. Bits to be manipulated are specified by a field within the instruction. Rotate can optionally concatenate the Carry flag to the byte to be manipulated. Both left and right shifting is supported. Right shifts can either shift 0 into bit 7 (logical shifts), or can replicate the sign in bits 6 and 7 (arithmetic shifts). All these instructions, Set Bit and Reset Bit, set the CPU flags according to the calculated result; the operand can be a register or a memory location specified by the Indirect Register or Indexed addressing mode.

The RLD and RRD instructions are provided for manipulating strings of BCD digits; these rotate 4-bit quantities in memory specified by the Indirect Register. The low-order four bits of the accumulator are used as a link between rotation of successive bytes.

Table 5-11. Bit Set/Reset/Test, Rotate and Shift Group

Instruction Name	Format	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
Bit Test	BIT dst	√	√	√	√	√	√	√	√	√	√
Reset Bit	RES dst	√	√	√	√	√	√	√	√	√	√
Rotate Left	RL dst	√	√	√	√	√	√	√	√	√	√
Rotate Left Accumulator	RLA	√									
Rotate Left Circular	RLC dst	√	√	√	√	√	√	√	√	√	√
Rotate Left Circular (Accumulator)	RLCA	√									
Rotate Left Digit	RLD	√									
Rotate Right	RR dst	√	√	√	√	√	√	√	√	√	√
Rotate Right Accumulator	RRA	√									
Rotate Right Circular	RRC dst	√	√	√	√	√	√	√	√	√	√
Rotate Right Circular (Accumulator)	RRCA	√									
Rotate Right Digit	RRD	√									
Set Bit	SET dst	√	√	√	√	√	√	√	√	√	√
Shift Left Arithmetic	SLA dst	√	√	√	√	√	√	√	√	√	√
Shift Right Arithmetic	SRA dst	√	√	√	√	√	√	√	√	√	√
Shift Right Logical	SRL	√	√	√	√	√	√	√	√	√	√

5

5.5.7 16-Bit Manipulation, Rotate and Shift Group

Instructions in this group (Table 5-12) rotate and shift word data one bit position. Rotate can optionally concatenate the Carry flag to the word to be manipulated. Both left and right shifting is supported. Right shifts can either shift 0 into bit 15 (logical shifts), or can replicate the sign in bits 14 and 15 (arithmetic shifts). The operand can be a register pair or memory location specified by the Indirect Register or Indexed addressing mode, as shown below.

Table 5-12. 16-Bit Rotate and Shift Group.

		Destination								
Instruction Name	Format	BC	DE	HL	IX	IY	(HL)	(HL)	(IX+d)	(IY+d)
Rotate Left Word	RLW dst	√	√	√	√	√	√	√	√	√
Rotate Left Circular Word	RLCW dst	√	√	√	√	√	√	√	√	√
Rotate Right Word	RRW dst	√	√	√	√	√	√	√	√	√
Rotate Right Circular Word	RRCW dst	√	√	√	√	√	√	√	√	√
Shift Left Arithmetic Word	SLAW dst	√	√	√	√	√	√	√	√	√
Shift Right Arithmetic Word	SRAW dst	√	√	√	√	√	√	√	√	√
Shift Right Logical Word	SRLW	√	√	√	√	√	√	√	√	√

5.5.9 External Input/Output Instruction Group

This group of instructions (Table 5-14) are used for transferring a byte, a word, or string of bytes or words between peripheral devices and the CPU registers or memory. Byte I/O port addresses transfer bytes on D7-D0 only. These 8-bit peripherals in a 16-bit data bus environment must be connected to data line D7-D0. In an 8-bit data bus environment, word I/O instructions to external I/O peripherals should not be used; however, on-chip peripherals which is external to the CPU core and assigned as word I/O device can still be accessed by word I/O instructions.

The instructions for transferring a single byte (IN, OUT) can transfer data between any 8-bit CPU register or memory address specified in the instruction and the peripheral port specified by the contents of the C register. The IN instruction sets the CPU flags according to the input data; however, special instructions restricted to using the CPU accumulator and Direct Address mode and do not affect the CPU flags. Another variant tests an input port specified by the contents of the C register and sets the CPU flags without modifying CPU registers or memory.

The instructions for transferring a single word (INW, OUTW) can transfer data between the register pair and the peripheral port specified by the contents of the C register. For Word I/O, the contents of B, D, or H appear on D7-D0 and the contents of C, E, or L appear D15-D7. These instructions do not affect the CPU flags.

Also, there are I/O instructions available which allow to specify 16-bit absolute I/O address (with DDIR decoder directives, a 24-bit or 32-bit address is specified) is available. These instructions do not affect the CPU flags.

The remaining instructions in this group form a powerful and complete complement of instructions for transferring blocks of data between I/O ports and memory. The operation of these instructions is very similar to that of the block move instructions described earlier, with the exception that one operand is always an I/O port whose address remains unchanged while the address of the other operand (a memory location) is incremented or decremented. In Word mode of transfer, the counter (i.e., BC register) holds the number of transfers, rather than number of bytes to transfer in memory-to-memory word block transfer. Both byte and word forms of these instructions are available. The automatically repeating forms of these instructions are interruptible, like memory-to-memory transfer.

The I/O addresses output on the address bus is dependant on the I/O instruction, as listed in Table 2-1.

5.5.10 Internal I/O Instruction Group

This group (Table 5-15) of instructions is used to access on-chip I/O addressing space on the Z380 CPU. This group consists of instructions for transferring a byte from/ to Internal I/O locations and the CPU registers or memory, or a blocks of bytes from the memory to the same size of Internal I/O locations for initialization purposes. These instructions are originally assigned as newly added I/O instructions on the Z180 MPU to access Page 0 I/O addressing space. There is 256 Internal I/O locations, and all of them are byte-wide. When one of these I/O instructions is executed, the Z380 MPU outputs the register address being accessed in a pseudo transaction of two BUSCLK durations cycle, with the address signals A31-A8 at 0. In the pseudo transactions, all bus control signals are at their inactive state.

The instructions for transferring a single byte (IN0, OUT0) can transfer data between any 8-bit CPU register and the Internal I/O address specified in the instruction. The IN0 instruction sets the CPU flags according to the input data; however, special instructions which do not have a destination in the instruction with Direct Address (IN0 (n)), do not affect the CPU register, but alters flags accordingly. Another variant, the TSTIO instruction, does a logical AND to the instruction operand with the internal I/O location specified by the C register and changes the CPU flags without modifying CPU registers or memory.

The remaining instructions in this group form a powerful and complete complement of instructions for transferring blocks of data from memory to Internal I/O locations. The operation of these instructions is very similar to that of the block move instructions described earlier, with the exception that one operand is always an Internal I/O location whose address also increments or decrements by one automatically, Also, the address of the other operand (a memory location) is incremented or decremented. Since Internal I/O space is byte-wide, only byte forms of these instructions are available. Automatically repeating forms of these instructions are interruptible, like memory-to-memory transfer.

Table 5-15. Internal I/O Instruction Group

Instruction Name	Format	
Input from Internal I/O Location	IN0 dst,(n)	dst=A, B, C, D, E, H or L
Input from Internal I/O Location(Nondestructive)	IN0 (n)	
Test I/O	TSTIO n	
Output to Internal I/O Location	OUT0 (n),src	src=A, B, C, D, E, H or L
Output to Internal I/O and Decrement	OTDM	
Output to Internal I/O and Increment	OTIM	
Output to Internal I/O, Decrement and Repeat	OTDMR	
Output to Internal I/O, Increment and Repeat	OTIMR	

Currently, the Z380 CPU core has the following registers as a part of the CPU core:

Register Name	Internal I/O address
Interrupt Enable Register	16H
Assigned Vector Base Register	17H
Trap Register	18H
Chip Version ID Register	0FFH

Chip Version ID register returns one byte data, which indicates the version of the CPU, or the specific implementation of the Z380 CPU based Superintegration device. Currently, the value 00H is assigned to the Z380 MPU, and other values are reserved.

For the other three registers, refer to Chapter 6, "Interrupt and Trap."

Also, the Z380 MPU has registers to control chip selects, refresh, waits, and I/O clock divide to Internal I/O address 00H to 10H. For these register, refer to Z380 MPU Product specification.

5.5.12 Decoder Directives

The Decoder Directives (Table 5-17) are a special instructions to expand the Z80 instruction set to handle the Z380's 4 Gbytes of linear memory addressing space. For details on this instruction, refer to Chapter 3.

Table 5-17. Decoder Directive Instructions

DDIR W	Word Mode
DDIR IB,W	Immediate Byte, Word Mode
DDIR IW,W	Immediate Word, Word Mode
DDIR IB	Immediate Byte
DDIR LW	Long Word Mode
DDIR IB,LW	Immediate Byte, Long Word Mode
DDIR IW,LW	Immediate Word, Long Word Mode
DDIR IW	Immediate Word

5.6 NOTATION AND BINARY ENCODING

The rest of this chapter consists of a detailed description of the Z380 CPU instructions, arranged in alphabetical order by mnemonic. This section describes the notational conventions used in the instruction descriptions and the binary encoding for register fields within the instruction's operation codes (opcodes).

The description of each instruction begins on a new page. The instruction mnemonic and name are printed in bold letters at the top of each page to enable the reader to easily locate a desired description. The assembly language syntax is then given in a single generic form that covers all the variants of the instruction, along with a list of applicable addressing modes. This is followed by a description of the operation performed by the instruction in "pseudo Pascal" fashion, a detailed description, a listing of all the flags that are affected by the instruction, and illustrations of the opcodes for all variants of the instruction.

Symbols. The following symbols are used to describe the instruction set.

n	An 8-bit constant
nn	A 16-bit constant
d	An 8-bit offset. (two's complement)
src	Source of the instruction
dst	Destination of the instruction
SR	Select Register
R	Any register. In Word operation, any register pair. Any 8-bit register (A, B, C, D, E, H, or L) for Byte operation.
IR	Indirect register
RX	Indexed register (IX or IY) in Word operation, IXH, IXL, IYH, or IYL for Byte operation.
SP	Current Stack Pointer
(C)	I/O Port pointed by C register
cc	Condition Code
[ ]	Optional field
( )	Indirect Address Pointer or Direct Address

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location.

The symbol "↔" indicates that the source and destination is swapping. For example,

dst ↔ src

indicates that the source data is swapped with the data in the destination; after the operation, data at "src" is in the "dst" location, and data in "dst " is in the "src" location.

The notation "dst (b)" is used to refer to bit "b" of a given location, "dst(m-n)" is used to refer to bit location m to n of the destination. For example,

HL(7) specifies bit 7 of the destination.
and
HL(23-16) specifies bit location 23 to 16 of the HL register.

Flags. The F register contains the following flags followed by symbols.

S	Sign Flag
Z	Zero Flag
H	Half Carry Flag
P/V	Parity/Overflow Flag
N	Add/Subtract Flag
C	Carry Flag

Table 5-18. Execution Time

Operation	Byte	Word	Word	Long	Long	Long	Long	Long
Sequence	B	W	B/B	W/W	W/B/B	B/W/B	B/B/W	B/B/B/B
Memory Read	3-4	3-4	5-6	5-6	7-8	7-8	7-8	9-10
Memory Write	0-1	0-1	2-3	2-3	4-5	4-5	4-5	6-7
Internal I/O Read	3-4	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Internal I/O Write	0-1	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1X External I/O Read	4-5	4-5	N/A	N/A	N/A	N/A	N/A	N/A
1X External I/O Write	1-2	1-2	N/A	N/A	N/A	N/A	N/A	N/A
2X External I/O Read	9-11	9-11	N/A	N/A	N/A	N/A	N/A	N/A
2X External I/O Write	1-3	1-3	N/A	N/A	N/A	N/A	N/A	N/A
4X External I/O Read	17-21	17-21	N/A	N/A	N/A	N/A	N/A	N/A
4X External I/O Write	1-5	1-5	N/A	N/A	N/A	N/A	N/A	N/A
6X External I/O Read	25-31	25-31	N/A	N/A	N/A	N/A	N/A	N/A
6X External I/O Write	1-7	1-7	N/A	N/A	N/A	N/A	N/A	N/A
8X External I/O Read	33-41	33-41	N/A	N/A	N/A	N/A	N/A	N/A
8X External I/O Write	1-9	1-9	N/A	N/A	N/A	N/A	N/A	N/A

Note: Units are in Clocks. "N/A" is not applicable for that particular transaction.

ADC
ADD WITH CARRY (WORD)

ADC HL,src dst = HL
src = BC, DE, HL, SP

Operation: HL(15-0) ← HL(15-0) + src(15-0) + C

The source operand together with the Carry flag is added to the HL register and the sum is stored in the HL register. The contents of the source are unaffected. Two's complement addition is performed.

Flags:

S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Set if there is a carry from bit 11 of the result; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
N: Cleared
C: Set if there is a carry from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	ADC HL,R	11101101 01rr1010	2	

Field Encodings: rr: 00 for BC, 01 for DE, 10 for HL, 11 for SP

5

ADD
ADD (BYTE)

ADD A,src src = R, RX, IM, IR, X

Operation: A ← A + src

The source operand is added to the accumulator and the sum is stored in the accumulator. The contents of the source are unaffected. Two's complement addition is performed.

Flags:
- S: Set if the result is negative; cleared otherwise
- Z: Set if the result is zero; cleared otherwise
- H: Set if there is a carry from bit 3 of the result; cleared otherwise
- V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
- N: Cleared
- C: Set if there is a carry from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	ADD A,R	10000-r-	2	
RX:	ADD A,RX	11y11101 1000010w	2	
IM:	ADD A,n	11000110 ——n—	2	
IR:	ADD A,(HL)	10000110	2+r	
X:	ADD A,(XY+d)	11y11101 10000110 ——d—	4+r	I

Field Encodings:
- r: per convention
- y: 0 for IX, 1 for IY
- w: 0 for high byte, 1 for low byte

5

ADD
ADD TO STACK POINTER (WORD)

ADD SP,src src = IM

Operation:

```
if (XM) then begin
    SP(31-0)   ←   SP(31-0) + src(31-0)
    end
else begin
    SP(15-0)   ←   SP(15-0) + src(15-0)
end
```

The source operand is added to the SP register and the sum is stored in the SP register. This has the effect of allocating or allocating space on the stack. Two's complement addition is performed.

Flags:

S: Unaffected
Z: Unaffected
H: Set if there is a carry from bit 11 of the result; cleared otherwise
V: Unaffected
N: Cleared
C: Set if there is a carry from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
IM:	ADD SP,nn	11101101 10000010 -n(low)- -n(high)	2	I, X

5

AND
AND (BYTE)

AND [A,]src src = R, RX, IM, IR, X

Operation: A ← A AND src

A logical AND operation is performed between the corresponding bits of the source operand and the accumulator and the result is stored in the accumulator. A 1 is stored wherever the corresponding bits in the two operands are both 1s; otherwise a 0 is stored. The contents of the source are unaffected.

Flags:
- S: Set if the most significant bit of the result is set; cleared otherwise
- Z: Set if all bits of the result are zero; cleared otherwise
- H: Set
- P: Set if the parity is even; cleared otherwise
- N: Cleared
- C: Cleared

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	AND [A,]R	10100-r-	2	
RX:	AND [A,]RX	11y11101 1010010w	2	
IM:	AND [A,]n	11100110 ——n—	2	
IR:	AND [A,](HL)	10100110	2+r	
X:	AND [A,](XY+d)	11y11101 10100110——d—	4+r	I

Field Encodings:
- r: per convention
- y: 0 for IX, 1 for IY
- w: 0 for high byte, 1 for low byte

5

BIT
BIT TEST

BIT b,dst dst = R, IR, X

Operation: Z ← NOT dst(b)

The specified bit b within the destination operand is tested, and the Zero flag is set to 1 if the specified bit is 0, otherwise the Zero flag is cleared to 0. The contents of the destination are unaffected. The bit to be tested is specified by a 3-bit field in the instruction; this field contains the binary encoding for the bit number to be tested. The bit number b must be between 0 and 7.

Flags:
S: Unaffected
Z: Set if the specified bit is zero; cleared otherwise
H: Set
V: Unaffected
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	BIT b,R	11001011 01bbb-r-	2	
IR:	BIT b,(HL)	11001011 01bbb110	2+r	
X:	BIT b,(XY+d)	11y11101 11001011 ——d— 01bbb110	4+r	I

Field Encodings: r: per convention
y: 0 for IX, 1 for IY

5

CALL
CALL

CALL [cc,]dst dst = DA

Operation:

```
if (cc is TRUE) then begin
    if (XM) then begin
        SP        ←   SP - 4
        (SP)      ←   PC(7-0)
        (SP+1)    ←   PC(15-8)
        (SP+2)    ←   PC(23-16)
        (SP+3)    ←   PC(31-24)
        PC(31-0)  ←   dst(31-0)
    else begin
        SP        ←   SP - 2
        (SP)      ←   PC(7-0)
        (SP+1)    ←   PC(15-8)
        PC(15-0)  ←   dst(15-0)
        end
    end
```

A conditional Call transfers program control to the destination address if the setting of a selected flag satisfies the condition code "cc" specified in the instruction; an Unconditional Call always transfers control to the destination address. The current contents of the Program Counter (PC) are pushed onto the top of the stack; the PC value used is the address of the first instruction byte following the Call instruction. The destination address is then loaded into the PC and points to the first instruction of the called procedure. At the end of a procedure a Return instruction (RET) can be used to return to the original program.

Each of the Zero, Carry, Sign, and Overflow Flags can be individually tested and a call performed conditionally on the setting of the flag.

The operand is not enclosed in parentheses with the CALL instruction.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
DA:	CALL CC,addr	11-cc100 -a(low)- -a(high)	note	I, X
	CALL addr	11001101 -a(low)- -a(high)	4+w	I, X

Field Encodings: cc: 000 for NZ, 001 for Z, 010 for NC, 011 for C,
100 for PO or NV, 101 for PE or V, 110 for P or NS, 111 for M or S

Note: 2 if CC is false, 4+w if CC is true

CCF
COMPLEMENT CARRY FLAG

CCF

Operation: C ← NOT C

The Carry flag is inverted.

Flags:
S: Unaffected
Z: Unaffected
H: The previous state of the Carry flag
V: Unaffected
N: Cleared
C: Set if the Carry flag was clear before the operation; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	CCF	00111111	2	

CPW
COMPARE (WORD)

CPW [HL,]src src = R, RX, IM, X

Operation: HL(15-0) – src(15-0)

The source operand is compared with the HL register and the flags are set accordingly. The contents of the HL register and the source are unaffected. Two's complement subtraction is performed.

Flags:
- S: Set if the result is negative; cleared otherwise
- Z: Set if the result is zero; cleared otherwise
- H: Set if there is a borrow from bit 12 of the result; cleared otherwise
- V: Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is of the same sign as the source; cleared otherwise
- N: Set
- C: Set if there is a borrow from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	CPW [HL,]R	11101101 101111rr	2	
RX:	CPW [HL,]RX	11y11101 10111111	2	
IM:	CPW [HL,]nn	11101101 10111110 -n(low)- n(high)-	2	
X:	CPW [HL,](XY+d)	11y11101 11111110 ——d—	4+r	

Field Encodings:
- rr: 00 for BC, 01 for DE, 11 for HL
- y: 0 for IX, 1 for IY

5

CPDR
COMPARE, DECREMENT AND REPEAT (BYTE)

CPDR

Operation:

```
Repeat until (BC=0 OR match) begin
    A - (HL)
    if (XM) then begin
        HL(31-0)    ←    HL(31-0) - 1
        end
    else begin
        HL(15-0)    ←    HL(15-0) - 1
        end
    BC(15-0)        ←    BC(15-0) - 1
end
```

This instruction is used for searching strings of byte data. The bytes of data starting at the location addressed by the HL register are compared with the contents of the accumulator until either an exact match is found or the string length is exhausted becuase the BC register has decremented to zero. The Sign and Zero flags are set to reflect the result of the comparison. The contents of the accumulator and the memory bytes are unaffected.Two's complement subtraction is performed.

After each comparison, the HL register is decremented by one, thus moving the pointer to the previous element in the string.

The BC register, used as a counter, is then decremented by one. If the result of decrementing the BC register is not zero and no match has been found, the process is repeated. If the contents of the BC register are zero at the start of this instruction, a string length of 65,536 is indicated.

This instruction can be interrupted after each execution of the basic operation. The PC value at the start of this instruction is pushed onto the stack so that the instruction can be resumed.

Flags:

S: Set if the last result is negative; cleared otherwise
Z: Set if the last result is zero, indicating a match; cleared otherwise
H: Set if there is a borrow from bit 4 of the last result; cleared otherwise
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	CPDR	11101101 10111001	(3+r)n	X

CPIR
COMPARE, INCREMENT AND REPEAT (BYTE)

CPIR

Operation:

```
Repeat until (BC=0 OR match) begin
    A - (HL)
    if (XM) then begin
        HL(31-0)      ←      HL(31-0) + 1
        end
    else begin
        HL(15-0)      ←      HL(15-0) + 1
        end
    BC(15-0)          ←      BC(15-0) - 1
end
```

This instruction is used for searching strings of byte data. The bytes of data starting at the location addressed by the HL register are compared with the contents of the accumulator until either an exact match is found or the string length is exhausted becuase the BC register has decremented to zero. The Sign and Zero flags are set to reflect the result of the comparison. The contents of the accumulator and the memory bytes are unaffected. Two's complement subtraction is performed.

After each comparison, the HL register is incremented by one, thus moving the pointer to the next element in the string. The BC register, used as a counter, is then decremented by one. If the result of decrementing the BC register is not zero and no match has been found, the process is repeated. If the contents of the BC register are zero at the start of this instruction, a string length of 65,536 is indicated.

This instruction can be interrupted after each execution of the basic operation. The PC value at the start of this instruction is pushed onto the stack so that the instruction can be resumed.

Flags:

S: Set if the last result is negative; cleared otherwise
Z: Set if the last result is zero, indicating a match; cleared otherwise
H: Set if there is a borrow from bit 4 of the last result; cleared otherwise
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	CPIR	11101101 10110001	(3+r)n	X

CPLW
COMPLEMENT HL REGISTER (WORD)

CPLW [HL]

Operation: HL(15-0) ← NOT HL(15-0)

The contents of the HL register are complemented (ones complement); all 1s are changed to 0 and vice-versa.

Flags:

S: Unaffected
Z: Unaffected
H: Set
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	CPLW [HL]	11011101 00101111	2	

5

DDIR
DECODER DIRECTIVE

DDIR mode mode = W or LW, IB or IW

Operation: None, decoder directive only

This is not an instruction, but rather a directive to the instruction decoder.

The instruction decoder may be directed to fetch an additional byte or word of immediate data or address with the instruction, as well as tagging the instruction for execution in either Word or Long Word mode. All eight combinations of the two options are supported, as shown in the encoding below. Instructions which do not support decoder directives are assembled by the instruction decoder as if the decoder directive were not present.

The IB decoder directive causes the decoder to fetch an additional byte immediately after the existing immediate data or direct address, and in front of any trailing opcode bytes (with instructions starting with DD-CB or FD-CB, for example).

Likewise, the IW decoder directive causes the decoder to fetch an additional word immediately after the existing immediate data or direct address, and in front of any trailing opcode bytes.

Byte ordering within the instruction follows the usual convention; least significant byte first, followed by more significant bytes. More-significant immediate data or direct address bytes not specified in the instruction are taken as all zeros by the processor.

The W decoder directive causes the instruction decoder to tag the instruction for execution in Word mode. This is useful while the Long Word (LW) bit in the Select Register (SR) is set, but 16-bit data manipulation is required for this instruction.

The LW decoder directive causes the instruction decoder to tag the instruction for execution in Long Word mode. This is useful while the LW bit in the SR is cleared, but 32-bit data manipulation is required for this instruction.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	DDIR mode	11w11101 110000im	0	

Field Encodings:

wim:	000 W	Word mode
	001 IB,W	Immediate byte, Word mode
	010 IW,W	Immediate word, Word mode
	011 IB	Immediate byte
	100 LW	Long Word mode
	101 IB,LW	Immediate byte, Long Word mode
	110 IW,LW	Immediate word, Long Word mode
	111 IW	Immediate word

DEC[W]
DECREMENT (WORD)

DEC[W] dst dst = R, RX

Operation:

```
if (XM) then begin
    dst(31-0)  ←   dst(31-0) - 1
    end
else begin
    dst(15-0)  ←   dst(15-0) - 1
    end
```

The destination operand is decremented by one and the result is stored in the destination. Two's complement subtraction is performed. Note that the length of the operand is controlled by the Extended/Native mode selection, which is consistent with the manipulation of an address by the instruction.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	DEC[W] R	00rr1011	2	X
RX:	DEC[W] RX	11y11101 00101011	2	X

Field Encodings: rr: 00 for BC, 01 for DE, 10 for HL, 11 for SP
y: 0 for IX, 1 for IY

DIVUW
DIVIDE UNSIGNED (WORD)

DIVUW [HL,]src src = R, RX, IM, X

Operation:
HL(15-0) ← HL / src
HL(31-16) ← remainder

The contents of the the HL register (dividend) are divided by the source operand (divisor) and the quotient is stored in the lower word of the HL register; the remainder is stored in the upper word of the HL register. The contents of the source are unaffected. Both operands are treated as unsigned, binary integers. There are three possible outcomes of the DIVUW instruction, depending on the division and the resulting quotient:

Case 1: If the quotient is less than 65536, then the quotient is left in the HL register, the Overflow and Sign flags are cleared to 0, and the Zero flag is set according to the value of the quotient.

Case 2: If the divisor is zero, the HL register is unchanged, the Zero and Overflow flags are set to 1, and the Sign flag is cleared to 0.

Case 3: If the quotient is greater than or equal to 65536, the HL register is unchanged, the Overflow flag is set to 1, and the Sign and Zero flags are cleared to 0.

Flags:
S: Cleared
Z: Set if the quotient or divisor is zero; cleared otherwise
H: Unaffected
V: Set if the divisor is zero or if the computed quotient is greater than or equal to 65536; cleared otherwise
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	DIVUW [HL,]R	11101101 11001011 101110rr	20	
RX:	DIVUW [HL,]RX	11101101 11001011 1011110y	20	
IM:	DIVUW [HL,]nn	11101101 11001011 10111111 -n(low)- -n(high)	20	
X:	DIVUW [HL,](XY+d)	11y11101 11001011 ——d— 10111010	22+r	I

Field Encodings: rr: 00 for BC, 01 for DE, 11 for HL
y: 0 for IX, 1 for IY

5

EI
ENABLE INTERRUPTS

EI [n]

Operation:

```
if (n is present) then begin
    for i=1 to 4 begin
        if (n(i) = 1) then begin
            IER(i-1)    ←    1
            end
        end
    if (n(0) = 1) then begin
        SR(5)           ←    1
        end
    end
else begin
    SR(5)               ←    1
    end
```

If an argument is present, enable the selected interrupts by setting the appropriate enable bits in the Interrupt Enable Register, and then set the Interrupt Enable Flag (IEF1) in the Select Register (SR) if the least-significant bit of the argument is set, enabling maskable interrupts. Bits 7-5 of the argument are ignored.

If no argument is present, IEF1 in the SR is set to 1, enabling maskable interrupts.

Note that during the execution of this instruction and the following instruction, maskable interrupts are not sampled.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	EI	11111011	2	
	EI n	11011101 11111011 —n——	2	

5

EX
EXCHANGE ADDRESSING REGISTER WITH TOP OF STACK

EX (SP),dst dst = HL, IX, IY

Operation:

```
if (LW) then begin
    (SP+3) ↔ dst(31-24)
    (SP+2) ↔ dst(23-16)
    end
(SP+1)     ↔ dst(15-8)
(SP)       ↔ dst(7-0)
```

The contents of the destination register are exchanged with the top of the stack. In Long Word mode this exchange is two words; otherwise it is one word.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	EX (SP),HL	11100011	3+r+w	L
	EX (SP),XY	11y11101 11100011	3+r+w	L

Field Encodings: y: 0 for IX, 1 for IY

5

EX
EXCHANGE REGISTER WITH ALTERNATE REGISTER (BYTE)

EX dst,src src = R

Operation: dst ↔ src

The contents of the destination are exchanged with the contents of the source, where the destination is a register in the primary bank and the source is the corresponding register in the alternate bank

Flags:
S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	EX R,R'	11001011 00110-r-	3	

Field Encoding: r: per convention

EX
EXCHANGE WITH ACCUMULATOR

EX A,src src = R, IR

Operation: dst ↔ src

The contents of the accumulator are exchanged with the contents of the source.

Flags:
S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	EX A,R	11101101 00-r-111	3	
IR:	EX A,(HL)	11101101 00110111	3+r+w	

Field Encodings: r: per convention

5

EXTS
EXTEND SIGN (BYTE)

EXTS [A]

Operation:

```
L                    ←    A
if (A(7)=0) then begin
    H  "   00h
    if (LW) then begin
        HL(31-16)    ←    0000h
        end
    end
else begin
    H  "   FFh
    if (LW) then begin
        HL(31-16)    ←    FFFFh
        end
    end
```

The contents of the accumulator, considered as a signed, two's complement integer, are sign-extended to 16 bits and the result is stored in the HL register. The contents of the accumulator are unaffected. This instruction is useful for conversion of short signed operands into longer signed operands.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	EXTS [A]	11101101 01100101	3	L

5

EXX
EXCHANGE REGISTERS WITH ALTERNATE BANK

EXX

Operation: SR(8) ← NOT SR(8)

Bit 8 of the Select Register (SR), which controls the selection of primary or alternate bank for the BC, DE, and HL registers, is complemented, thus effectively exchanging the BC, DE, and HL registers between the two banks.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	EXX	11011001	3	

EXXY
EXCHANGE IY REGISTER WITH ALTERNATE BANK

EXXY

Operation: SR(24) ← NOT SR(24)

Bit 24 of the Select Register (SR), which controls the selection of primary or alternate bank for the IY register, is complemented, thus effectively exchanging the IY register between the two banks.

Flags:
S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	EXXY	11111101 11011001	3	

IM
INTERRUPT MODE SELECT

IM p p = 0, 1, 2, 3

Operation: SR(4-3) ← p

The interrupt mode of operation is set to one of four modes. (See Chapter 6 for a description of the various modes for responding to interrupts). The current interrupt mode can be read from the Select Register (SR).

Flags:
S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	IM p	11101101 010pp110	4	

Field Encodings: pp: 00 for Mode 0, 01 for Mode 3, 10 for Mode 1, 11 for Mode 2

INW
INPUT (WORD)

INW dst,(C) dst = R

Operation: dst(15-0) ← (C)

The word of data from the selected peripheral is loaded into the destination register. During the I/O transaction, the contents of the 32-bit BC register are placed on the address bus.

Flags:
S: Set if the input data is negative; cleared otherwise
Z: Set if the input data is zero; cleared otherwise
H: Cleared
P: Set if the input data has even parity; cleared otherwise
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	INW R,(C)	11011101 01rrr000	2+i	

Field Encodings: rrr: 000 for BC, 010 for DE, 111 for HL

IN0
INPUT (FROM PAGE 0)

IN0 dst,(n) dst = R

Operation: dst ← (n)

The byte of data from the selected on-chip peripheral is loaded into the destination register. No external I/O transaction will be generated as a result of this instruction, although the I/O address will appear on the address bus while this internal read is occurring. The peripheral address is placed on the low byte of the address bus and zeros are placed on all other address lines. When the second opcode byte is 30h no data is stored in a destination; only the flags are updated.

Flags:
S: Set if the input data is negative; cleared otherwise
Z: Set if the input data is zero; cleared otherwise
H: Cleared
P: Set if the input data has even parity; cleared otherwise
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	IN0 R,(n)	11101101 00 -r- 000 ——n—	3+i	
none:	IN0 (n	11101101 00110000 ——n—	3+i	

Field Encodings: r: per convention

5

INAW
INPUT DIRECT FROM PORT ADDRESS (WORD)

INAW HL,(nn)

Operation: HL(15-0) ← (nn)

The word of data from the selected peripheral is loaded into the HL register. During the I/O transaction, the peripheral address from the instruction is placed on the address bus. Any bytes of address not specified in the instruction are driven on the address lines as all zeros.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	INAW HL,(nn)	11111101 11011011 -n(low)- -n(high)	3+i	I

INC[W]
INCREMENT (WORD)

INC[W] dst dst = R, RX

Operation:

```
if (XM) then begin
    dst(31-0)  <      dst(31-0) + 1
    end
else begin
    dst(15-0)  ←      dst(15-0) + 1
    end
```

The destination operand is incremented by one and the sum is stored in the destination. Two's complement addition is performed. Note that the length of the operand is controlled by the Extended/Native mode selection, which is consistent with the manipulation of an address by the instruction.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	INC[W] R	00rr0011	2	X
RX:	INC[W] RX	11y11101 00100011	2	X

Field Encodings: rr: 00 for BC, 01 for DE, 10 for HL, 11 for SP
y: 0 for IX, 1 for IY

5

INDW
INPUT AND DECREMENT (WORD)

INDW

Operation:

(HL) ← (DE)
BC(15-0) ← BC(15-0) – 1
HL ← HL – 2

This instruction is used for block input of strings of data. During the I/O transaction the 32-bit DE register is placed on the address bus.

First the word of data from the selected peripheral is loaded into the memory location addressed by the HL register. Then the BC register, used as a counter, is decremented by one. The HL register is then decremented by two, thus moving the pointer to the next destination for the input.

Flags:

S: Unaffected
Z: Set if the result of decrementing BC is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	INDW	11101101 11101010	2+i+w	

5

INDRW
INPUT, DECREMENT AND REPEAT (WORD)

INDRW

Operation:

```
repeat until (BC=0) begin
    (HL)        ←    (DE)
    BC(15-0)    ←    BC(15-0) – 1
    HL          ←    HL – 2
    end
```

This instruction is used for block input of strings of data. The string of input data from the selected peripheral is loaded into memory at consecutive addresses, starting with the location addressed by the HL register and decreasing. During the I/O transaction the 32-bit DE register is placed on the address bus.

First the BC register, used as a counter, is decremented by one. First the word of data from the selected peripheral is loaded into the memory location addressed by the HL register. Then the BC register, used as a counter, is decremented by one. The HL register is then decremented by two, thus moving the pointer to the next destination for the input. If the result of decrementing the BC register is 0, the instruction is terminated, otherwise the sequence is repeated. If the BC register contains 0 at the start of the execution of this instruction, 65536 bytes are input.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value at the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set if the result of decrementing BC is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	INDRW	11101101 11111010	n X (2+i+w)	

5

INIW
INPUT AND INCREMENT (WORD)

INIW

Operation:

(HL) ← (DE)
BC(15-0) ← BC(15-0) – 1
HL ← HL + 2

This instruction is used for block input of strings of data.
During the I/O transaction the 32-bit DE register is placed on the address bus.

First the word of data from the selected peripheral is loaded into the memory location addressed by the HL register. Then the BC register, used as a counter, is decremented by one. The HL register is then incremented by two, thus moving the pointer to the next destination for the input.

Flags:

S: Unaffected
Z: Set if the result of decrementing BC is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	INIW	11101101 11100010	2+i+w	

5

INIRW
INPUT, INCREMENT AND REPEAT (WORD)

INIRW

Operation:

```
repeat until (BC=0) begin
    (HL)        ←    (DE)
    BC(15-0)    ←    BC(15-0) – 1
    HL          ←    HL + 2
    end
```

This instruction is used for block input of strings of data. The string of input data from the selected peripheral is loaded into memory at consecutive addresses, starting with the location addressed by the HL register and increasing. During the I/O transaction the 32-bit DE register is placed on the address bus.

First the word of data from the selected peripheral is loaded into the memory location addressed by the HL register. Then the BC register, used as a counter, is decremented by one. The HL register is then incremented by two, thus moving the pointer to the next destination for the input. If the result of decrementing the BC register is 0, the instruction is terminated, otherwise the sequence is repeated. If the BC register contains 0 at the start of the execution of this instruction, 65536 bytes are input.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value at the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set if the result of decrementing BC is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	INIRW	11101101 11110010	n X (2+i+w)	

5

JR
JUMP RELATIVE

JR [cc,]dst dst = RA

Operation:

```
if (cc is TRUE) then begin
    dst ← SIGN EXTEND dst
    if (XM) then begin
        PC(31-0)    ←    PC(31-0) + dst(31-0)
        end
    else begin
        PC(15-0)    ←    PC(15-0) + dst(15-0)
        end
    end
```

A conditional Jump transfers program control to the destination address if the setting of a selected flag satisfies the condition code "cc" specified in the instruction; an unconditional Jump always transfers control to the destination address. Either the Zero or Carry flag can be tested for the conditional Jump. If the jump is taken, the Program Counter (PC) is loaded with the destination address; otherwise the instruction following the Jump Relative instruction is executed.

The destination address is calculated using relative addressing. The displacement in the instruction is added to the PC value for the instruction following the JR instruction, not the value of the PC for the JR instruction.

These instructions employ either an 8-bit, 16-bit, or 24-bit signed, two's complement displacement from the PC to permit jumps within a range of –126 to +129 bytes, –32,765 to +32,770 bytes, or –8,388,604 to +8,388,611 bytes from the location of this instruction.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
RA:	JR CC,addr	001cc000 —disp—	2	X
	JR addr	00011000 —disp—	2	X
	JR CC,addr	11011101 001cc000 -d(low)- -d(high)	2	X
	JR addr	11011101 00011000 -d(low)- -d(high)	2	X
	JR CC,addr	11111101 001cc000 -d(low)- -d(mid)- -d(high)	2	X
	JR addr	11111101 00011000 -d(low)- -d(mid)- -d(high)	2	X

Field Encodings: cc: 00 for NZ, 01 for Z, 10 for NC, 11 for C

LD
LOAD IMMEDIATE (BYTE)

LD dst,n dst = R, RX, IR, X

Operation: dst ← n

The byte of immediate data is loaded into the destination.

Flags:
S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	LD R,n	00-r-110 ——n—	2	
RX:	LD RX,n	11y11101 0010w110 ——n—	2	
IR:	LD (HL),n	00110110 ——n—	3+w	
X:	LD (XY+d),n	11y11101 00110110 ——d— ——n—	5+w	I

Field Encodings:
r: per convention
y: 0 for IX, 1 for IY
w: 0 for high byte, 1 for low byte

5

LDW
LOAD IMMEDIATE (WORD)

LDW dst,nn dst = IR

Operation:

```
if (LW) then begin
    dst(31-0)  ←  nn
    end
else begin
    dst(15-0)  ←  nn
end
```

The word of immediate data is loaded into the destination.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
IR:	LDW (IR),nn	11101101 00pp0110 -n(low)- -n(high)	3+w	I, L

Field Encodings: pp: 00 for BC, 01 for DE, 11 for HL

LD[W]
LOAD REGISTER (WORD)

LD[W] dst,src dst = R
src = R, RX, IR, DA, X, SR
or
dst = R, RX, IR, DA, X, SR
src = R

Operation:

```
if (LW) then begin
    dst(31-0)  ←  src(31-0)
    end
else begin
    dst(15-0)  ←  src(15-0)
    end
```

The contents of the source are loaded into the destination.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Load into Register

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	LD Rd,Rs	11rs1101 00rd0010	2	L
RX:	LD R,RX	11y11101 00rr1011	2	L
IR:	LD R,(IR)	11011101 00rr11ri	2+r	L
	LD RX,(IR)	11y11101 00ri0011	2+r	L
DA:	LD HL,(nn)	00101010 -n(low)- -n(high)	3+r	I, L
	LD R,(nn)	11101101 01ra1011 -n(low)- -n(high)	3+r	I, L
	LD RX,(nn)	11y11101 00101010 -n(low)- -n(high)	3+r	I, L
X:	LD R,(XY+d)	11y11101 11001011 ——d— 00rr0011	4+r	I, L
	LD IX,(IY+d)	11111101 11001011 ——d— 00100011	4+r	I, L
	LD IY,(IX+d)	11011101 11001011 ——d— 00100011	4+r	I, L
SR:	LD R,(SP+d)	11011101 11001011 ——d— 00rr0001	4+r	I, L
	LD RX,(SP+d)	11y11101 11001011 ——d— 00100001	4+r	I, L

LD
LOAD STACK POINTER

LD dst,src dst = SP
src = R, RX, IM, DA
or
dst = DA
src = SP

Operation:

```
if (LW) then begin
    dst(31-0)  ←  src(31-0)
    end
else begin
    dst(15-0)  ←  src(15-0)
    end
```

The contents of the source are loaded into the destination.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Load into Stack Pointer

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	LD SP,HL	11111001	2	L
RX:	LD SP,RX	11y11101 11111001	2	L
IM:	LD SP,nn	00110001 -n(low)- -n(high)	2	I, L
DA:	LD SP,(nn)	11101101 01111011 -n(low)- -n(high)	3+r	I, L

Field Encodings: y: 0 for IX, 1 for IY

Load from Stack Pointer

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
DA:	LD (nn),SP	11101101 01110011 -n(low)- -n(high)	4+w	I, L

5

LD
LOAD INTO I OR R REGISTER (BYTE)

LD dst,src dst = I, R
            src = A

Operation: dst ← src

The contents of the accumulator are loaded into the destination. Note that the R register does not contain the refresh address and is not modified by refresh transactions.

Flags:
S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	LD I,A	11101101 01000111	2	
	LD R,A	11101101 01001111	2	

LDCTL
LOAD CONTROL REGISTER (BYTE)

LDCTL dst,src

dst = DSR, XSR, YSR
src = A, IM
or
dst = A
src = DSR, XSR, YSR
or
dst = SR
src = A, IM

Operation:

```
if (dst = SR) then begin
    SR(31-24)  ←   src
    SR(23-16)  ←   src
    SR(15-8)   ←   src
    end
else begin
    dst        ←   src
    end
```

The contents of the source are loaded into the destination.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Load into Control Register

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	LDCTL SR,A	11011101 11001000	4	
	LDCTL Rd,A	11qq1101 11011000	4	
IM:	LDCTL SR,n	11011101 11001010 ——n—	4	
	LDCTL Rd,n	11qq1101 11011010 ——n—	4	

Field Encodings: qq: 01 for XSR, 10 for DSR, 11 for YSR

Load from Control Register

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	LDCTL A,Rs	11qq1101 11010000	2	

Field Encodings: qq: 01 for XSR, 10 for DSR, 11 for YSR

LDCTL
LOAD INTO CONTROL REGISTER (WORD)

LDCTL dst,src dst = SR
src = HL

Operation:

```
if (LW) then begin
    dst(31-16) ←  HL(31-16)
    end
else begin
    dst(31-24) ←  HL(15-8)
    dst(23-16) ←  HL(15-8)
    end
dst(15-8)      ←  HL(15-8)
dst(0)         ←  HL(0)
```

The contents of the HL register are loaded into the Select Register (SR). If Long Word mode is not in effect the upper byte of the HL register is copied into the three most significant bytes of the select register. This instruction does not modify the mode bits in the SR. There are dedicated instructions to modify the mode bits.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Load from Control Register

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	LDCTL SR,HL	11101101 11001000	4	L

5

LDDW
LOAD AND DECREMENT (WORD)

LDDW

Operation:

```
if (LW) then begin
    (DE)      ←    (HL)
    (DE+1)    ←    (HL+1)
    (DE+2)    ←    (HL+2)
    (DE+3)    ←    (HL+3)
    DE        ←    DE – 4
    HL        ←    HL – 4
    BC(15-0)  ←    BC(15-0) – 4
    end
else begin
    (DE)      ←    (HL)
    (DE+1)    ←    (HL+1)
    DE        ←    DE – 2
    HL        ←    HL – 2
    BC(15-0)  ←    BC(15-0) – 2
    end
```

This instruction is used for block transfers of words of data. The word of data at the location addressed by the HL register is loaded into the location addressed by the DE register. Both the DE and HL registers are then decremented by two or four, thus moving the pointers to the preceeding words in the array. The BC register, used as a byte counter, is then decremented by two or four.

Both DE and HL should be even, to allow word transfers on the bus. BC must be even, transferring an even number of bytes, or the operation is undefined.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	LDDW	11101101 11101000	3+r+w	L

5

LDDRW
LOAD, DECREMENT AND REPEAT (WORD)

LDDRW

Operation:

```
repeat until (BC=0) begin
    if (LW) then begin
        (DE)        ←    (HL)
        (DE+1)      ←    (HL+1)
        (DE+2)      ←    (HL+2)
        (DE+3)      ←    (HL+3)
        DE          ←    DE – 4
        HL          ←    HL – 4
        BC(15-0)    ←    BC(15-0) – 4
        end
    else begin
        (DE)        ←    (HL)
        (DE+1)      ←    (HL+1)
        DE          ←    DE – 2
        HL          ←    HL – 2
        BC(15-0)    ←    BC(15-0) – 2
        end
    end
```

This instruction is used for block transfers of strings of data. The words of data at the location addressed by the HL register are loaded into memory starting at the location addressed by the DE register. The number of words moved is determined by the contents of the BC register. If the BC register contains zero when this instruction is executed, 65,536 words are transferred. The effect of decrementing the pointers during the transfer is important if the source and destination strings overlap with the source string starting at a lower memory address. Placing the pointers at the highest address of the strings and decrementing the pointers ensures that the source string is copied without destroying the overlapping area.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted,so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
V: Cleared
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	LDDRW	11101101 11111000	nX(3+r+w)	L

LDIW
LOAD AND INCREMENT (WORD)

LDIW

Operation:

```
if (LW) then begin
    (DE)      ←    (HL)
    (DE+1)    ←    (HL+1)
    (DE+2)    ←    (HL+2)
    (DE+3)    ←    (HL+3)
    DE        ←    DE + 4
    HL        ←    HL + 4
    BC(15-0)  ←    BC(15-0) – 4
    end
else begin
    (DE)      ←    (HL)
    (DE+1)    ←    (HL+1)
    DE        ←    DE + 2
    HL        ←    HL + 2
    BC(15-0)  ←    BC(15-0) – 2
    end
```

This instruction is used for block transfers of words of data. The word of data at the location addressed by the HL register is loaded into the location addressed by the DE register. Both the DE and HL registers are then incremented by two or four, thus moving the pointers to the succeeding words in the array. The BC register, used as a byte counter, is then decremented by two or four.

Both DE and HL should be even, to allow word transfers on the bus. BC must be even, transferring an even number of bytes, or the operation is undefined.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	LDIW	11101101 11100000	3+r+w	L

5

LDIRW
LOAD, INCREMENT AND REPEAT (WORD)

LDIRW

Operation:

```
repeat until (BC=0) begin
    if (LW) then begin
        (DE)        ←    (HL)
        (DE+1)      ←    (HL+1)
        (DE+2)      ←    (HL+2)
        (DE+3)      ←    (HL+3)
        DE          ←    DE + 4
        HL          ←    HL + 4
        BC(15-0)    ←    BC(15-0) – 4
        end
    else begin
        (DE)        ←    (HL)
        (DE+1)      ←    (HL+1)
        DE          ←    DE + 2
        HL          ←    HL + 2
        BC(15-0)    ←    BC(15-0) – 2
        end
    end
```

This instruction is used for block transfers of strings of data. The words of data at the location addressed by the HL register are loaded into memory starting at the location addressed by the DE register. The number of words moved is determined by the contents of the BC register. If the BC register contains zero when this instruction is executed, 65,536 words are transferred. The effect of incrementing the pointers during the transfer is important if the source and destination strings overlap with the source string starting at a higher memory address. Placing the pointers at the lowest address of the strings and incrementing the pointers ensures that the source string is copied without destroying the overlapping area.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is save before the interrupt request is accepted,so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
V: Cleared
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	LDIRW	11101101 11110000	(3+r+w)n	L

MTEST
MODE TEST

MTEST

Operation:

S ← SR(7)
Z ← SR(6)
C ← SR(1)

The three mode control bits in the Select Register (SR) are transferred to the flags. This allows the program to determine the state of the machine.

Flags:

S: Set if Extended mode is in effect; cleared otherwise
Z: Set if Long word mode is in effect; cleared otherwise
H: Unaffected
V: Unaffected
N: Unaffected
C: Set if Lock mode is in effect; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	MTEST	11011101 11001111	2	

MULTUW
MULTIPLY UNSIGNED (WORD)

MULTUW [HL,]src src = R, RX, IM, X

Operation: HL(31-0) ← HL(15-0) x src(15-0)

The contents of the HL register are multiplied by the source operand and the product is stored in the HL register. The contents of the source are unaffected. Both operands are treated as unsigned, binary integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to indicate that the upper word of the HL register is required to represent the result; if the Carry flag is cleared, the product can be correctly represented in 16 bits and the upper word of the HL register merely holds zero.

Flags:
S: Cleared
Z: Set if the result is zero; cleared otherwise
H: Unaffected
V: Cleared
N: Unaffected
C: Set if the product is greater than or equal to 65536; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	MULTUW [HL,]R	11101101 11001011 100110rr	11	
RX:	MULTUW [HL,]RX	11101101 11001011 1001110y	11	
IM:	MULTUW [HL,]nn	11101101 11001011 10011111 -n(low)- -n(high)	11	
X:	MULTUW [HL,](XY+d)	11y11101 11001011 ——d— 10011010	13+r	I

Field Encodings: rr: 00 for BC, 01 for DE, 11 for HL
y: 0 for IX, 1 for IY

5

NEGW
NEGATE HL REGISTER (WORD)

NEGW [HL]

Operation: HL(15-0) ← -HL(15-0)

The contents of the HL register are negated, that is replaced by its two's complement value. Note that 8000h is, replaced by itself, because in two's complement representation the negative number with the greatest magnitude has no positive counterpart; for this case, the Overflow flag is set to 1.

Flags:

S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Set if there is a borrow from bit 4 of the result; cleared otherwise
V: Set if the content of the HL register was 8000h before the operation; cleared otherwise
N: Set
C: Set if the content of the HL register was not 0000h before the operation; cleared if the content of the HL register was 0000h

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	NEGW [HL]	11101101 01010100	2	

OR
OR (BYTE)

OR [A,]src src = R, RX, IM, IR, X

Operation: A ← A OR src

A logical OR operation is performed between the corresponding bits of the source operand and the accumulator and the result is stored in the accumulator. A 1 bit is stored wherever either of the corresponding bits in the two operands is 1; otherwise a 0 bit is stored. The contents of the source are unaffected.

Flags:
- S: Set if the most significant bit of the result is set; cleared otherwise
- Z: Set if all bits of the result are zero; cleared otherwise
- H: Cleared
- P: Set if the parity is even; cleared otherwise
- N: Cleared
- C: Cleared

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	OR [A,]R	10110-r-	2	
RX:	OR [A,]RX	11y11101 1011010w	2	
IM:	OR [A,]n	11110110 ——n—	2	
IR:	OR [A,](HL)	10110110	2+r	
X:	OR [A,](XY+d)	11y11101 10110110 ——d—	4+r	I

Field Encodings:
- r: per convention
- y: 0 for IX, 1 for IY
- w: 0 for high byte, 1 for low byte

5

OTDM
OUTPUT DECREMENT MEMORY

OTDM

Operation:

(C) ← (HL)
C ← C – 1
B ← B – 1
HL ← HL – 1

This instruction is used for block output of strings of data to on-chip peripherals. No external I/O transaction will be generated as a result of this instruction, although the I/O address will appear on the address bus and the write data will appear on the data bus while this internal write is occurring. The peripheral address is placed on the low byte of the address bus and zeros are placed on all other address lines. The byte of data from the memory location addressed by the HL register is loaded to the on-chip I/O port addressed by the C register. The C register, holding the port address, is decremented by one to select the next output port. The B register, used as a counter, is then decremented by one. The HL register is then decremented by one, thus moving the pointer to the next source for the output.

Flags:

S: Set if the result of decrementing B is negative; cleared otherwise
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Set if there is a borrow from bit 4 during the decrement of the B register; cleared otherwise
P: Set if the result of the decrement of the B register is even; cleared otherwise
N: Set if the most significant bit of the byte transferred was a 1; cleared otherwsie
C: Set if there is a borrow from the most significant bit during the decrement of the B register; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OTDM	11101101 10001011	2+r+o	

5

OTDR
OUTPUT, DECREMENT AND REPEAT (BYTE)

OTDR

Operation:

```
repeat until (B=0) begin
    B  ← B – 1
    (C) ← (HL)
    HL ← HL – 1
    end
```

This instruction is used for block output of strings of data. The string of output data is loaded into the selected peripheral from memory at consecutive addresses, starting with the location addressed by the HL register and decreasing. During the I/O transaction the 32-bit BC register is placed on the address bus. Note that the B register contains the loop count for this instruction so that A(15-8) are not useable as part of a fixed port address. The decremented B register is used in the address.

First the B register, used as a counter, is decremented by one. The byte of data from the memory location addressed by the HL register is loaded into the selected peripheral. The HL register is then decremented by one, thus moving the pointer to the next source for the output. If the result of decrementing the B register is 0, the instruction is terminated, otherwise the sequence is repeated. If the B register contains 0 at the start of the execution of this instruction, 256 bytes are output.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value at the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OTDR	11101101 10111011	2+r+o	

5

OTIM
OUTPUT INCREMENT MEMORY

OTIM

Operation:

(C) ← (HL)
C ← C + 1
B ← B − 1
HL ← HL + 1

This instruction is used for block output of strings of data to on-chip peripherals. No external I/O transaction will be generated as a result of this instruction, although the I/O address will appear on the address bus and the write data will appear on the data bus while this internal write is occurring. The peripheral address is placed on the low byte of the address bus and zeros are placed on all other address lines. The byte of data from the memory location addressed by the HL register is loaded to the on-chip I/O port addressed by the C register. The C register, holding the port address, is incremented by one to select the next output port. The B register, used as a counter, is then decremented by one. The HL register is then incremented by one, thus moving the pointer to the next source for the output.

Flags:

S: Set if the result of decrementing B is negative; cleared otherwise
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Set if there is a borrow from bit 4 during the decrement of the B register; cleared otherwise
P: Set if the result of the decrement of the B register is even; cleared otherwise
N: Set if the most significant bit of the byte transferred was a 1; cleared otherwise
C: Set if there is a borrow from the most significant bit during the decrement of the B register; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OTIM	11101101 10000011	2+r+o	

5

OTIR
OUTPUT, INCREMENT AND REPEAT (BYTE)

OTIR

Operation:

repeat until (B=0) begin
B ← B – 1
(C) ← (HL)
HL ← HL + 1
end

This instruction is used for block output of strings of data. The string of output data is loaded into the selected peripheral from memory at consecutive addresses, starting with the location addressed by the HL register and increasing. During the I/O transaction the 32-bit BC register is placed on the address bus. Note that the B register contains the loop count for this instruction so that A(15-8) are not useable as part of a fixed port address. The decremented B register is used in the address.

First the B register, used as a counter, is decremented by one. The byte of data from the memory location addressed by the HL register is loaded into the selected peripheral. The HL register is then incremented by one, thus moving the pointer to the next source for the output. If the result of decrementing the B register is 0, the instruction is terminated, otherwise the sequence is repeated. If the B register contains 0 at the start of the execution of this instruction, 256 bytes are output.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value at the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OTIR	11101101 10110011	2+r+o	

OUT
OUTPUT (BYTE)

OUT (C),src src = R, IM

Operation: (C) ← src

The byte of data from the source is loaded into the selected peripheral. During the I/O transaction, the contents of the 32-bit BC register are placed on the address bus.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	OUT (C),R	11101101 01 -r- 001	3+o	
IM:	OUT (C),n	11101101 01110001 —n—	3+o	

Field Encodings: r: per convention

OUT
OUTPUT ACCUMULATOR

OUT (n),A

Operation: (n) ← A

The byte of data from the accumulator is loaded into the selected peripheral. During the I/O transaction, the 8-bit peripheral address from the instruction is placed on the low byte of the address bus, the contents of the accumulator are placed on address lines A(15-8), and the high-order address lines are all zeros.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OUT (n),A	11010011 ——n—	3+o	

OUTA
OUTPUT DIRECT TO PORT ADDRESS (BYTE)

OUT (nn),A

Operation: (nn) ← A

The byte of data from the accumulator is loaded into the selected peripheral. During the I/O transaction, the peripheral address from the instruction is placed on the address bus. Any bytes of address not specified in the instruction are driven on the address lines are all zeros.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OUTA (nn),A	11101101 11010011 -n(low)- -n(high)	2+o	I

OUTD
OUTPUT AND DECREMENT (BYTE)

OUTD

Operation:

B ← B - 1
(C) ← (HL)
HL ← HL - 1

This instruction is used for block output of strings of data. During the I/O transaction the 32-bit BC register is placed on the address bus. Note that the B register contains the loop count for this instruction so that A15-A8 are not useable as part of a fixed port address. The decremented B register is used in the address.

First the B register, used as a counter, is decremented by one. The byte of data from the memory location addressed by the HL register is loaded into the selected peripheral. The HL register is then decremented by one, thus moving the pointer to the next source for the output.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OUTD	11101101 10101011	2+r+o	

5

OUTI
OUTPUT AND INCREMENT (BYTE)

OUTI

Operation:

B ← B - 1
(C) ← (HL)
HL ← HL + 1

This instruction is used for block output of strings of data. During the I/O transaction the 32-bit BC register is placed on the address bus. Note that the B register contains the loop count for this instruction so that A15-A8 are not useable as part of a fixed port address. The decremented B register is used in the address.

First the B register, used as a counter, is decremented by one. The byte of data from the memory location addressed by the HL register is loaded into the selected peripheral. The HL register is then incremented by one, thus moving the pointer to the next source for the output.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	OUTI	11101101 10100011	2+r+o	

5

POP
POP ACCUMULATOR

POP dst dst = AF

Operation:

```
F    ← (SP)
A    ← (SP+1)
SP   ← SP + 2
if (LW) then begin
    SP ← SP + 2
    end
```

The contents of the memory location addressed by the Stack Pointer (SP) are loaded into the destination in ascending byte order from ascending address memory locations. For this instruction, the Flag register is the least significant byte, followed by the Accumulator. The SP is then incremented by two (by four in the Long Word mode). Note that in the Long Word mode only one word is read from memory, although the SP is in fact incremented by four.

Flags:

S: Loaded from (SP)
Z: Loaded from (SP)
H: Loaded from (SP)
V: Loaded from (SP)
N: Loaded from (SP)
C: Loaded from (SP)

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	POP AF	11110001	2+r	L

5

POP
POP REGISTER

POP dst dst = R, RX

Operation:

```
if (LW) then begin
    dst(7-0 )   ←   (SP)
    dst(15-8)   ←   (SP+1)
    dst(23-16)  ←   (SP+2)
    dst(31-24)  ←   (SP+3)
    SP          ←   SP + 4
    end
else begin
    dst(7-0)    ←   (SP)
    dst(15-8)   ←   (SP+1)
    SP          ←   SP + 2
    end
```

The contents of the memory location addressed by the Stack Pointer (SP) are loaded into the destination in ascending byte order from ascending address memory locations. The SP is then incremented by two (by four in the Long Word mode).

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	POP R	11rr 0001	1+r	L
RX:	POP RX	11y11101 11100001	1+r	L

Field Encodings: rr: 00 for BC, 01 for DE, 10 for HL
y: 0 for IX, 1 for IY

5

PUSH
PUSH CONTROL REGISTER

PUSH src src = SR

Operation:

```
if (LW) then begin
    SP      ← SP - 4
    (SP)    ← src(7-0)
    (SP+1)  ← src(15-8)
    (SP+2)  ← src(23-16)
    (SP+3)  ← src(31-24)
    end
else begin
    SP      ← SP - 2
    (SP)    ← src(7-0)
    (SP+1)  ← src(15-8)
    end
```

The Stack Pointer (SP) is decremented by two (by four in Long Word mode) and the source is loaded into the memory locations addressed by the SP in ascending byte order in ascending address memory locations. The contents of the source are unaffected.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	PUSH SR	11101101 11000101	3+w	L

PUSH
PUSH REGISTER

PUSH src src = R, RX

Operation:

```
if (LW) then begin
    SP      ← SP - 4
    (SP)    ← src(7-0)
    (SP+1)  ← src(15-8)
    (SP+2)  ← src(23-16)
    (SP+3)  ← src(31-24)
    end
else begin
    SP      ← SP - 2
    (SP)    ← src(7-0)
    (SP+1)  ← src(15-8)
    end
```

The Stack Pointer (SP) is decremented by two (by four in Long Word mode) and the source is loaded into the memory locations addressed by the SP in ascending byte order in ascending address memory locations. The contents of the source are unaffected.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	PUSH R	11rr0101	3+w	L
RX:	PUSH RX	11y11101 11100101	3+w	L

Field Encodings: rr: 00 for BC, 01 for DE, 10 for HL
y: 0 for IX, 1 for IY

RESC
RESET CONTROL BIT

RESC mode mode = LCK, LW

Operation:

```
if (mode = LCK) then begin
    SR(1) ← 0
    end
else begin
    SR(6) ← 0
    end
```

When reseting Lock mode (LCK), the LCK bit (bit 1) in the Select Register (SR) is set to 0, enabling external bus requests. Note that these requests cannot be granted until after the instruction has been executed, and that one or more of the succeeding instructions may also have been fetched for decoding before this instruction has been executed.

When reseting Long Word mode (LW), the LW bit (bit 6) in the SR is set to 0, selecting 16-bit words. When using 16-bit words, all word load operations transfer 16 bits.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	RESC mode	11mm1101 11111111	4	

Field Encodings: mm: 01 for LW, 10 for LCK

RETB
RETURN FROM BREAKPOINT

Operation: PC (31-0) ← SPC (31-0)

This instruction is used to return to a previously executing procedure at the end of a breakpoint. The contents of the Shadow Program Counter (SPC), which holds the address of the next instruction of the previously executing procedure, are loaded into the Program Counter (PC).

Note that maskable interrupts (if IEF1 is set) and non-maskable interrupt are enabled after the instruction following RETB is executed.

Flags:
S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	RETB	11101101 01010101	2	

RETN
RETURN FROM NONMASKABLE INTERRUPT

RETN

Operation:

```
if (XM) then begin
    PC(7-0)    ←    (SP)
    PC(15-8)   ←    (SP+1)
    PC(23-16)  ←    (SP+2)
    PC(31-24)  ←    (SP+3)
    SP         ←    SP + 4
    end
else begin
    PC(7-0)    ←    (SP)
    PC(15-8)   ←    (SP+1)
    SP         ←    SP + 2
    end
IEF1           ←    IEF2
```

This instruction is used to return to a previously executing procedure at the end of a procedure entered by a nonmaskable interrupt. The contents of the location addressed by the Stack Pointer (SP) are popped into the Program Counter (PC), thereby specifying the location of the next instruction to be executed. The previous setting of the interrupt enable bit is restored by execution of this instruction.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	RETN	11101101 01000101	2+r	X

RLW
ROTATE LEFT (WORD)

RLW dst dst = R, RX, IR, X

Operation:

tmp ← dst
dst(0) ← C
C ← dst(15)
dst(n+1) ← tmp(n) for n = 0 to 14

The contents of the destination operand are concatenated with the Carry flag and together they are rotated left one bit position. The most significant bit of the destination operand is moved to the Carry flag and the Carry flag is moved to bit 0 of the destination.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from the most significant bit was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	RLW R	11101101 11001011 000100rr	2	
RX:	RLW RX	11101101 11001011 0001010y	2	
IR:	RLW (HL)	11101101 11001011 00010010	2+r	
X:	RLW (XY+d)	11y11101 11001011 ——d— 00010010	4+r	I

Field Encodings: rr: 00 for BC, 01 for DE, 11 for HL
y: 0 for IX, 1 for IY

5

RLC
ROTATE LEFT CIRCULAR (BYTE)

RLC dst dst = R, IR, X

Operation:

tmp ← dst
C ← dst(7)
dst(0) ← tmp(7)
dst(n+1) ← tmp(n) for n = 0 to 6

The contents of the destination operand are rotated left one bit position. Bit 7 of the destination operand is moved to the bit 0 position and also replaces the Carry flag.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from bit 7 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	RLC R	11001011 00000-r-	2	
IR:	RLC (HL)	11001011 00000110	2+r	
X:	RLC (XY+d)	11y11101 11001011 ——d— 00000110	4+r	I

Field Encodings: r: per convention
y: 0 for IX, 1 for IY

5

RLCA
ROTATE LEFT CIRCULAR (ACCUMULATOR)

RLCA

Operation:

tmp ← A
C ← A(7)
A(0) ← tmp(7)
A(n+1) ← tmp(n) for n = 0 to 6

The contents of the accumulator are rotated left one bit position. Bit 7 of the accumulator is moved to the bit 0 position and also replaces the Carry flag.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
P: Unaffected
N: Cleared
C: Set if the bit rotated from bit 7 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	RLCA	00000111	2	

RR
ROTATE RIGHT (BYTE)

RR dst dst = R, IR, X

Operation:

tmp ← dst
dst(7) ← C
C ← dst(0)
dst(n) ← tmp(n+1) for n = 0 to 6

The contents of the destination operand are concatenated with the Carry flag and together they are rotated right one bit position. Bit 0 of the destination operand is moved to the Carry flag and the Carry flag is moved to bit 7 of the destination.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from bit 0 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	RR R	11001011 00011-r-	2	
IR:	RR (HL)	11001011 00011110	2+r	
X:	RR (XY+d)	11y11101 11001011 ——d— 00011110	4+r	I

Field Encodings: r: per convention
y: 0 for IX, 1 for IY

5

RRA
ROTATE RIGHT (ACCUMULATOR)

RRA

Operation:

tmp ← A
A(7) ← C
C ← A(0)
A(n) ← tmp(n+1) for n = 0 to 6

The contents of the accumulator are concatenated with the Carry flag and together they are rotated right one bit position. Bit 0 of the accumulator is moved to the Carry flag and the Carry flag is moved to bit 7 of the accumulator.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
P: Unaffected
N: Cleared
C: Set if the bit rotated from bit 0 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	RRA	00011111	2	

RRCW
ROTATE RIGHT CIRCULAR (WORD)

RRCW dst dst = R, RX, IR, X

Operation:

tmp ← dst
C ← dst(0)
dst(15) ← tmp(0)
dst(n) ← tmp(n+1) for n = 0 to 14

The contents of the destination operand are rotated right one bit position. Bit 0 of the destination operand is moved to the most significant bit position and also replaces the Carry flag.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from bit 0 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	RRCW R	11101101 11001011 000010rr	2	
RX:	RRCW RX	11101101 11001011 0000110y	2	
IR:	RRCW (HL)	11101101 11001011 00001010	2+r	
X:	RRCW (XY+d)	11y11101 11001011 ——d— 00001010	4+r	I

Field Encodings: rr: 00 for BC, 01 for DE, 11 for HL
y: 0 for IX, 1 for IY

5

RRD
ROTATE RIGHT DIGIT

RRD

Operation:

tmp(3-0) ← A(3-0)
A(3-0) ← dst(3-0)
dst(3-0) ← dst(7-4)
dst(7-4) ← tmp(3-0)

The low digit of the accumulator is logically concatenated to the destination byte whose memory address is in the HL register. The resulting three-digit quantity is rotated to the right by one BCD digit (four bits). The upper digit of the source is moved to the lower digit of the source; the lower digit of the source is moved to the lower digit of the accumulator, and the lower digit of the accumulator is moved to the upper digit of the source. The upper digit of the accumulator is unaffected. In multiple-digit BCD arithmetic, this instruction can be used to shift to the right a string of BCD digits, thus dividing it by a power of ten. The accumulator serves to transfer digits between successive bytes of the string. This is analogous to the use of the Carry flag in multiple-precision shifting using the RR instruction.

Flags:

S: Set if the accumulator is negative after the operation; cleared otherwise
Z: Set if the accumulator is zero after the operation; cleared otherwise
H: Cleared
P: Set if the parity of the accumulator is even after the operation; cleared otherwise
N: Cleared
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	RRD	11101101 01100111	3+r	

5

SBC
SUBTRACT WITH CARRY (BYTE)

SBC A,src src = R, RX, IM, IR, X

Operation: A ← A - src - C

The source operand together with the Carry flag is subtracted from the accumulator and the difference is stored in the accumulator. The contents of the source are unaffected. Two's complement subtraction is performed.

Flags:
- S: Set if the result is negative; cleared otherwise
- Z: Set if the result is zero; cleared otherwise
- H: Set if there is a borrow from bit 4 of the result; cleared otherwise
- V: Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is of the same sign as the source; cleared otherwise
- N: Set
- C: Set if there is a borrow from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	SBC A,R	10011-r-	2	
RX:	SBC A,RX	11y11101 1001110w	2	
IM:	SBC A,n	11011110 ——n—	2	
IR:	SBC A,(HL)	10011110	2+r	
X:	SBC A,(XY+d)	11y11101 10011110 ——d—	4+r	I

Field Encodings:
- r: per convention
- y: 0 for IX, 1 for IY
- w: 0 for high byte, 1 for low byte

5

SBCW
SUBTRACT WITH CARRY (WORD)

SBCW [HL,]src src = R, RX, IM, X

Operation: HL(15-0) ← HL(15-0) - src(15-0) - C

The source operand together with the Carry flag is subtracted from the HL register and the difference is stored in the HL register. The contents of the source are unaffected. Two's complement subtraction is performed.

Flags:
- S: Set if the result is negative; cleared otherwise
- Z: Set if the result is zero; cleared otherwise
- H: Set if there is a borrow from bit 12 of the result; cleared otherwise
- V: Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is of the same sign as the source; cleared otherwise
- N: Set
- C: Set if there is a borrow from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	SBCW [HL,]R	11101101 100111rr	2	
RX:	SBCW [HL,]RX	11y11101 10011111	2	
IM:	SBCW [HL,]nn	11101101 10011110 -n(low) -n(high)-	2	
X:	SBCW [HL,](XY+d)	11y11101 11011110 ——d—	4+r	I

Field Encodings:
- rr: 00 for BC, 01 for DE, 11 for HL
- y: 0 for IX, 1 for IY

5

SET
SET BIT

SET b, dst dst = R, IR, X

Operation: dst(b) ← 1

The specified bit b within the destination operand is set to 1. The other bits in the destination are unaffected. The bit to be set is specified by a 3-bit field in the instruction; this field contains the binary encoding for the bit number to be set. The bit number b must be between 0 and 7.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	SET b,R	11001011 11bbb -r-	2	
IR:	SET b,(HL)	11001011 11bbb110	2+r	
X:	SET b,(XY+d)	11y11101 11001011 ——d— 11bbb110	4+r	I

Field Encodings:

r: per convention
y: 0 for IX, 1 for IY

5

SLA
SHIFT LEFT ARITHMETIC (BYTE)

SLA dst dst = R, IR, X

Operation:

tmp ← dst
C ← dst(7)
dst(0) ← 0
dst(n+1) ← tmp(n) for n = 0 to 6

The contents of the destination operand are shifted left one bit position. Bit 7 of the destination operand is moved to the Carry flag and zero is shifted into bit 0 of the destination.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit shifted from bit 7 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	SLA R	11001011 00100-r-	2	
IR:	SLA (HL)	11001011 00100110	2+r	
X:	SLA (XY+d)	11y11101 11001011 ——d— 00100110	4+r	I

Field Encodings: r: per convention
y: 0 for IX, 1 for IY

5

SLP
SLEEP

SLP

Operation:

```
if (STBY not enabled) then
    CPU Halts
else
    Z380 enters Standby mode
```

With Standby mode disabled, this instruction is interpreted and executed as a HALT instruction.

With Standby mode enabled, executing this instruction causes all device operation to stop, thus minimizing power dissipation. The /STNBY signal is asserted to indicate this Standby mode status. /STNBY remains asserted until an interrupt or reset request is accepted, which causes the device to exit Standby mode. If the option is enabled, an external bus request also causes the devcie to exit the Standby mode.

Flags:

S: Unaffected
Z: Unaffected
H: Unaffected
V: Unaffected
N: Unaffected
C: Unaffected

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
	SLP	11101101 01110110	2	

SRAW
SHIFT RIGHT ARITHMETIC (WORD)

SRAW dst dst = R, RX, IR, X

Operation:

tmp ← dst
C ← dst(0)
dst(15) ← tmp(15)
dst(n) ← tmp(n+1) for n = 0 to 14

The contents of the destination operand are shifted right one bit position. Bit 0 of the destination operand is moved to the Carry flag and the most significant bit remains unchanged.

Flags:

S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit shifted from bit 0 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	SRAW R	11101101 11001011 001010rr	2	
RX:	SRAW RX	11101101 11001011 0010110y	2	
IR:	SRAW (HL)	11101101 11001011 00101010	2+r	
X:	SRAW (XY+d)	11y11101 11001011 ——d— 00101010	4+r	I

Field Encodings: rr: 00 for BC, 01 for DE, 11 for HL
y: 0 for IX, 1 for IY

5

SRLW
SHIFT RIGHT LOGICAL (WORD)

SRLW dst dst = R, RX, IR, X

Operation:

tmp ← dst
C ← dst(0)
dst(15) ← 0
dst(n) ← tmp(n+1) for n = 0 to 14

The contents of the destination operand are shifted right one bit position. Bit 0 of the destination operand is moved to the Carry flag and zero is shifted into the most significant bit of the destination.

Flags:

S: Cleared
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit shifted from bit 0 was a 1; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	SRLW R	11101101 11001011 001110rr	2	
RX:	SRLW RX	11101101 11001011 0011110y	2	
IR:	SRLW (HL)	11101101 11001011 00111010	2+r	
X:	SRLW (XY+d)	11y11101 11001011 ——d— 00111010	4+r	I

Field Encodings: rr: 00 for BC, 01 for DE, 11 for HL
y: 0 for IX, 1 for IY

5

SUB
SUBTRACT (WORD)

SUB HL,src src = DA

Operation:

```
if (XM) then begin
    HL(31-0)  ←  HL(31-0) - src(31-0)
    end
else begin
    HL(15-0)  ←  HL(15-0) - src(15-0)
    end
```

The source operand is subtracted from the HL register and the difference is stored in the HL register. The contents of the source are unaffected. Two's complement subtraction is performed. Note that the length of the operand is controlled by the Extended/Native mode selection, which is consistent with the manipulation of an address by the instruction.

Flags:

S: Unaffected
Z: Unaffected
H: Set if there is a borrow from bit 12 of the result; cleared otherwise
V: Unaffected
N: Set
C: Set if there is a borrow from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
DA:	SUB HL,(nn)	11101101 11010110 -n(low)- -n(high)	2+r	I, X

5

SUBW
SUBTRACT (WORD)

SUBW [HL,]src src = R, RX, IM, X

Operation: HL(15-0) ← HL(15-0) - src(15-0)

The source operand is subtracted from the HL register and the difference is stored in the HL register. The contents of the source are unaffected. Two's complement subtraction is performed.

Flags:
- S: Set if the result is negative; cleared otherwise
- Z: Set if the result is zero; cleared otherwise
- H: Set if there is a borrow from bit 12 of the result; cleared otherwise
- V: Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is of the same sign as the source; cleared otherwise
- N: Set
- C: Set if there is a borrow from the most significant bit of the result; cleared otherwise

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	SUBW [HL,]R	11101101 100101rr	2	
RX:	SUBW [HL,]RX	11y11101 10010111	2	
IM:	SUBW [HL,]nn	11101101 10010110 -n(low)- n(high)-	2	
X:	SUBW [HL,](XY+d)	11y11101 11010110 ——d—	2+r	I

Field Encodings:
- rr: 00 for BC, 01 for DE, 11 for HL
- y: 0 for IX, 1 for IY

5

TST
TEST (BYTE)

TST src src = R, IM, IR

Operation: A AND src

A logical AND operation is performed between the corresponding bits of the source operand and the accumulator. The contents of both the accumulator and the source are unaffected; only the flags are modified as a result of this instruction.

Flags:
S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if all bits of the result are zero; cleared otherwise
H: Set
P: Set if the parity is even; cleared otherwise
N: Cleared
C: Cleared

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	TST R	11101101 00-r-100	2	
IM:	TST n	11101101 01100100 ——n—	2	
IR:	TST (HL)	11101101 00110100	2+r	

Field Encodings: r: per convention

XOR
EXCLUSIVE OR (BYTE)

XOR [A,]src src = R, RX, IM, IR, X

Operation: A ← A XOR src

A logical EXCLUSIVE OR operation is performed between the corresponding bits of the source operand and the accumulator and the result is stored in the accumulator. A 1 bit is stored wherever the corresponding bits in the two operands are different; otherwise a 0 bit is stored. The contents of the source are unaffected.

Flags:
- S: Set if the most significant bit of the result is set; cleared otherwise
- Z: Set if all bits of the result are zero; cleared otherwise
- H: Cleared
- P: Set if the parity is even; cleared otherwise
- N: Cleared
- C: Cleared

Addressing Mode	Syntax	Instruction Format	Execute Time	Note
R:	XOR [A,]R	10101-r-	2	
RX:	XOR [A,]RX	11y11101 1010110w	2	
IM:	XOR [A,]n	11101110 ——n—	2	
IR:	XOR [A,](HL)	10101110	2+r	
X:	XOR [A,](XY+d)	11y11101 10101110 ——d—	4+r	I

Field Encodings:
- r: per convention
- y: 0 for IX, 1 for IY
- w: 0 for high byte, 1 for low byte

5

CHAPTER 6
INTERRUPTS AND TRAPS

6.1 INTRODUCTION

Exceptions are conditions that can alter the normal flow of program execution. The Z380™ CPU supports three kinds of exceptions; interrupts, traps, and resets.

Interrupts are asynchronous events generated by a device external to the CPU; peripheral devices use interrupts to request service from the CPU. Traps are synchronous events generated internally in the CPU by a particular condition that can occur during the attempted execution of an instruction—in particular, when executing undefined instructions. Thus, the difference between Traps and Interrupts is their origin. A Trap condition is always reproducible by re-executing the program that created the Trap, whereas an Interrupt is generally independent of the currently executing task.

A hardware reset overrides all other conditions, including Interrupts and Traps. It occurs when the /RESET line is activated and causes certain CPU control registers to be initialized. Resets are discussed in detail in Chapter 7.

The Z380 MPU's Interrupt and Trap structure provides compatibility with the existing Z80 and Z180 MPU's with the following exception—the undefined opcode Trap occurrence is with respect to the Z380 instruction set, and its response is improved (vs the Z180) to make Trap handling easier. The Z380 MPU also offers additional features to enhance flexibility in system design.

6.2 INTERRUPTS

Of the five external Interrupt inputs provided, one is assigned as a Nonmaskable Interrupt, /NMI. The remaining inputs, /INT3-/INT0, are four asynchronous maskable Interrupt requests.

The Nonmaskable Interrupt; (NMI) is an Interrupt that cannot be disabled (masked) by software. Typically NMI is reserved for high priority external events that need immediate attention, such as an imminent power failure. Maskable Interrupts are Interrupts that can be disabled (masked) through software by cleaning the appropriate bits in the Interrupt Enable Register (IER) and IEF1 bit in the Select Register (SR).

All of these four maskable Interrupt inputs (/INT3-/INT0) are external input signals to the Z380 CPU core. The four Interrupt enable bits in the Interrupt Enable Register determine (IER; Internal I/O address: 17H) which of the requested Interrupts are accepted. Each Interrupt input has a fixed priority, with /INT0 as the highest and /INT3 as the lowest.

The Enable Interrupt (EI) instruction is used to selectively enable the maskable Interrupts (by setting the appropriate bits in the IER register and IEF1 bit in the SR register) and the Disable Interrupt instruction is used to selectively disable interrupts (by clearing appropriate bits in the IER, and/or clearing IEF1 bit in the SR register). When an Interrupt source has been disabled, the CPU ignores any request from that source. Because maskable Interrupt requests are not retained by the CPU, the request signal on a maskable Interrupt line must be asserted until the CPU acknowledges the request.

When enabling Interrupts with the EI instruction, all maskable Interrupts are automatically disabled (whether previously enabled or not) for the duration of the execution of the EI instruction and the instruction immediately following.

Interrupts are always accepted between instructions. The block move, block search, and block I/O instructions can be interrupted after any iteration.

The Z380 CPU has four selectable modes for handling externally generated Interrupts, using the IM instruction. The first three modes extend the Z80 CPU Interrupt Modes to accommodate the Z380 CPU's additional Interrupt inputs in a compatible fashion. The fourth mode allows more flexibility in interrupt handling.

6.2.2.1 IEF1, IEF2

IEF1 controls the overall enabling and disabling of all on-chip peripheral and external maskable Interrupt requests. If IEF1 is at logic 0, all such Interrupts are disabled. The purpose of IEF2 is to correctly manage the occurrence of /NMI. When /NMI is acknowledged, the state of IEF1 is copied to IEF2 and then IEF1 is cleared to logic 0. At the end of the /NMI interrupt service routine, execution of the Return From Nonmaskable Interrupt instruction, RETN, automatically copies the state of IEF2 back to IEF1. This is a means to restore the Interrupt enable condition existing before the occurrence of /NMI. Table 6-3 summarizes the states of IEF1 and IEF2 resulting from various operations.

Table 6-3. Operation Effects on IEF1 and IEF2

Operation	IEF1	IEF2	Comments
/RESET	0	0	Inhibits all interrupts except Trap and /NMI.
Trap	0	0	Disables interrupt nesting.
/NMI	0	IEF1	IEF1 value copied to IEF2, then IEF1 is cleared.
RETN	IEF2	NC	Returns from /NMI service routine.
/INT3-/INT0	0	0	Disables interrupt nesting.
RETI	NC	NC	Returns from Interrupt service routine, Z80 I/O device.
RET	NC	NC	Returns from service routine, or returns from Interrupt service routine for a non-Z80 I/O device.
EI	1	1	
DI	0	0	
LD A,I or LD R,I	NC	NC	IEF2 value is copied to P/V Flag.
LD HL,I or LD HL,R	NC	NC	

(NC = No Change)

6.2.2.2 I, I Extend

The 8-bit Interrupt Register and the 16-bit Interrupt Register Extension are cleared during reset.

6.2.2.3 Interrupt Enable Register

D7-D4 Reserved Read as 0, should write to as 0.
D3-D0 IE3-IE0 (Interrupt Request Enable Flags)

These flags individually indicate if /INT3, /INT2, /INT1, or /INT0 is enabled. Note that these flags are conditioned with the Enable and Disable Interrupt instructions (with arguments) (See Figure 6.1).

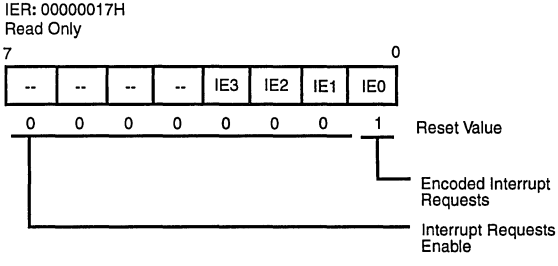


Figure 6-1. Interrupt Enable Register

6.2.2.4 Assigned Vectors Base Register

D7-D1 AB15-AB9 (Assigned Vectors Base). The Interrupt Register Extension, Iz, together with AB15-AB9, define the base address of the assigned Interrupt vectors table in memory space (See Figure 6-2).

D0 Reserved. Read as 0, should write to as 0.

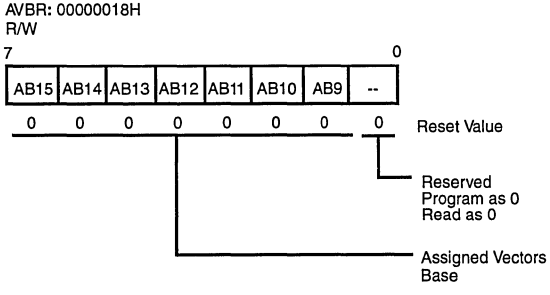


Figure 6-2. Assigned Vectors Base Register

6.4 NONMASKABLE INTERRUPT

The Nonmaskable Interrupt Input /NMI is edge sensitive, with the Z380 MPU internally latching the occurrence of its falling edge. When the latched version of /NMI is recognized, the following operations are performed.

1. The Interrupted PC (Program Counter) value is pushed onto the stack. The size of the PC value pushed onto the stack depends on Native (one word) or Extended mode (two words) in effect.
2. The state of IEF1 is copied to IEF2, then IEF1 is cleared.
3. The Z380 MPU commences to fetch and execute instructions from address 00000066H.

6.5 INTERRUPT RESPONSE FOR MASKABLE INTERRUPT ON /INT0

The transactions caused by the Maskable Interrupt on /INT0 are different depends on the Interrupt Mode in effect at the time when the interrupt has been accepted, as described below.

6.5.1 Interrupt Mode 0 Response for Maskable Interrupt /INT0

This mode is similar to the 8080 CPU Interrupt response mode. During the Interrupt acknowledge transaction, the external I/O device being acknowledged is expected to output a vector onto the upper portion of the data bus, D15-D8. The Z380 MPU interprets the vector as an instruction opcode. IEF1 and IEF2 are reset to logic 0, disabling all further maskable interrupt requests. Note that unlike the other interrupt responses, the PC is not automatically pushed onto the stack. Typically, a Restart instruction (RST) is used, since the Restart opcode is only one byte long, meaning that the interrupting peripheral needs to supply only one byte of information. For this case, it pushes the interrupted PC (Program Counter) value onto the stack and resumes execution at a fixed memory location. Alternatively, a 3-byte call to any location can be executed.

Note that a Trap occurs if an undefined opcode is supplied by the I/O device as a vector.

6.5.2 Interrupt Mode 1 Response for Maskable Interrupt /INT0

In Interrupt Mode 1, the Z380 CPU automatically executes a Restart to a fixed location (00000038H) when an interrupt occurs. An Interrupt acknowledge transaction is generated, during which the data bus contents are ignored by the Z380 MPU. The interrupted PC value is pushed onto the stack. The size of the PC value pushed onto the stack is depends on Native (one word) or Extended mode (two words) in effect. The IEF1 and IEF2 are reset to logic 0 so as to disable further maskable interrupt requests. Instruction fetching and execution restarts at memory location 00000038H.

6.5.3 Interrupt Mode 2 Response for Maskable Interrupt /INT0

Interrupt Mode 2 is a vectored Interrupt response mode, wherein the interrupting device identifies the starting location of service routine using an 8-bit vector read by the CPU during the Interrupt acknowledge cycle.

During the Interrupt acknowledge transaction, the external I/O device being acknowledged is expected to output a vector onto the upper portion of the data bus, D15-D8. The interrupted PC value is pushed onto the stack and IEF1 and IEF2 are reset to logic 0 so as to disable further maskable interrupt requests. The size of the PC value pushed onto the stack is depends on Native (one word) or Extended mode (two words) in effect. The Z380 MPU then reads an entry from a table residing in memory and loads it into the PC to resume execution. The address of the table entry is composed of the I Extend (Iz) contents as A31-A16, the I Register contents as A15-A8 and the vector supplied by the I/O device as A7-A0. Note that the table entry is effectively the starting address of the interrupt service routine designed for the I/O device being acknowledged, and the table composing of starting addresses for all the Interrupt Mode 2 service routines can be referred to as the Interrupt Mode 2 vector table. Each table entry should be word-sized if the Z380 MPU is in the Native mode and Long Word-sized if in the Extended mode, in either case even-aligned (least significant byte with address A0 = 0), meaning 128 different vectors can be used in the Native mode, and 64 different vectors can be used in Extended mode.

6.5.4 Interrupt Mode 3 Response for Maskable Interrupt /INT0

Interrupt Mode 3 is similar to mode 2 except that a 16-bit vector is expected to be placed on the data bus D15-D0 by the I/O device during the Interrupt acknowledge transaction. The interrupted PC is pushed onto the stack. The size of the PC value pushed onto the stack depends on the

CHAPTER 7
RESET

7.1 INTRODUCTION

The Z380 CPU is placed in a dormant state when the /RESET input is asserted. All its operations are terminated, including any interrupt, bus request, or bus transaction that may be in progress. On the Z380 MPU, the IOCLK goes Low on the next BUSCLK rising edge and enters into the BUSCLK divided-by-eight mode. The address and data buses are tri-stated, and the bus control signals are driven to their inactive states. The effect of /RESET on the Z380 CPU and related internal I/O registers is depicted in Table 7-1.

The /RESET input may be asynchronous to BUSCLK, though it is sampled internally at BUSCLK's falling edges. For proper initialization of the Z380 CPU, V_{DD} must be within operating specifications and the CLK input must be stable for more than five cycles with /RESET held Low.

The Z380 CPU proceeds to fetch the first instruction 3.5 BUSCLK cycles after /RESET is deasserted, provided such deassertion meets the proper setup and hold times with reference to the falling edge of BUSCLK. On the Z380 MPU implementation, with the proper setup and hold times being met, IOCLK's first rising edge is 11.5 BUSCLK cycles after the /RESET deassertion, preceded by a minimum of four BUSCLK cycles when IOCLK is at Low.

Note that if /BREQ is active when /RESET is deasserted, the Z380 MPU would relinquish the bus instead of fetching its first instruction. IOCLK synchronization would still take place as described before.

Requirements to reset the device, and the initial state after reset might be different depending on the particular implementation of the Z380 CPU on the individual Superintegration version of the device. For /RESET effects and requirements, refer to the individual product specification.

ZiLOG

Z380™ BENCHMARKING

This application note compares the performance and program memory requirements among the new 16-bit CPU from Zilog Z80380 and several competing processors, including the Intel 80186, 80960 and Motorola 68020 and CPU32.

INTRODUCTION

Zilog's new Z380™ Central Processing Unit is a high performance CPU engine designed to meet today's application requirements. The Z380 CPU incorporates advanced architectural features that allow fast and efficient throughput and increased memory addressing capability while maintaining Z80®/Z180® object code compatibility.

The Z380 CPU is an enhanced version of the Z80 CPU. The Z80 instruction set has been retained, adding a full compliment of 16-bit arithmetic and logical operations, multiply and divide, a complete set of register-to-register loads and exchanges, plus 32-bit load and exchange, and 32-bit arithmetic operations for address calculations.

The addressing modes of the Z80 have been enhanced with Stack pointer relative loads and stores, 16-bit and 24-bit indexed offsets, and more flexible indirect register addressing. All of the addressing modes allow access to the entire 32-bit addressing space.

The register set of the Z80 microprocessor is expanded to 32 bits, and has been replicated four times to allow for fast context switching among tasks in a dedicated control environment.

The following are the key features of the Z380:

- Full static CMOS design with low power standby mode support
- 32-bit internal data paths and ALU
- 16-bit (64K) or 32-bit (4G) linear addressing space
- 16-bit internal data bus
- Two clock cycle minimum instruction execution
- Two clock cycle Memory bus
- Programmable I/O bus protocols and clock rates
- Four banks of 32-bit registers
- Enhanced interrupt capabilities, including 16-bit vectors and four external Interrupt inputs
- Undefined opcode trap for full Z380 CPU instruction set

The Z380 block diagram is shown in Figure 1. For a detailed description of the Z380 please refer to the Z380 Technical Manual, DC #8297-00, and the Z380 Preliminary Product Specification DC #6003-02 from Zilog.

8

BENCHMARKS AMONG EMBEDDED PROCESSORS

In response to a recent microprocessor selection process by a major customer, Zilog's Datacom Marketing group compared the performance and program memory requirements among the new Z80380 and several competing processors, including the Intel 80186 and 80960 and the Motorola 68020 and CPU32. (The CPU32 is the heart of the Motorola's 803xx series of integrated products.)

METHOD

Benchmarking consisted of selecting four code fragments judged to be typical of embedded applications, coding the four fragments in assembly language for each of the four processors, and calculating the execution time for each fragment on each processor, at 16, 25, and 40 MHz clock rates as applicable to each.

The results were then tabulated in a spreadsheet that first normalized them to the figure for the 25 MHz 80380, and then averaged the normalized values for memory code size and execution time, as well as an overall "figure of merit".

The code fragments were called "I/O Loop", "Signed Byte Handling". "Multiply/Accumulate". and "Interrupt". Since the execution time for I/O Loop is a function of the number of times through the loop, and because it was felt to be the most typical of user requirements, it was counted twice toward the composite performance and merit figures, once for a single iteration and once for eight times through the loop. Finally, a fifth performance category was included, the time required for memory-to-memory block movement of data. This made six performance values that were averaged with four program-size values for the overall Figure of Merit, an imbalance that "felt right" in terms of the way we think many users view the value of an embedded microprocessor.

ASSUMPTIONS

Because execution time can be a complex matter for today's pipelined processors, our benchmarks made several assumptions that simplified performance evaluation. The most presumptive was that the memory on all processors was fast enough that there would be NO WAIT STATES. (In many cases this would mandate fast Static RAM rather than larger, more economical Dynamic RAM, which makes sense for some applications but not others.)

A second assumption was that all operands were ALIGNED to the natural boundaries for their size: data accessed 16 bits at a time was located at an address that was a multiple of two bytes, while data to be accessed 32 bits at a time was located at an address that's a multiple of 4 bytes. This characteristic can be guaranteed by many high-level-language compilers, and is questionable only for the Block Move operations.

For processors that include a cache (the 68020 and 80960), the timing was calculated such that the first access to each instruction was a cache miss, and any subsequent accesses were cache hits. In other words, we assumed that these code fragments were not part of a central loop, but were executed in response to specific events that were sufficiently infrequency that the code was superseded in the cache between events.

DESCRIPTION OF THE CODE FRAGMENTS

I/O Loop

This code fragment reads received data, two bytes at a time, from a 16C30 Universal Serial Controller (USC), and stores the data in a memory buffer for each frame. The USC is the successor to Zilog's popular SCC, and has a 32-byte FIFO capacity. First, each sequence sets up whatever registers are needed to access the USC, the memory buffer, and a current pointer into the buffer named "rxi".

At the start of each loop, the code reads the number of bytes currently in the receive FIFO, from the MSbyte of a USC register called RICR. It also reads a 16-bit status register called RCSR.

IF there are no bytes left in the FIFO, the code exits from the fragment. If there is one byte in the RxFIFO, the code checks the status to see if the byte is either the last one of a frame, or is the byte at which a Receive Overrun condition occurred. If neither of these is the case, the code leaves the byte in the RxFIFO for the future, and exits from the fragment. Otherwise, or if there are two or more bytes in the FIFO, the code:

1. ensures that no interrupt can occur between the following steps,
2. reads two bytes from the FIFO via the USC register called RDR
   (the USC will only provide one if there's only one in the FIFO)
3. stores the data in memory at the address in the pointer "rxi",
4. increments rxi by 2,
5. stores rxi back in memory, and
6. enables interrupts to occur again.

After these operations the code tests the status obtained earlier from RCSR, and if the data just stored didn't represent the end of a frame, it goes back to the start of the loop described above. The following code calls an end-of-frame-handling subroutine called "_Handle_RxStatus"_this part of the fragment counts toward the code memory required but not toward the execution time, because a frame ends only once in many executions of the loop.

Signed Byte Handling

This code fragment originally came from a customer code in the hard disk field. It examines three 8-bit variables in memory called NORM, Q, and K2. Actually NORM can range from -256 to +255 and is implemented as a 16-bit variable. It computes an eight-bit result in any of six ways, depending on the sign of NORM and how it compares to that of Q, as described in the comments at the top of each page of code.

First the code may access some or all or the three input variables and/or set up registers to point to one or more of them. Then it tests the sign of NORM, branching to the second "half" of the code fragment if it's positive. In each "half", the code compares NORM and Q and branches around in a tree-structured fashion to compute the result dictated by relative values of NORM and Q.

To evaluate the overall execution time of the fragment, we computed the execution time for each of the six result cases, and averaged them.

This may be the least clear code fragment as to its cosmic purpose, but it is a reasonable example of the kind of decision-tree processing that's typical of many I/O handling and control systems.

Multiply/Accumulate

This code fragments is also taken from a customer code in a hard-disk application. It uses four 16-bit input values in memory, CURSEC, POSN_ERR, S_GRAT, and K_GRAT, plus two memory tables of 16-bit values called S_TABLE and C_TABLE, each as large as the largest possible values of CURSEC. From these the code extracts S_TABLE (CURSEC) and saves the result in a memory variable S_VALUE, and similarly extracts C_TABLE (CUSEC) and saves it in K_VALUE. The code also multiplies each value by POSN_ERR, scales/divides each result by 64, and adds the results into memory variables S_ACCUM and K_ACCUM respectively, Finally it calculates R_CP= (S_VALUE*S_GRAT + K_VALUE*KGRAT) /32.

This code includes four 16x16 multiplications and 32-bit scale/shift operations. For all the processors except the CPU32, the fragment is coded to loop back once to minimize memory requirements, by taking advantage of the similarity of the computations for the "S" and "K" values.

Interrupt

These code fragments service a "receive status" interrupt from a Zilog 16C30 Universal Serial Controller (USC). The actual code size and execution time are reduced from a full-blown ISR, by evaluating for the case of a "Receive Overrun" event, and by isolating the details of handling an End of Frame event in a separate subroutine. This is done

SUMMARY (Continued)

Of course there's something a little out of line about including the 80960KA in this comparison, which
* costs far more than any of the other processors,
* entails added system-level expense because of its 32-bit data path and required memory width (also true of the 68020), and
* requires special "block transfer" memory design techniques

In fact, Intel has another member of its 80960 that is more like the other processors herein, the 8096KA. This device has a 16-bit data bus like the 80380 and 80186, and a more compact package that lowers its cost into a more competitive range. Unfortunately we were unable to obtain any timing information for the 80960SA in the time frame required for this benchmarking.

However, we did find an Intel brochure that allows the 80960SA to participate in these results in a small way. It showed a "Dhrystone" (fixed point) figure for the 80960SA of 12145, compared to 19740 for the 960KA. Multiplying the performance figures for the 960KA by 19740/12145 (smaller is better in our figures while larger is better for the Dhrystone) yielded the results shown in the third-last and last lines. For the last line that combines code size and execution time into a final figure of Merit, only the execution time values were scaled by Intel's Dhrystone results.

To wrap up, considering both code density and execution time for these code fragments, the new Zilog 80380 blows away other 16-bit processors including the 80960SA, and comes out about equal to the much more expensive 32-bit 80960KA if skewed by one speed grade (25 MHz 380 vs. 16 MHz 960, 40 MHz 380 vs. 25 MHz 960).

I/O LOOP: 680X0

```
        ; the following 680x0 code reads data from a USC
        ; this code is not warranted to be correct nor operative, and is
        ;  intended for performance benchmarking purposes only
        ; this version assumes that rxi variable is in first 64 Kbytes

Bytes Clks (CPU32)
—     —
4     8     MOVE.L      rxi,A1                  ; address in rcv area
6     10    MOVE.L      #uscBase+ICR,A2         ; address of ICR in USC
      RxPoll16U_lp:
6     11    CMP.B #1,RICR-ICR(A2)               ; <> 1 byte in RxFIFO?
4     7     MOVE.W      RCSR-ICR(A2),D0         ; get status
2     4/10  BHI     RxPoll16U_hav               ; around if > 1 byte
2     4/10  BLO     RxPoll16U_end               ; nothing to do if < 1 byte
4     5     AND.B #$12,D0                       ; 1 byte: RxBound or overrun?
2     4/10  BZ      RxPoll16U_end               ; ignore 1 byte if neither
      RxPoll16U_hav:
4     9     BCLR  #7,(A2)                       ; disable interrupts
4     8     MOVE.W      RDR-ICR(A2),(A1)+       ; 2 serial bytes to Rx area
4     8     MOVE.L      A1,rxi                  ; store rx pointer
4     9     BSET  #7,(A2)                       ; re-enable ints
4     5     AND.B #$10,D0                       ; RxBound?
2     4/10  BZ      RxPoll16U_lp                ; loop if not
2           MOVEQ       #$C0,D0
4           AND.B CCR+1-ICR(A2),D0              ; RSBs in use?
2           BNZ   RxPoll16U_rsb                 ; around if so
4           MOVE.W      RCSR-ICR(A2),D0         ; take status from RCSR if not
2           BRA   RxPoll116U_call
      RxPoll16U_rsb:
4           MOVE.W      RDR-ICR(A2),D0          ; take status from RDR
      RxPoll16U_call:
4           BCLR  #1,D0
2           MOVE.W      D0,-(SP)
4           BCLR  #7,(A2)                       ; disable interrupts
4           JSR   _Handle_RxStatus              ; call the RxBound subroutine
2           ADDQ  #2,SP
4           BSET  #7,(A2)                       ; enable interrupts
2           BRA   RxPoll16U_lp                  ; and loop
      RxPoll16U_end:

—     —
92    50+77*N clocks (CPU32)
      56+48*N clocks (68020)
```

I/O LOOP: 68020

Bytes	Source	BC	CC	WC	1st	subs	
4	"MOVE.L rxi,A1"	3	6	8	6.5	4.5	
6	"MOVE.L #uscBase+ICR,A2"	0	5	6	3.5	2.5	
	subtotal: start						31
6	"lp: CMP.B #1,RICR-ICR(A2)"	3	7	10	8	5	
4	"MOVE.W RCSR-ISR(A2),D0"	3	7	9	7	5	
2	BHI hav (taken)	3	6	9	7.5	4.5	
	BHI hav (not taken)	1	4	5	3.5	2.5	
2	BLO end (taken)	3	6	9	7.5	4.5	
	BLO end (not taken)	1	4	5	3.5	2.5	
4	"AND.B #$12,D0"	0	4	6	4	2	
2	BZ end (taken)	3	6	9	7.5	4.5	
	subtotal: exit						24.8
	BZ end (not taken)	1	4	5	3.5	2.5	
4	"hav: BCLR #7,(A2)"	7	8	9	8.5	7.5	
4	"MOVE.W RDR-ICR(A2),(A1)+"	6	8	11	10	7	
4	"MOVE.L A1,rxi"	5	6	9	8.5	5.5	
4	"BSET #7,(A2)"	7	8	9	8.5	7.5	
4	"AND.B #$10,D0"	0	4	6	4	2	
2	BZ lp (taken)	3	6	9	7.5	4.5	
	subtotal: loop						48.5
	BZ lp (not taken)	1	4	5	3.5	2.5	
2	"MOVEQ #$C0,D0"						
4	"AND.B CCR+1-ICR(A2),D0"						
2	BNZ rsb						
4	"MOVE.W RCSR-ISR(A2),D0"						
2	BRA call						
4	"rsb: MOVE.W RDR-ICR(A2),D0"						
4	"BCLR #1,D0"						
2	"MOVE.W D0,-(SP)"						
4	"BCLR #7,(A2)"						
4	JSR _Handle_RxStatus						
2	"ADDQ #2,SP"						
4	"BSET #7,(A2)"						
2	BRA lp						
——	end:						
92	total						56+48N

I/O LOOP: 80960KA

```
# the following 80960KA code reads data from a Zilog 16C30 USC
# this code is not warranted to be correct nor operative, and is
#  intended for performance benchmarking purposes only
# this code assumes the rxi variable is in the first 4K bytes

Bytes
-----
 8          lda     uscBase+ICR,r3                  # address in USC
 4          ld      rxi,r4                          # buffer address from variable
 4          ldob    (r3),r7                         # get lsbyte of ICR
 4          clrbit  7,r7,r8
      RxPoll16U_lp:
 4          ldob    RICR+1-ICR(r3),r5 # get hi byte of RICR
 4          ldos    RCSR-ICR(r3),r6                 # get status
 4          cmpo    1,r5
 4          bg      RxPoll16U_hav                   # around if more than 1 byte
 4          bl      RxPoll16U_end                   # nothing to do if no bytes
 4          and     0x12,r6,r7                      # 1 byte: EOF or overrun?
 4          cmpobe          0,r7,RxPoll16U_end      # ignore 1 byte if not
      RxPoll16U_hav:
 4          stob    r8,(r3)                         # clear MIE, disable ints
 4          ldos    RDR-ICR(r3),r9                  # get 16 bits from USC
 4          stos    r9,(r4)                         # store in memory
 4          addo    2,r4                            # increment address
 4          st      r4,rxi                          # save address
 4          stob    r7,(r3)                         # set MIE, enable ints
 4          bbc     4,r6,RxPoll16U_lp               # loop if not EOF
 4          ldob    CCR+1-ICR(r3),r9                # get CCR hi byte
 4          bbs     7,r9,RxPoll16U_rsb              # RSB's in use?
 4          bbs     6,r9,RxPoll16U_rsb              # around if so
 4          ldos    RCSR-ICR(r3),g0                 # take status from RCSR if not
 4          b       RxPoll116U_call
      RxPoll16U_rsb:
 4          ldos    RDR-ICR(r3),g0                  # take status from RDR if so
      RxPoll16U_call:
 4          clrbit  2,g0                            # hide the overrun bit
 4          stob    r8,(r3)                         # clear MIE, disable ints
 4          bal     _Handle_RxStatus                # call the RxBound subroutine
 4          stob    r7,(r3)                         # set MIE, enable ints
 4          b       RxPoll16U_lp                    # and loop
      RxPoll16U_end:
-- --
120  55 + 24N  (see spreadsheet)
```

I/O Loop: 80960KA

					X						CF							
											D	CF						
											EA	D	CF	CF				
											Aon	EA	D	D				
											DonB		EA	X				
											W							
												AonB			CF			
												DonB			D			
												X						
													AonB			CF		
													DonB			D		
													W					
															AonB		CF	
															DonB		D	
24 clocks per repeat															W		X	
				CF												AonB		
				D	CF											DonB		
				EA	D	CF										W		
				Aon	EA	D	CF											
				DonB			D	CF										
					Aon	X		D										
					DonB		X											
								X										
9 clocks to get out																		F

SIGNED BYTE HANDLING: 680X0

```
; the following 680x0 code handles signed bytes.
; there are 3 signed byte variables in memory, Q, K2, and NORM.
;  Actually NORM can range from -256 to +255, so we test the
;  MSbyte of a 16-bit NORM but use only the LSbyte otherwise.
;  The result is as follows
; if      NORM < 0 then
;         if NORM > -Q then result := NORM
;         else if NORM > Q then result := -2*K2-NORM
;         else result := Q - K2
; else if NORM <= Q then result := NORM
;         else if NORM <= -Q then result := 2*K2-NORM
;         else result := K2 - Q
; Routines can leave the result wherever is most convenient.
; this code is not warranted to be correct nor operative, and
;  is intended for performance benchmarking purposes only.
; this code assumes that all variables are in the first 64K
;  bytes of memory

Bytes Clks (CPU32)
—     —
4     7          MOVE.B   Q,D0          ; get variable
4     7          MOVE.W   NORM,D1       ; get variable
2     4/10       BPL.S    npos          ; around if positive
2     2          NEG.B    D0            ; -Q
2     2          CMP.B    D1,D0         ; -Q-NORM
2     4/10       BMI.S    rnorm         ; go if -Q-NORM<0, NORM>-Q
2     2          NEG.B    D0            ; Q
2     2          CMP.B    D1,D0         ; Q-NORM
2     4/10       BMI.S    m2k2          ; go if Q-NORM<0, NORM>Q
4     7          SUB.B    K2,D0         ; Q - K2
2     10         BRA.S    next
4     7 m2k2:    MOVE.B   K2,D0         ; K2
2     2          NEG.B    D0            ; -K2
2     10         BRA.S    dmn
2     2 rnorm:   MOVE.B   D1,D0         ; NORM
2     10         BRA.S    next
2     2 npos:    CMP.B    D1,D0         ; Q-NORM
2     4/10       BPL      rnorm         ; go if Q-NORM>=0, NORM<=Q
2     2          NEG.B    D0            ; -Q
2     2          CMP.B    D1,D0         ; -Q-NORM
2     4/10       BPL.S    p2k2          ; go if -Q-NORM>=0, NORM<=-Q
4     7          ADD.B    K2,D0         ; K2 - Q
2     10         BRA.S    next
4     7 p2k2:    MOVE.B   K2,D0         ; K2
2     2 dmn:     ADD.B    D0,D0         ; +- 2K2
2     2          SUB.B    D1,D0         ; +- 2K2 - NORM
      next:
—     —
64    CPU32 68020
      48    NORM (pos)     40
      44    NORM (neg)     38
      55    2*K2-NORM      48
      63    -2*K2-NORM     56
      55    K2-Q           48
      51    Q-K2           46
      52.67 average        45.92
```

8

SIGNED BYTE HANDLING: 68020

Bytes	Clks	Source	BC	CC	WC
4	6.5	"move.b Q,D0"	3	6	8
4	6.5	"move.w NORM,D1"	3	6	8
2	7.5	bpl.s npos (taken)	3	6	9
	3.5	bpl.s npos (not taken)	1	4	5
2	2	neg.b D0	0	2	3
2	2	"cmp.b D1,D0"	0	2	3
2	7.5	bmi.s rnorm (taken)	3	6	9
	3.5	bmi.s rnorm (not taken)	1	4	5
2	2	neg.b D0	0	2	3
2	2	"cmp.b D1,D0"	0	2	3
2	7.5	bmi.s m2k2 (taken)	3	6	9
	3.5	bmi.s m2k2 (not taken)	1	4	5
4	7.5	"sub.b k2,d0"	3	6	9
2	7.5	bra.s next	3	6	9
4	6.5	"m2k2: move.b K2,d0"	3	6	8
2	2	neg.b D0	0	2	3
2	7.5	bra.s dmn	3	6	9
2	2	"rnorm: move.b D1,D0"	0	2	3
2	7.5	bra.s next	3	6	9
2	2	"npos: cmp.b D1,D0"	0	2	3
2	7.5	bpl rnorm (taken)	3	6	9
	3.5	bpl rnorm (not taken)	1	4	5
2	2	neg.b D0	0	2	3
2	2	"cmp.b D1,D0"	0	2	3
2	7.5	bpl.s p2k2 (taken)	3	6	9
	3.5	bpl.s p2k2 (not taken)	1	4	5
4	7.5	"add.b K2,D0"	3	6	9
2	7.5	bra.s next	3	6	9
4	6.5	"p2k2: move.b k2,d0"	3	6	8
2	2	"dmn: add.b d0,d0"	0	2	3
2	2	"sub.b d1,d0"	0	2	3
——	——	next:			
64	39.5	NORM (pos)			
	37.5	NORM (neg)			
	48	2*K2-NORM			
	55.5	-2*K2-NORM			
	48.5	K2-Q			
	46.5	Q-K2			
	45.92	average			

8

SIGNED BYTE HANDLING: 80960KA

```
# the following 80960KA code handles signed bytes.
# there are 3 signed byte variables in memory, Q, K2, and NORM.
#  Actually NORM can range from -256 to +255, so we test the
#  MSbyte of a 16-bit NORM but use only the LSbyte otherwise.
#  The result is as follows
# if      NORM < 0 then
#         if NORM > -Q then result := NORM
#         else if NORM > Q then result := -2*K2-NORM
#         else result := Q - K2
# else if NORM <= Q then result := NORM
#         else if NORM <= -Q then result := 2*K2-NORM
#         else result := K2 - Q
# Routines can leave the result wherever is most convenient.
# this code is not warranted to be correct nor operative, and is
#  intended for performance benchmarking purposes only
```

Bytes	ID			
—	—			
8	B	ldib	Q,r4	# get variable
8	C	ldis	NORM,r3	# get variable
8	D	ldib	K2,r5	# get variable
4	E	subi	r4,0,r6	# make -Q
4	F	bbc	8,r3,npos	# around if NORM non-negative
4	G	cmpibgt	r3,r6,next	# result=NORM if NORM>-Q
4	H	cmpibgt	r3,r4,m2k2	# go if NORM>Q
4	I	subi	r5,r4,r3	# result = Q - K2
4	J	b	next	
4	K m2k2:	sub	r5,0,r5	# -K2
4	L p2k2:	add	r5,r5,r5	
4	M	sub	r3,r5,r3	
4	N	b	next	
4	O npos:	cmpible	r3,r4,next	# result=NORM if NORM<=Q
4	P	cmpible	r3,r6,p2k2	# go if NORM<=-Q
4	Q	subi	r4,r5,r3	# result = K2-Q
	next:			

—	—		
76	26	NORM (pos)	see attached chart
	26	NORM (neg)	
	36	2*K2-NORM	
	37	-2*K2-NORM	
	29	K2-Q	
	30	Q-K2	
	30.67	average	

8

Signed Byte Handling: 80960KA

30							X	X	end case Q-K2							
19		begin case NORM(pos)											F			
20													F			
21													F			
22													D	F		
23													X	D	F	
24													X		D	F
25													X			
26		end case NORM(pos)											X			
19		begin case 2*K2-NORM											F			
20													F			
21													F			
22													D	F		
23													X	D	F	
24													X		D	F
25													X			
26														X		
27														X		
28														X		
29														X		
30										F						
31										F						
32										F						
33										D	F					
34										X	D	F				
35										X		D				
36		end case 2*k2-NORM									X	X				
19		begin case K2-Q											F			
20													F			
21													F			
22													D	F		
23													X	D	F	
24													X		D	
25													X			
26														X		
27														X		
28														X		
29		end case K2-Q													X	

8

```
2    36        IMUL    AX,BP                          ; times VALUE
1    3         INC     BX
1    3         INC     BX
3    3=150     CMP     BL,K_VALUE MOD 256

2    4/13      JNE     kdone                          ; around if K group done

2    2         MOV     CX,DX                          ; save MS16 of product
2    2         MOV     DI,AX                          ; save LS16 of product
4    9         MOV     AX,C_TABLE[SI]                 ; get K_VALUE from table
2    14=27     JMP     lp                             ; go back and do K group

2    3 kdone:  ADD     AX,DI                          ; add S_VALUE*S_GRAT to K...
2    3         ADC     DX,CX
3    10        SHR     AX,5
3    8         SHL     DX,3
2    3         OR      AH,DL                          ; divide by 32
3    9=36      MOV     WORD PTR R_CP,AX
—    —
72   404 (24+150+4+27+150+13+36)
```

MULTIPLY/ACCUMULATE: CPU32

Bytes	Clks	Source	Hop	Top	Cop	Hea1	Tea1	Cea1
4	7	"MOVE.W CURSEC,D0"	0	0	2	1	3	5
6	10	"MOVE.W S_TABLE(D0.W*2),D1"	0	0	2	2	2	8
4	5	"LEA S_VALUE,A0"	0	0	2	1	1	3
2	5	"MOVE.W D1,(A0)+"	1	1	5	0	0	0
2	2	"MOVE.W D1,D2"	0	0	2	0	0	0
4	31	"MULS.W POSN_ERR,D1"	0	0	26	1	3	5
2	6	"ASR.L #6,D1"	4	0	6	0	0	0
2	7	"ADD.W D1,(A0)+"	0	3	5	1	1	3
2	29	"MULS.W (A0)+,D2"	0	0	26	1	1	3
6	10	"MOVE.W C_TABLE(D0.W*2),D1"	0	0	2	2	2	8
2	5	"MOVE.W D1,(A0)+"	1	1	5	0	0	0
2	2	"MOVE.L D1,D0"	0	0	2	0	0	0
4	31	"MULS.W POSN_ERR,D1"	0	0	26	1	3	5
2	6	"ASR.L #6,D1"	4	0	6	0	0	0
2	7	"ADD.W D1,(A0)+"	0	3	5	1	1	3
2	29	"MULS.W (A0)+,D0"	0	0	26	1	1	3
2	2	"ADD.L D2,D0"	0	0	2	0	0	0
2	6	"ASR.L #5,D0"	4	0	6	0	0	0
2	4	"MOVE.W D0,(A0)+"	1	1	5	0	0	0
54	204							

MULTIPLY/ACCUMULATE: 80380

```
; this 80380 code performs a 16-bit multiply/accumulate:
; several 16-bit variables pre-exist in memory, including
;  CURSEC, POSN_ERR, S_GRAT, and K_GRAT.  In addition,
;  two tables S_TABLE and C_TABLE are of a size equal to
;  the possible range of values of CURSEC.  16-bit results
;  of this calculation in memory include S_VALUE, K_VALUE,
;  R_CP, and two accumulators S_ACCUM and K_ACCUM:

;  S_VALUE := S_TABLE(CURSEC)
;  K_VALUE := C_TABLE(CURSEC)
;  S_ACCUM := S_ACCUM + ((S_VALUE*POSN_ERR)/64)
;  K_ACCUM := K_ACCUM + ((K_VALUE*POSN_ERR)/64)
;  R_CP := (S_VALUE*S_GRAT + K_VALUE*K_GRAT) / 32

; to optimize memory accessing, all routines may assume
;  that variables S_VALUE, S_GRAT, S_ACCUM, K_VALUE, K_GRAT,
;  K_ACCUM are consecutive in memory in whatever order is
;  optimal for their instruction set, while CURSEC. POSN_ERR,
;  S_TABLE, and C_TABLE are at unrelated locations.  R_CP
;  can be in either place.

; the order in this version in S_VALUE, S_ACCUM, S_GRAT, K_VALUE,
;  K_ACCUM, K_GRAT, R_CP.

; this code is not warranted to be correct nor operative, and is
;  intended for performance benchmarking purposes only
; this code assumes that the global LW and XM bits are cleared.
```

Bytes	Clks			
4	6	LD	IX,(CURSEC)	
2	2	ADD	IX,IX	
2	2	DDIR	IB	
5	8	LD	HL,(IX+S_TABLE)	; get S_VALUE from table
2	2	DDIR	IB	
5	8	LD	IY,(IX+C_TABLE)	; get K_VALUE from table
3	2=35	LD	DE,S_VALUE	; start pointer into variables
2	3 lp:	LD	(DE),HL	; save VALUE in memory
2	2	LD	IX,HL	; save in reg
4	6	LD	BC,(POSN_ERR)	
3	10	MULTW	HL,BC	; VALUE * POSN_ERR (16x16=32)
2	2	DDIR	LW	
1	2	ADD	HL,HL	
2	2	DDIR	LW	
1	2	ADD	HL,HL	
1	2	LD	A,H	
2	2	SWAP	HL	
1	2	LD	H,L	
1	2	LD	L,A	; 16 bit product/64
1	2	INC	DE	
1	2	INC	DE	
2	6	LD	BC,(DE)	; get accum
1	2	ADD	HL,BC	; add
2	3	LD	(DE),HL	; save accum

MULTIPLY/ACCUMULATE: 80960KA

```
# this 80960 code performs a 16-bit multiply/accumulate:
# several 16-bit variables pre-exist in memory, including
#  CURSEC, POSN_ERR, S_GRAT, and K_GRAT.  In addition,
#  two tables S_TABLE and C_TABLE are of a size equal to
#  the possible range of values of CURSEC.  16-bit results
#  of this calculation in memory include S_VALUE, K_VALUE,
#  R_CP, and two accumulators S_ACCUM and K_ACCUM:

#  S_VALUE := S_TABLE(CURSEC)
#  K_VALUE := C_TABLE(CURSEC)
#  S_ACCUM := S_ACCUM + ((S_VALUE*POSN_ERR)/64)
#  K_ACCUM := K_ACCUM + ((K_VALUE*POSN_ERR)/64)
#  R_CP := (S_VALUE*S_GRAT + K_VALUE*K_GRAT) / 32

# to optimize memory accessing, all routines may assume
#  that variables S_VALUE, S_GRAT, S_ACCUM, K_VALUE, K_GRAT,
#  K_ACCUM are consecutive in memory in whatever order is
#  optimal for their instruction set, while CURSEC. POSN_ERR,
#  S_TABLE, and C_TABLE are at unrelated locations.  R_CP
#  can be in either place.

# the order in this version is S_VALUE, S_ACCUM, S_GRAT,
#  K_VALUE, K_ACCUM, K_GRAT, R_CP.

# this code is not warranted to be correct nor operative, and is
#  intended for performance benchmarking purposes only
```

Bytes	ID				
8	B		ldos	CURSEC,r3	# get variables
8	C		ldis	POSN_ERR,r4	
8	D		ldis	S_TABLE[r3*2],r5	# get S_VALUE from table
8	E		lda	S_VALUE,r6	# start pointer into variables
4	F		mov	r6,r12	# copy that
4	G	lp:	muli	r4,r5,r7	# S_VALUE*POSN_ERR
4	H		stis	r5,0(r6)	# save S_VALUE
4	I		ldis	2(r6),r8	# get accum
4	J		ldis	4(r6),r9	# get S_GRAT
4	K		shri	6,r7,r7	# divide by 64
4	L		addi	r7,r8,r8	# accumulate
4	M		muli	r5,r9,r9	# S_VALUE*S_GRAT
4	N		stis	r8,2(r6)	# save accum
4	O		cmpibne	r6,r12,kdone	
4	P		addi	6,r6	
8	Q		ldis	C_TABLE[r3*2],r5	# get K_VALUE from table
4	R		mov	r9,r13	
4	S		b	lp	
4	T	kdone:	addi	r13,r9,r9	#S_VALUE*S_GRAT + K_VALUE*K_GRAT
4	U		shri	5,r9,r9	# divide by 32
4	V		stis	r9,6(r6)	# save in R_CP
104	92 (see chart)				

Multiply/Accumulate: 80960KA

57						X	D	CF													
58						X	EA	D	CF												
59						X	AonB		D	CF											
60						X	DonB			D	CF										
61						X	W				D	CF									
62						X		AonB				D									
63						X		DonB													
64						X		X	AonB												
65						X			DonB												
66						X			X												
67						X															
68						X															
69						X															
70						X															
71						X															
72										X		X									
73										X		X									
74											X	X									
75												X									
76												X									
77												X	CF								
78												X	D	CF							
79												X	EA	D							
80												X	Aon	X							
81												X	Don	X							
82												X	W	X							
83												X		X							
84												X		X							
85												X							CF		
86												X							D	F	
87																			X	D	F
88																				X	
89																				X	
90																				X	AonB
91																					DonB
92																					W

8

INTERRUPT: 680X0

```
        ; This 680x0 code handles Rx Status interrupts from a 16C30.
        ; It is evaluated for an overrun condition, so that End Of
        ;  Frame processing, which is handled by a separate subroutine,
        ;  doesn't count toward the totals.
        ; It is not warranted to be correct nor operative, and is
        ;  intended for performance benchmarking purposes only

        ; It assumes the USC is in a 24-bit addressed memory space
        ;  and that the hardware includes byte/word addressing
        ;  hardware (i.e., an environment like the IUSC/AT Starter Kit)

Bytes Clks (CPU32)
—     —
      32  interrupt (per CPU32 ref man p.8-27)
      rxStInt:
       ; save registers
4     73      MOVEM.L A0-6/D0-7,-(SP)       ; could save less, but we don't
       ; begin handling the interrupt   know what procEOF does...
6     7       LEA     uscBase,A0
6     10      MOVE.B  #clrIP+RS_IP,DCCR(A0)        ; clear IP
4     7       MOVE.W  RCSR(A0),D0; get status
4     4       BTST    #rxOv,D0          ; test overflow
2     4       BEQ     noOver            ; around if not
       ; handle Rx overrun
6     10      MOVE.B  #EnterHuntMode,RCSR+1(A0) ; force Rx into Hunt
6     12      OR.B    #PurgeRx,CCAR+1(A0) ; issue purge Rx command
       ; handle RxBound (end of frame)
4     4       BTST    #rxBnd,D0
2     10      BZ      noEOF             ; around if no End of Frame
4             BSR     procEOF                   ; call subr if so
       ; clear interrupt hardware
4     5 noEOF: AND.B          #$F6,D0                   ; mask status
4     6       MOVE.B  D0,RCSR(A0) ; unlatch status bits we saw
4     7       MOVE.B  RICR(A0),D0  ; save arm bits
4     6       CLR.B   RICR(A0)          ; disarm all
4     6       MOVE.B  D0,RICR(A0)  ; rearm
6     10      MOVE.B  #clrIUS+RS_IUS,DCCR+1(A0)
       ; restore regs, dismiss interrupt and return
4     74      MOVEM.L (SP)+,A0-6/D0-7
2     26      RTE
—     —
80    313 clocks (CPU32)
      288 clocks (68020)
```

8

INTERRUPT: 80380

```
        ; This 380 code handles Rx Status interrupts from a 16C30.
        ; It is evaluated for an overrun condition, so that End Of
        ;  Frame processing, which is handled by a separate subroutine,
        ;  doesn't count toward the totals.
        ; It is not warranted to be correct nor operative, and is
        ;  intended for performance benchmarking purposes only

        ; It assumes the USC is in a 24-bit addressed memory space
        ;  and that the hardware includes byte/word addressing
        ;  hardware (i.e., an environment like the IUSC/AT Starter Kit)

Bytes Clks
— —
        18 (interrupt time)
        rxStInt:
        ; save registers
2       2       DDIR  LW
2       6       PUSH  SR              ; save old control settings
3       4       LDCTL SR,intBank      ; one reg bank dedicated
                                      ; for unnested interrupts
        ; begin handling the interrupt
2       2       DDIR  IB
5       4       LD    IX,uscBase      ; set 24-bit address of USC
4       6       LD    (IX+DCCR),clrIP+RS_IP ; clear IP bit
4       7       LD    BC,(IX+RCSR)    ; get status
2       2       BIT   rxOv,C
2       2/6     JR    Z,noOver        ; around if no overflow flag
        ; handle Rx overrun
4       6       LD    (IX+RCSR+1),EnterHuntMode ; force Rx hunt mode
3       7       LD    A,(IX+CCAR+1)
2       2       OR    A,PurgeRx
3       6       LD    (IX+CCAR+1),A; issue purge Rx command
        ; handle RxBound (End of Frame)
2       2/6 noOver:BIT rxBd,C
3       2       CALL  NZ,procEOF      ; call End of Frame procedure
        ; clear interrupt hardware
2       2       AND   C,0F6H
3       6       LD    (IX+RCSR),C     ; unlatch status bits we saw
3       7       LD    A,(IX+RICR)     ; get IA bits
4       6       LD    (IX+RICR),0     ; drop IA bits
3       6       LD    (IX+RICR),A     ; rearm them
4       6       LD    (IX+DCCR+1),clrIUS+RS_IUS   ; clear IUS
        ; restore registers, dismiss interrupt and return
2       2       DDIR  LW
2       8       POP   SR
2       8       RETI
—       —
66      133
```

8

Interrupt: 80960KA

71												W			X											
72															X			F								
73																		F								
74																		F								
75																		D	F							
76																		EA	D	F						
77																			X	D	F					
78																	AonB		X	EA	D					
79	.																DonB				EA					
80																	W									
81																		AonB								
82																		DonB								
83																		X		AonB						
84																				DonB						
85																				W						
86																						F				
87																						F				
88																						F				
89																						D	F			
90																						EA	D	F		
91																						X	EA	D	F	
92																					AonB					
93																					DonB					
94																					W					
95																							AonB			
96																							DonB			
97																							W	X		
98																								X		
99																								X		
100																								X		
101																								X		
102																								X		
103																								X		
104												restore arithmetic controls from int record														A
105																										A
106																										A
107												restore process controls from int record														P
108																										P
109																										P
110												copy resumption record														R
111																										R
112																										R
113																										R
114																										R
115																										R
116																										R
117																										R
118												dealloc stack frame, remove interrupt record														IS
119																										IS
120																										IS
121																										IS
122												switch back to former stack														OS
123																										OS

8

Summary of Benchmarks

Normalized to 25MHz 80380										
Proc	i 80186	CPU32	CPU32	68020	68020	Z380	Z380	Z380	80960KA	80960KA
clock rate, MHz	16	16	25	16	25	16	25	40	16	25
clk period, nS	62.5	62.5	40	62.5	40	62.5	40	25	62.5	40
I/O Loop (bytes)	61	92	92	92	92	65	65	65	120	120
Bytes, Z380=1	0.94	1.42	1.42	1.42	1.42	1.00	1.00	1.00	1.85	1.85
I/O Loop (formula)	60+80*N	50+77*N	50+77*N	56+48N	56+48N	41+53*N	41+53*N	41+53*N	56+24*N	56+24*N
I/O Loop (clks @ N=1)	140	127	127	104	104	94	94	94	80	80
I/O Loop (nS @ N=1)	8750	7938	5080	6500	4160	5875	3760	2350	5000	3200
nS, N=1, 25MHz Z380=1	2.33	2.11	1.35	1.73	1.11	1.56	1.00	0.63	1.33	0.85
I/O Loop (clks @ N=8)	700	666	666	440	440	465	465	465	248	248
I/O Loop (nS @ N=8)	43750	41625	26640	27500	17600	29063	18600	11625	15500	9920
nS, N=8, 25MHz Z380=1	2.35	2.24	1.43	1.48	0.95	1.56	1.00	0.63	0.83	0.53
signed bytes (bytes)	63	64	64	64	64	52	52	52	76	76
bytes, Z380=1	1.21	1.23	1.23	1.23	1.23	1.00	1.00	1.00	1.46	1.46
signed bytes (clks)	79	53	53	46	46	43	43	43	31	31
signed bytes (nS)	4917	3292	2107	2875	1840	2667	1707	1067	1917	1227
nS, 25MHz Z380=1	2.88	1.93	1.23	1.68	1.08	1.56	1.00	0.63	1.12	0.72
multiply/accum (bytes)	72	54	54	54	54	95	95	95	104	104
bytes (Z380=1)	0.76	0.57	0.57	0.57	0.57	1.00	1.00	1.00	1.09	1.09
multiply/accum (clks)	404	204	204	212	212	254	254	254	92	92
multiply/accum (nS)	25250	12750	8160	13250	8480	15875	10160	6350	5750	3680
nS, 25MHz Z380=1	2.49	1.25	0.80	1.30	0.83	1.56	1.00	0.63	0.57	0.36
interrupt (bytes)	63	80	80	80	80	66	66	66	92	92
bytes (Z380=1)	0.95	1.21	1.21	1.21	1.21	1.00	1.00	1.00	1.39	1.39
interrupt (clks)	328	313	313	288	288	133	133	133	123	123
interrupt (nS)	20500	19563	12520	18000	11520	8313	5320	3325	7688	4920
nS, 25MHz Z380=1	3.85	3.68	2.35	3.38	2.17	1.56	1.00	0.63	1.45	0.92
Block move, clks/byte	4.00	4.25	4.25	2.875	2.875	2.75	2.75	2.75	1.25	1.25
Block move, nS/byte	250	266	170	180	115	172	110	69	78	50
nS, 25MHz Z380=1	2.27	2.41	1.55	1.63	1.05	1.56	1.00	0.63	0.71	0.45
Bytes, ave of Z380=1	0.97	1.11	1.11	1.11	1.11	1.00	1.00	1.00	1.45	1.45
nS, ave of 25 MHz Z380=1	2.70	2.27	1.45	1.87	1.20	1.56	1.00	0.63	1.00	0.64
est for 80960SA*									1.63	1.04
ave of all 25MHz Z380=1	2.00	1.81	1.31	1.56	1.16	1.34	1.00	0.78	1.18	0.96
est for 80960SA*									1.56	1.20
* 80960SA times estimated per intel's Dhrystone figures: 19740 for KA, 12145 for SA										

8

ZiLOG

Z380™
QUESTIONS AND ANSWERS

GENERAL OVERVIEW

Q: What is currently assigned as the value in the Chip ID version register?
A: Currently the value 00H is assigned to the Z380 MPU, and other values are reserved. Note that the internal I/O address for this register is 0FFH.

Q: Can data be accessed in the memory space beyond the 64K boundary in Native mode?
A: Yes. The Z380 in Native/Word mode behaves exactly like the Z80, but has access to the entire 4 Gbytes of memory for data and 4G locations of I/O space because the upper 16 bits of all CPU registers (except the PC) are still accessable to the software using new Z380 instructions. Note that the program must reside within the first 64K of memory because the upper word of the PC is not accessable in Native mode and is always all zeros in this mode.

Q: Z380 is binary code compatible with which processor?
A: The Z80 and Z180. Please note that the Z380 is not binary code compatible with the Z280.

Q: What are the two modes that Z380 can operate in?
A: The Z380 can operate in Native mode or Extended mode. In Native mode all of the address manipulations operate on 16-bit quantities whereas in Extended mode all of the address manipulations operate on 32-bit quantities.

Q: What are the specifics of the Z380 PC in Extended mode?
A: In extended mode the PC increments across all 32 bits since the entire 4G Byte of addressing capability is in use.

Q: How would one determine during a memory read, whether or not the cycle is instruction fetch or data?
A: There is a Fetch signal available in the PGA version that goes active during an instruction fetch.

Q: What are the Interrupt acknowledge and I/O transactions timings relative to?
A: All of the Interrupt Acknowledge and I/O transactions are in reference to the I/O clock which is a program controlled divided-down version of the BUSCLK.

Q: How can the Z380 return from Extended to Native mode of operation?
A: Hardware Reset is the ONLY way that one can go back to Native mode.

Q: Is the Z380 an Intel based architecture or Motorola based?
A: The Z380, being compatible with the original Z80, is Intel based. Intel based means the memory organization is the "LSbyte first followed by MSbytes" whereas the Motorola architecture has "MSbyte first followed by LSbytes".

RESET

Q: What is the effect of the reset on the Z380?
A: Reset will cause the address and data lines to float. All of the control lines will go to the inactive state.

Q: What is the status of the memory chip select signals during Reset?
A: They are all tri-stated, since the Address bus is tri-stated.

Q: Will reset affect all of the registers on Z380?
A: Not all of the registers are effected by Reset. CPU registers are not affected by Reset. Please refer to Product spec DC#6003-02 page 102 for the effect of Reset on Z380 CPU and related I/O registers.

Q: How long do one need to have the /RESET line active for proper operation?
A: The /RESET line must be kept Low for a minimum of 10 BUSCLK cycles. The /RESET signal does not need to be synchronized to BUSCLK.

Q: When is the /RESET signal be internally by the CPU?
A: The /RESET input signal may be asynchronous to BUSCLK, though it is sampled internally by the falling edge of BUSCLK. For proper initialization of the MPU V_{DD} must be within operating specification and BUSCLK must be stable for more than 10 cycles with /RESET held low.

Q: Does the /RESET input include a Schmitt-trigger buffer?
A: Yes. The /RESET input on Z380 includes a Schmitt-trigger buffer to facilitate power-on reset generation through a simple RC network.

Q: How are the devices external to the Z380 MPU that are clocked by IOCLK affected by /RESET pulse width?
A: This depends on the specific device, but in general they will require a /RESET pulse width that spans several IOCLK cycles for proper initialization.

Q: How many BUSCLK cycles after the deassertion of /RESET will the Z380 proceed to fetch the first instruction?
A: The first memory read, for an instruction fetch, will start 3.5 BUSCLK cycles after the deassertion of /RESET, providing that the proper setup and hold times are met with respect to the BUSCLK falling edge.

Q: When is the first IOCLK rising edge after deassertion of /RESET signal?
A: The first rising edge of IOCLK occurs 11.5 BUSCLK cycles after the deassertion of /RESET, providing that the proper setup and hold times are met with respect to the BUSCLK falling edge. This first rising edge on IOCLK is proceeded by a minimum of 4 BUSCLK cycles where IOCLK is Low.

Q: What happens if the /BREQ signal is active when /RESET is deasserted?
A: In this case the Z380 will relinquish the bus instead of fetching the first instruction, but the IOCLK synchronization will still take place as it normally does.

POWER DOWN MODE

Q: What are the status of the output drivers when the CPU is in power down situation?
A: When the Z380 is without the power the output drivers appear to be in a high impedance state.

Q: How many ways are available to exit the Standby mode?
A: One can exit standby mode by: /BREQ, /RESET, /NMI, or /INT0-3. Note that /BREQ can be disabled as a Standby mode exit condition with a bit in the Standby Mode Control Register (SMCR) at internal I/O address 00000016H. Also, /INT0-3 will only cause an exit from the Standby mode if interrupts were globally enabled (with the IEF1 flag) when the Standby mode was entered.

Q: How could a user select the warm-up time appropriate for the crystal being used?
A: The WM2-WM0 bits in the Standby Mode Control Register (SMCR) at internal I/O address 00000016H control the warm-up time for the crystal oscillator when exiting the Standby mode.

Q: If the Standby mode option is not enabled, how does the Z380 interpret the SLP (Sleep) instruction?
A: In this case the SLP instruction is interpreted and executed identically to the HALT instruction, stopping the Z380 from further instruction execution.

Q: In the above case what would happen to /HALT signal?
A: In this case the /HALT signal goes to active (Low) to indicate that the Z380 is in the Halt state.

INTERRUPT SECTION

Q: What is the state of the IEF1 and IEF2 flags after execution of the DI (Disable Interrupt) instruction for the Z380?
A: Both IEF1 and IEF2 are set to zero by the DI instruction.

Q: What are the specifics of /INT0 Mode 3 for the Z380?
A: Mode 3 is similar to Mode 2 (as in the Z180 or Z80) except that a 16-bit interrupt vector is expected from the peripherals.

Q: How can the user take advantage of INT0 mode 3 with 8-bit I/O devices?
A: All of the upper 8 bits of the data bus need to be pulled either High or Low with external resistors.

Q: How many clocks are required for the Interrupt sequence in Interrupt mode 2 on the Z380?
A: With no wait states and a 1X I/O bus, the time from /INT0 assertion to the start of first service routine instruction fetch (Interrupt Mode 2) is 18 clocks.

Q: Is there a problem with interrupt vectors in Extended mode?
A: In Extended mode the Interrupt Vector in Interrupt Mode 2 has the two least significant bits both "0". This can cause a problem when connecting to Z80/Z8500 peripherals if the vector includes status from those devices. This is because most of these devices modify the vector starting with the bit just after the least-significant bit. Thus in certain cases this bit may be returned as a "1" from the interrupting device.

Q: How would the user access the Iz register (the Interrupt Register Extension)?
A: The LD I,HL and LD HL,I instructions (in Long Word mode) will transfer 32 bits to or from the I register.

Appendix A

APPENDIX A
Z380™ CPU INSTRUCTION FORMATS

Four formats are used to generate the machine language bit encoding for the Z380 CPU instructions. Also, the Z380 CPU has eight Decoder Directives which work as a special escape sequence to the certain instructions, to expand its capability as explained in Chapter 3.

The bit encoding of the Z380 CPU instructions are partitioned into bytes. Every instructions encoding contains one byte dedicated to specifying the type of operation to be performed; this byte is referred to as the instruction's operation code, or opcode. Besides specifying a particular operation, opcode typically include bit encoding specifying the operand addressing mode for the instruction and identifying any general purpose registers used by the instruction. Along with the opcode, instruction encoding may include bytes that contain an address, displacement, and/or immediate value used by the instruction, and special bytes called "escape codes" that determine the meaning of the opcode itself.

By themselves, one byte opcode would allow the encoding of only 256 unique instructions. Therefore, special "escape codes" that precede the opcode in the instruction encoding are used to expand the number of possible instructions. There are two types of escape codes; addressing mode and opcode. Escape codes for the Z80 original instructions are one bytes in length, and the escape codes used to expand the Z380 instructions are one or two bytes in length.

These instruction formats are differentiated by the opcode escape value used. Format 1 is for instructions without an opcode escape byte(s), Format 2 is for instructions with an opcode escape byte. Format 3 is for instructions whose opcode escape byte has the value 0CBH, and Format 4 is for instructions whose escape bytes are 0ED, followed by 0CBH.

For the opcode escape byte, the Z380 CPU uses 0DDH and 0FDH as well, which on the Z80 CPU, these are used only as an address escape byte.

In Format 2 and 4, the opcode escape byte immediately precedes the opcode byte itself.

In Format 3, a 1-byte displacement may be between the opcode escape byte and opcode itself. Opcode escape bytes are used to distinguish between two different instructions with the same opcode bytes, thereby allowing more than 256 unique instructions. For example, the 01H opcode, when alone, specifies a form of a Load Register Word instruction; when proceeded by 0CBH escape code, the opcode 01H specifies a Rotate Left Circular instruction.

Format 3 instructions with DDIR Immediate data Decoder Directives, 1 to 3 bytes of displacement is between the opcode escape byte and opcode itself.

Format 4 instructions are proceeded by 0EDH, 0CBH, and a opcode. Optionally, with immediate word field follows.

Addressing mode escape codes are used to determine the type of encoding for the addressing mode field within an instruction's opcode, and can be used in instructions with and without opcode escape value. An addressing mode escape byte can have the value of 0DDH or 0FDH. The addressing mode escape byte, if present, is always the first byte of the instruction's machine code, and is immediately followed by either the opcode (Format 1), or the opcode escape byte (Format 2 and 3). For example, the 46H opcode, when alone, specifies a Load B register from memory location pointed by (HL) register; when proceeded by the 0DDH escape byte, the opcode 46H specifies a Load B register from the memory location pointed by (IX+d).

Appendix B

APPENDIX B

Z380™ INSTRUCTIONS IN ALPHABETIC ORDER

This Appendix contains a quick reference guide when programming.

It has the Z380 instructions sorted by alphabetic order.

The column "Mode" indicates whether the instruction is affected by DDIR immediate Decoder Directives, Extended mode or Native mode of operation, and Word or Long Word mode of operation; "I" means the instruction can be used with DDIR IM to expand its immediate constant, "X" means that the operation of the instruction is affected by the XM status bit, and "L" means that the instruction is affected by LW status bit, or can be used with DDIR LW or DDIR W. The Native/Extended modes, Word/Long Word modes and Decoder Directives are discussed in Chapter 3 in this manual.

Source Code		Mode	Object Code			
AND	IYU		FD	A4		
AND	L		A5			
ANDW	(IX+12H)	I	DD	E6	12	
ANDW	(IY+12H)	I	FD	E6	12	
ANDW	1234H		ED	A6	34	12
ANDW	BC		ED	A4		
ANDW	DE		ED	A5		
ANDW	HL		ED	A7		
ANDW	HL,(IX+12H)	I	DD	E6	12	
ANDW	HL,(IY+12H)	I	FD	E6	12	
ANDW	HL,1234H		ED	A6	34	12
ANDW	HL,BC		ED	A4		
ANDW	HL,DE		ED	A5		
ANDW	HL,HL		ED	A7		
ANDW	HL,IX		DD	A7		
ANDW	HL,IY		FD	A7		
ANDW	IX		DD	A7		
ANDW	IY		FD	A7		
BIT	0,(HL)		CB	46		
BIT	0,(IX+12H)	I	DD	CB	12	46
BIT	0,(IY+12H)	I	FD	CB	12	46
BIT	0,A		CB	47		
BIT	0,B		CB	40		
BIT	0,C		CB	41		
BIT	0,D		CB	42		
BIT	0,E		CB	43		
BIT	0,H		CB	44		
BIT	0,L		CB	45		
BIT	1,(HL)		CB	4E		
BIT	1,(IX+12H)	I	DD	CB	12	4E
BIT	1,(IY+12H)	I	FD	CB	12	4E
BIT	1,A		CB	4F		
BIT	1,B		CB	48		
BIT	1,C		CB	49		
BIT	1,D		CB	4A		
BIT	1,E		CB	4B		
BIT	1,H		CB	4C		
BIT	1,L		CB	4D		
BIT	2,(HL)		CB	56		
BIT	2,(IX+12H)	I	DD	CB	12	56
BIT	2,(IY+12H)	I	FD	CB	12	56
BIT	2,A		CB	57		
BIT	2,B		CB	50		
BIT	2,C		CB	51		
BIT	2,D		CB	52		
BIT	2,E		CB	53		
BIT	2,H		CB	54		
BIT	2,L		CB	55		
BIT	3,(HL)		CB	5E		
BIT	3,(IX+12H)	I	DD	CB	12	5E
BIT	3,(IY+12H)	I	FD	CB	12	5E
BIT	3,A		CB	5F		
BIT	3,B		CB	58		
BIT	3,C		CB	59		

Source Code		Mode	Object Code			
BIT	3,D		CB	5A		
BIT	3,E		CB	5B		
BIT	3,H		CB	5C		
BIT	3,L		CB	5D		
BIT	4,(HL)		CB	66		
BIT	4,(IX+12H)	I	DD	CB	12	66
BIT	4,(IY+12H)	I	FD	CB	12	66
BIT	4,A		CB	67		
BIT	4,B		CB	60		
BIT	4,C		CB	61		
BIT	4,D		CB	62		
BIT	4,E		CB	63		
BIT	4,H		CB	64		
BIT	4,L		CB	65		
BIT	5,(HL)		CB	6E		
BIT	5,(IX+12H)	I	DD	CB	12	6E
BIT	5,(IY+12H)	I	FD	CB	12	6E
BIT	5,A		CB	6F		
BIT	5,B		CB	68		
BIT	5,C		CB	69		
BIT	5,D		CB	6A		
BIT	5,E		CB	6B		
BIT	5,H		CB	6C		
BIT	5,L		CB	6D		
BIT	6,(HL)		CB	76		
BIT	6,(IX+12H)	I	DD	CB	12	76
BIT	6,(IY+12H)	I	FD	CB	12	76
BIT	6,A		CB	77		
BIT	6,B		CB	70		
BIT	6,C		CB	71		
BIT	6,D		CB	72		
BIT	6,E		CB	73		
BIT	6,H		CB	74		
BIT	6,L		CB	75		
BIT	7,(HL)		CB	7E		
BIT	7,(IX+12H)	I	DD	CB	12	7E
BIT	7,(IY+12H)	I	FD	CB	12	7E
BIT	7,A		CB	7F		
BIT	7,B		CB	78		
BIT	7,C		CB	79		
BIT	7,D		CB	7A		
BIT	7,E		CB	7B		
BIT	7,H		CB	7C		
BIT	7,L		CB	7D		
BTEST			ED	CF		
CALL	1234H	I X	CD	34	12	
CALL	C,1234H	I X	DC	34	12	
CALL	M,1234H	I X	FC	34	12	
CALL	NC,1234H	I X	D4	34	12	
CALL	NZ,1234H	I X	C4	34	12	
CALL	P,1234H	I X	F4	34	12	
CALL	PE,1234H	I X	EC	34	12	
CALL	V, 1234H	I X	EC	34	12	
CALL	PO,1234H	I X	E4	34	12	

Source Code		Mode	Object Code
DEC	IY	X	FD 2B
DEC	IYL		FD 2D
DEC	IYU		FD 25
DEC	L		2D
DEC	SP	X	3B
DECW	BC	X	0B
DECW	DE	X	1B
DECW	HL	X	2B
DECW	IX	X	DD 2B
DECW	IY	X	FD 2B
DECW	SP	X	3B
DI	1FH		DD F3 1F
DI			F3
DIVUW	(IX+12H)	I	DD CB 12 BA
DIVUW	(IY+12H)	I	FD CB 12 BA
DIVUW	1234H		ED CB BF
DIVUW	BC		ED CB B8
DIVUW	DE		ED CB B9
DIVUW	HL		ED CB BB
DIVUW	HL,(IX+12H)	I	DD CB 12 BA
DIVUW	HL,(IY+12H)	I	FD CB 12 BA
DIVUW	HL,1234H		ED CB BF
DIVUW	HL,BC		ED CB B8
DIVUW	HL,DE		ED CB B9
DIVUW	HL,HL		ED CB BB
DIVUW	HL,IX		ED CB BC
DIVUW	HL,IY		ED CB BD
DIVUW	IX		ED CB BC
DIVUW	IY		ED CB BD
DJNZ	123456H	X	FD 10 56 34 12
DJNZ	1234H	X	DD 10 34 12
DJNZ	12H	X	10 12
EI	1FH		DD FB 1F
EI			FB
escape			CB
escape			DD
escape			ED
escape			FD
escape			ED CB
escape			DD CB
escape			FD CB
EX	(SP),HL	L	E3
EX	(SP),IX	L	DD E3
EX	(SP),IY	L	FD E3
EX	A,(HL)		ED 37
EX	A,A		ED 3F
EX	A,A'		CB 37
EX	A,B		ED 07
EX	A,C		ED 0F
EX	A,D		ED 17
EX	A,E		ED 1F
EX	A,H		ED 27
EX	A,L		ED 2F
EX	AF,AF'		08
EX	B,B'	CB	30

Source Code		Mode	Object Code
EX	BC,BC'	L	ED CB 30
EX	BC,DE	L	ED 05
EX	BC,HL	L	ED 0D
EX	BC,IX	L	ED 03
EX	BC,IY	L	ED 0B
EX	C,C'		CB 31
EX	D,D'		CB 32
EX	DE,DE'	L	ED CB 31
EX	DE,HL	L	EB
EX	DE,IX	L	ED 13
EX	DE,IY	L	ED 1B
EX	E,E'		CB 33
EX	H,H'		CB 34
EX	HL,HL'	L	ED CB 33
EX	HL,IX	L	ED 33
EX	HL,IY	L	ED 3B
EX	IX,IX'		ED CB 34
EX	IX,IY	L	ED 2B
EX	IY,IY'	L	ED CB 35
EX	L,L'		CB 35
EXALL			ED D9
EXTS	A	L	ED 65
EXTS		L	ED 65
EXTSW	HL		ED 75
EXTSW			ED 75
EXX			D9
EXXX			DD D9
EXXY			FD D9
HALT			76
IM	0		ED 46
IM	1		ED 56
IM	2		ED 5E
IM	3		ED 4E
IN	A,(12H)		DB 12
IN	A,(C)		ED 78
IN	B,(C)		ED 40
IN	C,(C)		ED 48
IN	D,(C)		ED 50
IN	E,(C)		ED 58
IN	H,(C)		ED 60
IN	L,(C)		ED 68
IN0	(12H)		ED 30 12
IN0	A,(12H)		ED 38 12
IN0	B,(12H)		ED 00 12
IN0	C,(12H)		ED 08 12
IN0	D,(12H)		ED 10 12
IN0	E,(12H)		ED 18 12
IN0	H,(12H)		ED 20 12
IN0	L,(12H)		ED 28 12
INA	A,(1234H)	I	ED DB 34 12
INAW	HL,(1234H)	I	FD DB 34 12
INC	(HL)		34
INC	(IX+12H)	I	DD 34 12
INC	(IY+12H)	I	FD 34 12
INC	A		3C

B

Source Code	Mode		Object Code
LD (IX+12H),IY	I	L	DD CB 12 2B
LD (IX+12H),L	I		DD 75 12
LD (IY+12H),34H	I		FD 36 34 12
LD (IY+12H),A	I		FD 77 12
LD (IY+12H),B	I		FD 70 12
LD (IY+12H),BC	I	L	FD CB 12 0B
LD (IY+12H),C	I		FD 71 12
LD (IY+12H),D	I		FD 72 12
LD (IY+12H),DE	I		FD CB 12 1B
LD (IY+12H),E	I	L	FD 73 12
LD (IY+12H),H	I		FD 74 12
LD (IY+12H),HL	I	L	FD CB 12 3B
LD (IY+12H),IX	I	L	FD CB 12 2B
LD (IY+12H),L	I		FD 75 12
LD (SP+12H),BC	I	L	DD CB 12 09
LD (SP+12H),DE	I	L	DD CB 12 19
LD (SP+12H),HL	I	L	DD CB 12 39
LD (SP+12H),IX	I	L	DD CB 12 29
LD (SP+12H),IY	I	L	FD CB 12 29
LD A,(1234H)	I		3A 34 12
LD A,(BC)			0A
LD A,(DE)			1A
LD A,(HL)			7E
LD A,(IX+12H)	I		DD 7E 12
LD A,(IY+12H)	I		FD 7E 12
LD A,12H			3E 12
LD A,A			7F
LD A,B			78
LD A,C			79
LD A,D			7A
LD A,E			7B
LD A,H			7C
LD A,I			ED 57
LD A,IXL			DD 7D
LD A,IXU			DD 7C
LD A,IYL			FD 7D
LD A,IYU			FD 7C
LD A,L			7D
LD A,R			ED 5F
LD B,(HL)			46
LD B,(IX+12H)	I		DD 46 12
LD B,(IY+12H)	I		FD 46 12
LD B,12H			06 12
LD B,A			47
LD B,B			40
LD B,C			41
LD B,D			42
LD B,E			43
LD B,H			44
LD B,IXL			DD 45
LD B,IXU			DD 44
LD B,IYL			FD 45
LD B,IYU			FD 44
LD B,L			45

Source Code	Mode		Object Code
LD BC,(1234H)	I	L	ED 4B 34 12
LD BC,(BC)		L	DD 0C
LD BC,(DE)		L	DD 0D
LD BC,(HL)		L	DD 0F
LD BC,(IX+12H)	I	L	DD CB 12 03
LD BC,(IY+12H)	I	L	FD CB 12 03
LD BC,(SP+12H)	I	L	DD CB 12 01
LD BC,1234H	I	L	01 34 12
LD BC,BC		L	ED 02
LD BC,DE		L	DD 02
LD BC,HL		L	FD 02
LD BC,IX		L	DD 0B
LD BC,IY		L	FD 0B
LD C,(HL)			4E
LD C,(IX+12H)	I		DD 4E 12
LD C,(IY+12H)	I		FD 4E 12
LD C,12H			0E 12
LD C,A			4F
LD C,B			48
LD C,C			49
LD C,D			4A
LD C,E			4B
LD C,H			4C
LD C,IXL			DD 4D
LD C,IXU			DD 4C
LD C,IYL			FD 4D
LD C,IYU			FD 4C
LD C,L			4D
LD D,(HL)			56
LD D,(IX+12H)	I		DD 56 12
LD D,(IY+12H)	I		FD 56 12
LD D,12H			16 12
LD D,A			57
LD D,B			50
LD D,C			51
LD D,D			52
LD D,E			53
LD D,H			54
LD D,IXL			DD 55
LD D,IXU			DD 54
LD D,IYL			FD 55
LD D,IYU			FD 54
LD D,L			55
LD DE,(1234H)	I	L	ED 5B 34 12
LD DE,(BC)		L	DD 1C
LD DE,(DE)		L	DD 1D
LD DE,(HL)		L	DD 1F
LD DE,(IX+12H)	I	L	DD CB 12 13
LD DE,(IY+12H)	I	L	FD CB 12 13
LD DE,(SP+12H)	I	L	DD CB 12 11
LD DE,1234H	I	L	11 34 12
LD DE,BC		L	ED 12
LD DE,DE		L	DD 12
LD DE,HL		L	FD 12

B

Source Code		Mode		Object Code			
LD	L,H			6C			
LD	L,L			6D			
LD	R,A			ED	4F		
LD	SP,(1234H)	I	L	ED	7B	34	12
LD	SP,1234H	I	L	31	34	12	
LD	SP,HL		L	F9			
LD	SP,IX		L	DD	F9		
LD	SP,IY		L	FD	F9		
LDCTL	A,DSR			ED	D0		
LDCTL	A,XSR			DD	D0		
LDCTL	A,YSR			FD	D0		
LDCTL	DSR,01H			ED	DA	01	
LDCTL	DSR,A			ED	D8		
LDCTL	HL,SR		L	ED	C0		
LDCTL	SR,01H			DD	CA	01	
LDCTL	SR,A			DD	C8		
LDCTL	SR,HL		L	ED	C8		
LDCTL	XSR,01H			DD	DA	01	
LDCTL	XSR,A			DD	D8		
LDCTL	YSR,01H			FD	DA	01	
LDCTL	YSR,A			FD	D8		
LDD				ED	A8		
LDDR				ED	B8		
LDDRW			L	ED	F8		
LDDW			L	ED	E8		
LDI				ED	A0		
LDIR				ED	B0		
LDIRW			L	ED	F0		
LDIW			L	ED	E0		
LDW	(BC),1234H	I	L	ED	06	34	12
LDW	(DE),1234H	I	L	ED	16	34	12
LDW	(HL),1234H	I	L	ED	36	34	12
LDW	HL,I		L	DD	57		
LDW	I,HL		L	DD	47		
MLT	BC			ED	4C		
MLT	DE			ED	5C		
MLT	HL			ED	6C		
MLT	SP			ED	7C		
MTEST				DD	CF		
MULTUW	(IX+12H)	I		DD	CB	12	9A
MULTUW	(IY+12H)	I		FD	CB	12	9A
MULTUW	1234H			ED	CB	9F	
MULTUW	BC			ED	CB	98	
MULTUW	DE			ED	CB	99	
MULTUW	HL			ED	CB	9B	
MULTUW	HL,(IX+12H)	I		DD	CB	12	9A
MULTUW	HL,(IY+12H)	I		FD	CB	12	9A
MULTUW	HL,1234H			ED	CB	9F	
MULTUW	HL,BC			ED	CB	98	
MULTUW	HL,DE			ED	CB	99	
MULTUW	HL,HL			ED	CB	9B	
MULTUW	HL,IX			ED	CB	9C	
MULTUW	HL,IY			ED	CB	9D	
MULTUW	IX			ED	CB	9C	
MULTUW	IY			ED	CB	9D	

Source Code		Mode	Object Code				
MULTW	(IX+12H)	I	DD	CB	12	92	
MULTW	(IY+12H)	I	FD	CB	12	92	
MULTW	1234H		ED	CB	97	34	12
MULTW	BC		ED	CB	90		
MULTW	DE		ED	CB	91		
MULTW	HL		ED	CB	93		
MULTW	HL,(IX+12H)	I	DD	CB	12	92	
MULTW	HL,(IY+12H)	I	FD	CB	12	92	
MULTW	HL,1234H		ED	CB	97	34	12
MULTW	HL,BC		ED	CB	90		
MULTW	HL,DE		ED	CB	91		
MULTW	HL,HL		ED	CB	93		
MULTW	HL,IX		ED	CB	94		
MULTW	HL,IY		ED	CB	95		
MULTW	IX		ED	CB	94		
MULTW	IY		ED	CB	95		
NEG	A		ED	44			
NEG			ED	44			
NEGW	HL		ED	54			
NEGW			ED	54			
NOP			00				
OR	(HL)		B6				
OR	(IX+12H)	I	DD	B6	12		
OR	(IY+12H)	I	FD	B6	12		
OR	12H		F6	12			
OR	A		B7				
OR	A,(HL)		B6				
OR	A,(IX+12H)	I	DD	B6	12		
OR	A,(IY+12H)	I	FD	B6	12		
OR	A,12H		F6	12			
OR	A,A		B7				
OR	A,B		B0				
OR	A,C		B1				
OR	A,D		B2				
OR	A,E		B3				
OR	A,H		B4				
OR	A,IXL		DD	B5			
OR	A,IXU		DD	B4			
OR	A,IYL		FD	B5			
OR	A,IYU		FD	B4			
OR	A,L		B5				
OR	B		B0				
OR	C		B1				
OR	D		B2				
OR	E		B3				
OR	H		B4				
OR	IXL		DD	B5			
OR	IXU		DD	B4			
OR	IYL		FD	B5			
OR	IYU		FD	B4			
OR	L		B5				
ORW	(IX+12H)	I	DD	F6	12		
ORW	(IY+12H)	I	FD	F6	12		
ORW	1234H		ED	B6	34	12	
ORW	BC		ED	B4			

B

Source Code		Mode		Object Code
RES	4,E			CB A3
RES	4,H			CB A4
RES	4,L			CB A5
RES	5,(HL)			CB AE
RES	5,(IX+12H)	I		DD CB 12 AE
RES	5,(IY+12H)	I		FD CB 12 AE
RES	5,A			CB AF
RES	5,B			CB A8
RES	5,C			CB A9
RES	5,D			CB AA
RES	5,E			CB AB
RES	5,H			CB AC
RES	5,L			CB AD
RES	6,(HL)			CB B6
RES	6,(IX+12H)	I		DD CB 12 B6
RES	6,(IY+12H)	I		FD CB 12 B6
RES	6,A			CB B7
RES	6,B			CB B0
RES	6,C			CB B1
RES	6,D			CB B2
RES	6,E			CB B3
RES	6,H			CB B4
RES	6,L			CB B5
RES	7,(HL)			CB BE
RES	7,(IX+12H)	I		DD CB 12 BE
RES	7,(IY+12H)	I		FD CB 12 BE
RES	7,A			CB BF
RES	7,B			CB B8
RES	7,C			CB B9
RES	7,D			CB BA
RES	7,E			CB BB
RES	7,H			CB BC
RES	7,L			CB BD
RESC	LCK			ED FF
RESC	LW			DD FF
reserved				ED 55
RET	C		X	D8
RET	M		X	F8
RET	NC		X	D0
RET	NS		X	F0
RET	NV		X	E0
RET	NZ		X	C0
RET	P		X	F0
RET	PE		X	E8
RET	PO		X	E0
RET	S		X	F8
RET	V		X	E8
RET	Z		X	C8
RET			X	C9
RETI			X	ED 4D
RETN			X	ED 45
RL	(HL)			CB 16
RL	(IX+12H)	I		DD CB 12 16
RL	(IY+12H)	I		FD CB 12 16

Source Code		Mode	Object Code
RL	A		CB 17
RL	B		CB 10
RL	C		CB 11
RL	D		CB 12
RL	E		CB 13
RL	H		CB 14
RL	L		CB 15
RLA			17
RLC	(HL)		CB 06
RLC	(IX+12H)	I	DD CB 12 06
RLC	(IY+12H)	I	FD CB 12 06
RLC	A		CB 07
RLC	B		CB 00
RLC	C		CB 01
RLC	D		CB 02
RLC	E		CB 03
RLC	H		CB 04
RLC	L		CB 05
RLCA			07
RLCW	(HL)		ED CB 02
RLCW	(IX+12H)	I	DD CB 12 02
RLCW	(IY+12H)	I	FD CB 12 02
RLCW	BC		ED CB 00
RLCW	DE		ED CB 01
RLCW	HL		ED CB 03
RLCW	IX		ED CB 04
RLCW	IY		ED CB 05
RLD			ED 6F
RLW	(HL)		ED CB 12
RLW	(IX+12H)	I	DD CB 12 12
RLW	(IY+12H)	I	FD CB 12 12
RLW	BC		ED CB 10
RLW	DE		ED CB 11
RLW	HL		ED CB 13
RLW	IX		ED CB 14
RLW	IY		ED CB 15
RR	(HL)		CB 1E
RR	(IX+12H)	I	DD CB 12 1E
RR	(IY+12H)	I	FD CB 12 1E
RR	A		CB 1F
RR	B		CB 18
RR	C		CB 19
RR	D		CB 1A
RR	E		CB 1B
RR	H		CB 1C
RR	L		CB 1D
RRA			1F
RRC	(HL)		CB 0E
RRC	(IX+12H)	I	DD CB 12 0E
RRC	(IY+12H)	I	FD CB 12 0E
RRC	A		CB 0F
RRC	B		CB 08
RRC	C		CB 09
RRC	D		CB 0A
RRC	E		CB 0B

Source Code	Mode	Object Code
SET 4,B		CB E0
SET 4,C		CB E1
SET 4,D		CB E2
SET 4,E		CB E3
SET 4,H		CB E4
SET 4,L		CB E5
SET 5,(HL)		CB EE
SET 5,(IX+12H)	I	DD CB 12 EE
SET 5,(IY+12H)	I	FD CB 12 EE
SET 5,A		CB EF
SET 5,B		CB E8
SET 5,C		CB E9
SET 5,D		CB EA
SET 5,E		CB EB
SET 5,H		CB EC
SET 5,L		CB ED
SET 6,(HL)		CB F6
SET 6,(IX+12H)	I	DD CB 12 F6
SET 6,(IY+12H)	I	FD CB 12 F6
SET 6,A		CB F7
SET 6,B		CB F0
SET 6,C		CB F1
SET 6,D		CB F2
SET 6,E		CB F3
SET 6,H		CB F4
SET 6,L		CB F5
SET 7,(HL)		CB FE
SET 7,(IX+12H)	I	DD CB 12 FE
SET 7,(IY+12H)	I	FD CB 12 FE
SET 7,A		CB FF
SET 7,B		CB F8
SET 7,C		CB F9
SET 7,D		CB FA
SET 7,E		CB FB
SET 7,H		CB FC
SET 7,L		CB FD
SETC LCK		ED F7
SETC LW		DD F7
SETC XM		FD F7
SLA (HL)		CB 26
SLA (IX+12H)	I	DD CB 12 26
SLA (IY+12H)	I	FD CB 12 26
SLA A		CB 27
SLA B		CB 20
SLA C		CB 21
SLA D		CB 22
SLA E		CB 23
SLA H		CB 24
SLA L		CB 25
SLAW (HL)		ED CB 22
SLAW (IX+12H)	I	DD CB 12 22
SLAW (IY+12H)	I	FD CB 12 22
SLAW BC		ED CB 20
SLAW DE		ED CB 21

Source Code	Mode	Object Code
SLAW HL		ED CB 23
SLAW IX		ED CB 24
SLAW IY		ED CB 25
SLP		ED 76
SRA (HL)		CB 2E
SRA (IX+12H)	I	DD CB 12 2E
SRA (IY+12H)	I	FD CB 12 2E
SRA A		CB 2F
SRA B		CB 28
SRA C		CB 29
SRA D		CB 2A
SRA E		CB 2B
SRA H		CB 2C
SRA L		CB 2D
SRAW (HL)		ED CB 2A
SRAW (IX+12H)	I	DD CB 12 2A
SRAW (IY+12H)	I	FD CB 12 2A
SRAW BC		ED CB 28
SRAW DE		ED CB 29
SRAW HL		ED CB 2B
SRAW IX		ED CB 2C
SRAW IY		ED CB 2D
SRL (HL)		CB 3E
SRL (IX+12H)	I	DD CB 12 3E
SRL (IY+12H)	I	FD CB 12 3E
SRL A		CB 3F
SRL B		CB 38
SRL C		CB 39
SRL D		CB 3A
SRL E		CB 3B
SRL H		CB 3C
SRL L		CB 3D
SRLW (HL)		ED CB 3A
SRLW (IX+12H)	I	DD CB 12 3A
SRLW (IY+12H)	I	FD CB 12 3A
SRLW BC		ED CB 38
SRLW DE		ED CB 39
SRLW HL		ED CB 3B
SRLW IX		ED CB 3C
SRLW IY		ED CB 3D
SUB A,(HL)		96
SUB A,12H		D6 12
SUB A,A		97
SUB A,(IX+12H)	I	DD 96 12
SUB A,(IY+12H)	I	FD 96 12
SUB 12H		D6 12
SUB A,B		90
SUB A,C		91
SUB A,D		92
SUB A,E		93
SUB A,H		94
SUB A,IXL		DD 95
SUB A,IXU		DD 94
SUB A,IYL		FD 95

Appendix C

APPENDIX C

Z380™ INSTRUCTION IN NUMERIC ORDER

The following Appendix has the Z380 instructions sorted by numeric order.

The column "Mode" indicates whether the instruction is affected by DDIR immediate Decoder Directives, Extended mode or Native mode of operation, and Word or Long Word Mode of operation; "I" means the instruction can be used with DDIR IM to expand its immediate constant, "X" means that the operation of the instruction is affected by the XM status bit, and "L" means that the instruction is affected by LW status bit, or can be used with DDIR LW or DDIR W. The Native/Extended modes, Word/Long Word modes and Decoder Directives are discussed in Chapter 3 in this manual.

Object Code	Source Code		Mode
63	LD	H,E	
64	LD	H,H	
65	LD	H,L	
66	LD	H,(HL)	
67	LD	H,A	
68	LD	L,B	
69	LD	L,C	
6A	LD	L,D	
6B	LD	L,E	
6C	LD	L,H	
6D	LD	L,L	
6E	LD	L,(HL)	
6F	LD	L,A	
70	LD	(HL),B	
71	LD	(HL),C	
72	LD	(HL),D	
73	LD	(HL),E	
74	LD	(HL),H	
75	LD	(HL),L	
76	HALT		
77	LD	(HL),A	
78	LD	A,B	
79	LD	A,C	
7A	LD	A,D	
7B	LD	A,E	
7C	LD	A,H	
7D	LD	A,L	
7E	LD	A,(HL)	
7F	LD	A,A	
80	ADD	A,B	
81	ADD	A,C	
82	ADD	A,D	
83	ADD	A,E	
84	ADD	A,H	
85	ADD	A,L	
86	ADD	A,(HL)	
87	ADD	A,A	
88	ADC	A,B	
89	ADC	A,C	
8A	ADC	A,D	
8B	ADC	A,E	
8C	ADC	A,H	
8D	ADC	A,L	
8E	ADC	A,(HL)	
8F	ADC	A,A	
90	SUB	A,B	
91	SUB	A,C	
92	SUB	A,D	
93	SUB	A,E	
94	SUB	A,H	
95	SUB	A,L	
96	SUB	A,(HL)	
97	SUB	A,A	
98	SBC	A,B	

Object Code	Source Code		Mode
99	SBC	A,C	
9A	SBC	A,D	
9B	SBC	A,E	
9C	SBC	A,H	
9D	SBC	A,L	
9E	SBC	A,(HL)	
9F	SBC	A,A	
A0	AND	A,B	
A0	AND	B	
A1	AND	A,C	
A1	AND	C	
A2	AND	A,D	
A2	AND	D	
A3	AND	A,E	
A3	AND	E	
A4	AND	A,H	
A4	AND	H	
A5	AND	A,L	
A5	AND	L	
A6	AND	(HL)	
A6	AND	A,(HL)	
A7	AND	A	
A7	AND	A,A	
A8	XOR	A,B	
A8	XOR	B	
A9	XOR	A,C	
A9	XOR	C	
AA	XOR	A,D	
AA	XOR	D	
AB	XOR	A,E	
AB	XOR	E	
AC	XOR	A,H	
AC	XOR	H	
AD	XOR	A,L	
AD	XOR	L	
AE	XOR	(HL)	
AE	XOR	A,(HL)	
AF	XOR	A	
AF	XOR	A,A	
B0	OR	A,B	
B0	OR	B	
B1	OR	A,C	
B1	OR	C	
B2	OR	A,D	
B2	OR	D	
B3	OR	A,E	
B3	OR	E	
B4	OR	A,H	
B4	OR	H	
B5	OR	A,L	
B5	OR	L	
B6	OR	(HL)	
B6	OR	A,(HL)	
B7	OR	A	

Object Code	Source Code		Mode
CB 51	BIT	2,C	
CB 52	BIT	2,D	
CB 53	BIT	2,E	
CB 54	BIT	2,H	
CB 55	BIT	2,L	
CB 56	BIT	2,(HL)	
CB 57	BIT	2,A	
CB 58	BIT	3,B	
CB 59	BIT	3,C	
CB 5A	BIT	3,D	
CB 5B	BIT	3,E	
CB 5C	BIT	3,H	
CB 5D	BIT	3,L	
CB 5E	BIT	3,(HL)	
CB 5F	BIT	3,A	
CB 60	BIT	4,B	
CB 61	BIT	4,C	
CB 62	BIT	4,D	
CB 63	BIT	4,E	
CB 64	BIT	4,H	
CB 65	BIT	4,L	
CB 66	BIT	4,(HL)	
CB 67	BIT	4,A	
CB 68	BIT	5,B	
CB 69	BIT	5,C	
CB 6A	BIT	5,D	
CB 6B	BIT	5,E	
CB 6C	BIT	5,H	
CB 6D	BIT	5,L	
CB 6E	BIT	5,(HL)	
CB 6F	BIT	5,A	
CB 70	BIT	6,B	
CB 71	BIT	6,C	
CB 72	BIT	6,D	
CB 73	BIT	6,E	
CB 74	BIT	6,H	
CB 75	BIT	6,L	
CB 76	BIT	6,(HL)	
CB 77	BIT	6,A	
CB 78	BIT	7,B	
CB 79	BIT	7,C	
CB 7A	BIT	7,D	
CB 7B	BIT	7,E	
CB 7C	BIT	7,H	
CB 7D	BIT	7,L	
CB 7E	BIT	7,(HL)	
CB 7F	BIT	7,A	
CB 80	RES	0,B	
CB 81	RES	0,C	
CB 82	RES	0,D	
CB 83	RES	0,E	
CB 84	RES	0,H	
CB 85	RES	0,L	
CB 86	RES	0,(HL)	

Object Code	Source Code		Mode
CB 87	RES	0,A	
CB 88	RES	1,B	
CB 89	RES	1,C	
CB 8A	RES	1,D	
CB 8B	RES	1,E	
CB 8C	RES	1,H	
CB 8D	RES	1,L	
CB 8E	RES	1,(HL)	
CB 8F	RES	1,A	
CB 90	RES	2,B	
CB 91	RES	2,C	
CB 92	RES	2,D	
CB 93	RES	2,E	
CB 94	RES	2,H	
CB 95	RES	2,L	
CB 96	RES	2,(HL)	
CB 97	RES	2,A	
CB 98	RES	3,B	
CB 99	RES	3,C	
CB 9A	RES	3,D	
CB 9B	RES	3,E	
CB 9C	RES	3,H	
CB 9D	RES	3,L	
CB 9E	RES	3,(HL)	
CB 9F	RES	3,A	
CB A0	RES	4,B	
CB A1	RES	4,C	
CB A2	RES	4,D	
CB A3	RES	4,E	
CB A4	RES	4,H	
CB A5	RES	4,L	
CB A6	RES	4,(HL)	
CB A7	RES	4,A	
CB A8	RES	5,B	
CB A9	RES	5,C	
CB AA	RES	5,D	
CB AB	RES	5,E	
CB AC	RES	5,H	
CB AD	RES	5,L	
CB AE	RES	5,(HL)	
CB AF	RES	5,A	
CB B0	RES	6,B	
CB B1	RES	6,C	
CB B2	RES	6,D	
CB B3	RES	6,E	
CB B4	RES	6,H	
CB B5	RES	6,L	
CB B6	RES	6,(HL)	
CB B7	RES	6,A	
CB B8	RES	7,B	
CB B9	RES	7,C	
CB BA	RES	7,D	
CB BB	RES	7,E	
CB BC	RES	7,H	

Object Code	Source Code		Mode
DD 23	INC	IX	X
DD 23	INCW	IX	X
DD 24	INC	IXU	
DD 25	DEC	IXU	
DD 26 12	LD	IXU,12H	
DD 27	LD	IX,IY	L
DD 28 34 12	JR	Z,1234H	X
DD 29	ADD	IX,IX	X
DD 2A 34 12	LD	IX,(1234H)	I L
DD 2B	DEC	IX	X
DD 2B	DECW	IX	X
DD 2C	INC	IXL	
DD 2D	DEC	IXL	
DD 2E 12	LD	IXL,12H	
DD 2F	CPLW	HL	
DD 2F	CPLW		
DD 30 34 12	JR	NC,1234H	X
DD 31	LD	(HL),IX	L
DD 32	LD	HL,DE	L
DD 33	LD	IX,(HL)	L
DD 34 12	INC	(IX+12H)	I
DD 35 12	DEC	(IX+12H)	I
DD 36 12 34	LD	(IX+12H),34H	I
DD 37	LD	IX,HL	L
DD 38 34 12	JR	C,1234H	X
DD 39	ADD	IX,SP	X
DD 3B	LD	HL,IX	L
DD 3C	LD	HL,(BC)	L
DD 3D	LD	HL,(DE)	L
DD 3E	SWAP	IX	
DD 3F	LD	HL,(HL)	L
DD 40	INW	BC,(C)	
DD 41	OUTW	(C),BC	
DD 44	LD	B,IXU	
DD 45	LD	B,IXL	
DD 46 12	LD	B,(IX+12H)	I
DD 47	LD	I,HL	L
DD 47	LDW	I,HL	L
DD 4C	LD	C,IXU	
DD 4D	LD	C,IXL	
DD 4E 12	LD	C,(IX+12H)	I
DD 50	INW	DE,(C)	
DD 51	OUTW	(C),DE	
DD 54	LD	D,IXU	
DD 55	LD	D,IXL	
DD 56 12	LD	D,(IX+12H)	I
DD 57	LD	HL,I	L
DD 57	LDW	HL,I	L
DD 5D	LD	E,IXL	
DD 5D	LD	E,IYL	
DD 5E 12	LD	E,(IX+12H)	I
DD 60	LD	IXU,B	
DD 61	LD	IXU,C	
DD 62	LD	IXU,D	

Object Code	Source Code		Mode
DD 63	LD	IXU,E	
DD 64	LD	IXU,IXU	
DD 65	LD	IXU,IXL	
DD 66 12	LD	H,(IX+12H)	I
DD 67	LD	IXU,A	
DD 68	LD	IXL,B	
DD 69	LD	IXL,C	
DD 6A	LD	IXL,D	
DD 6B	LD	IXL,E	
DD 6C	LD	IXL,IXU	
DD 6D	LD	IXL,IXL	
DD 6E 12	LD	L,(IX+12H)	I
DD 6F	LD	IXL,A	
DD 70 12	LD	(IX+12H),B	I
DD 71 12	LD	(IX+12H),C	I
DD 72 12	LD	(IX+12H),D	I
DD 73 12	LD	(IX+12H),E	I
DD 74 12	LD	(IX+12H),H	I
DD 75 12	LD	(IX+12H),L	I
DD 77 12	LD	(IX+12H),A	I
DD 78	INW	HL,(C)	
DD 79	OUTW	(C),HL	
DD 7C	LD	A,IXU	
DD 7D	LD	A,IXL	
DD 7E 12	LD	A,(IX+12H)	I
DD 84	ADD	A,IXU	
DD 85	ADD	A,IXL	
DD 86 12	ADD	A,(IX+12H)	I
DD 87	ADDW	HL,IX	
DD 87	ADDW	IX	
DD 8C	ADC	A,IXU	
DD 8D	ADC	A,IXL	
DD 8E 12	ADC	A,(IX+12H)	I
DD 8F	ADCW	HL,IX	
DD 8F	ADCW	IX	
DD 94	SUB	A,IXU	
DD 95	SUB	A,IXL	
DD 96 12	SUB	A,(IX+12H)	I
DD 97	SUBW	HL,IX	
DD 97	SUBW	IX	
DD 9C	SBC	A,IXU	
DD 9D	SBC	A,IXL	
DD 9E 12	SBC	A,(IX+12H)	I
DD 9F	SBCW	HL,IX	
DD 9F	SBCW	IX	
DD A4	AND	A,IXU	
DD A4	AND	IXU	
DD A5	AND	A,IXL	
DD A5	AND	IXL	
DD A6 12	AND	(IX+12H)	I
DD A6 12	AND	A,(IX+12H)	I
DD A7	ANDW	HL,IX	
DD A7	ANDW	IX	
DD AC	XOR	A,IXU	

C

Object Code	Source Code		Mode			
DD E3	EX	(SP),IX				L
DD E4 34 12	CALR	PO,1234H			X	
DD E5	PUSH	IX				L
DD E6 12	ANDW	(IX+12H)	I			
DD E6 12	ANDW	HL,(IX+12H)	I			
DD E9	JP	(IX)			X	
DD EC 34 12	CALR	PE,1234H			X	
DD EE 12	XORW	(IX+12H)	I			
DD EE 12	XORW	HL,(IX+12H)	I			
DD F3 1F	DI	1FH				
DD F4 34 12	CALR	P,1234H			X	
DD F6 12	ORW	(IX+12H)	I			
DD F6 12	ORW	HL,(IX+12H)	I			
DD F7	SETC	LW				
DD F9	LD	SP,IX				L
DD FB 1F	EI	1FH				
DD FC 34 12	CALR	M,1234H			X	
DD FE 12	CPW	(IX+12H)	I			
DD FE 12	CPW	HL,(IX+12H)	I			
DD FF	RESC	LW				
DE 12	SBC	A,12H				
DF	RST	18H			X	
E0	RET	NV			X	
E0	RET	PO			X	
E1	POP	HL				L
E2 34 12	JP	NV,1234H	I		X	
E2 34 12	JP	PO,1234H	I		X	
E3	EX	(SP),HL				L
E4 34 12	CALL	NV, 1234H	I		X	
E4 34 12	CALL	PO,1234H	I		X	
E5	PUSH	HL				L
E6 12	AND	12H				
E6 12	AND	A,12H				
E7	RST	20H			X	
E8	RET	PE			X	
E8	RET	V			X	
E9	JP	(HL)			X	
EA 34 12	JP	PE,1234H	I		X	
EA 34 12	JP	V,1234H	I		X	
EB	EX	DE,HL				L
EC 34 12	CALL	V, 1234H	I		X	
EC 34 12	CALL	PE,1234H	I		X	
ED 00 12	IN0	B,(12H)				
ED 01 12	OUT0	(12H),B				
ED 02	LD	BC,BC				L
ED 03	EX	BC,IX				L
ED 04	TST	B				
ED 05	EX	BC,DE				L
ED 06 34 12	LDW	(BC),1234H	I			L
ED 07	EX	A,B				
ED 08 12	IN0	C,(12H)				
ED 09 12	OUT0	(12H),C				
ED 0B	EX	BC,IY				L
ED 0C	TST	C				
ED 0D	EX	BC,HL				L
ED 0E	SWAP	BC				

Object Code	Source Code		Mode			
ED 0F	EX	A,C				
ED 10 12	IN0	D,(12H)				
ED 11 12	OUT0	(12H),D				
ED 12	LD	DE,BC				L
ED 13	EX	DE,IX				L
ED 14	TST	D				
ED 16 34 12	LDW	(DE),1234H	I			L
ED 17	EX	A,D				
ED 18 12	IN0	E,(12H)				
ED 19 12	OUT0	(12H),E				
ED 1B	EX	DE,IY				L
ED 1C	TST	E				
ED 1E	SWAP	DE				
ED 1F	EX	A,E				
ED 20 12	IN0	H,(12H)				
ED 21 12	OUT0	(12H),H				
ED 24	TST	H				
ED 27	EX	A,H				
ED 28 12	IN0	L,(12H)				
ED 29 12	OUT0	(12H),L				
ED 2B	EX	IX,IY				L
ED 2C	TST	L				
ED 2F	EX	A,L				
ED 30 12	IN0	(12H)				
ED 32	LD	HL,BC				L
ED 33	EX	HL,IX				L
ED 34	TST	(HL)				
ED 36 34 12	LDW	(HL),1234H	I			L
ED 37	EX	A,(HL)				
ED 38 12	IN0	A,(12H)				
ED 39 12	OUT0	(12H),A				
ED 3B	EX	HL,IY				L
ED 3C	TST	A				
ED 3E	SWAP	HL				
ED 3F	EX	A,A				
ED 40	IN	B,(C)				
ED 41	OUT	(C),B				
ED 42	SBC	HL,BC				
ED 43 34 12	LD	(1234H),BC	I			L
ED 44	NEG	A				
ED 44	NEG					
ED 45	RETN				X	
ED 46	IM	0				
ED 47	LD	I,A				
ED 48	IN	C,(C)				
ED 49	OUT	(C),C				
ED 4A	ADC	HL,BC				
ED 4B 34 12	LD	BC,(1234H)	I			L
ED 4C	MLT	BC				
ED 4D	RETI				X	
ED 4E	IM	3				
ED 4F	LD	R,A				
ED 50	IN	D,(C)				
ED 51	OUT	(C),D				

C

Object Code	Source Code		Mode			
ED B5	ORW	DE				
ED B5	ORW	HL,DE				
ED B6 34 12	ORW	1234H				
ED B6 34 12	ORW	HL,1234H				
ED B7	ORW	HL				
ED B7	ORW	HL,HL				
ED B8	LDDR					
ED B9	CPDR				X	
ED BA	INDR					
ED BB	OTDR					
ED BC	CPW	BC				
ED BC	CPW	HL,BC				
ED BD	CPW	DE				
ED BD	CPW	HL,DE				
ED BE 34 12	CPW	1234H				
ED BE 34 12	CPW	HL,1234H				
ED BF	CPW	HL				
ED BF	CPW	HL,HL				
ED C0	LDCTL	HL,SR				L
ED C1	POP	SR				L
ED C4 12	CALR	NZ,12H			X	
ED C5	PUSH	SR				L
ED C6 34 12	ADD	HL,(1234H)	I		X	
ED C8	LDCTL	SR,HL				L
ED CB 00	RLCW	BC				
ED CB 01	RLCW	DE				
ED CB 02	RLCW	(HL)				
ED CB 03	RLCW	HL				
ED CB 04	RLCW	IX				
ED CB 05	RLCW	IY				
ED CB 08	RRCW	BC				
ED CB 09	RRCW	DE				
ED CB 0A	RRCW	(HL)				
ED CB 0B	RRCW	HL				
ED CB 0C	RRCW	IX				
ED CB 0D	RRCW	IY				
ED CB 10	RLW	BC				
ED CB 11	RLW	DE				
ED CB 12	RLW	(HL)				
ED CB 13	RLW	HL				
ED CB 14	RLW	IX				
ED CB 15	RLW	IY				
ED CB 18	RRW	BC				
ED CB 19	RRW	DE				
ED CB 1A	RRW	(HL)				
ED CB 1B	RRW	HL				
ED CB 1C	RRW	IX				
ED CB 1D	RRW	IY				
ED CB 20	SLAW	BC				
ED CB 21	SLAW	DE				
ED CB 22	SLAW	(HL)				
ED CB 23	SLAW	HL				
ED CB 24	SLAW	IX				
ED CB 25	SLAW	IY				

Object Code	Source Code		Mode
ED CB 28	SRAW	BC	
ED CB 29	SRAW	DE	
ED CB 2A	SRAW	(HL)	
ED CB 2B	SRAW	HL	
ED CB 2C	SRAW	IX	
ED CB 2D	SRAW	IY	
ED CB 30	EX	BC,BC'	L
ED CB 31	EX	DE,DE'	L
ED CB 33	EX	HL,HL'	L
ED CB 34	EX	IX,IX'	
ED CB 35	EX	IY,IY'	L
ED CB 38	SRLW	BC	
ED CB 39	SRLW	DE	
ED CB 3A	SRLW	(HL)	
ED CB 3B	SRLW	HL	
ED CB 3C	SRLW	IX	
ED CB 3D	SRLW	IY	
ED CB 90	MULTW	BC	
ED CB 90	MULTW	HL,BC	
ED CB 91	MULTW	DE	
ED CB 91	MULTW	HL,DE	
ED CB 93	MULTW	HL	
ED CB 93	MULTW	HL,HL	
ED CB 94	MULTW	HL,IX	
ED CB 94	MULTW	IX	
ED CB 95	MULTW	HL,IY	
ED CB 95	MULTW	IY	
ED CB 97 34 12	MULTW	1234H	
ED CB 97 34 12	MULTW	HL,1234H	
ED CB 98	MULTUW		BC
ED CB 98	MULTUW		HL,BC
ED CB 99	MULTUW		DE
ED CB 99	MULTUW		HL,DE
ED CB 9B	MULTUW		HL
ED CB 9B	MULTUW		HL,HL
ED CB 9C	MULTUW		HL,IX
ED CB 9C	MULTUW		IX
ED CB 9D	MULTUW		HL,IY
ED CB 9D	MULTUW		IY
ED CB 9F	MULTUW		1234H
ED CB 9F	MULTUW		HL,1234H
ED CB B8	DIVUW	BC	
ED CB B8	DIVUW	HL,BC	
ED CB B9	DIVUW	DE	
ED CB B9	DIVUW	HL,DE	
ED CB BB	DIVUW	HL	
ED CB BB	DIVUW	HL,HL	

C

Object Code	Source Code		Mode	
FD 3F	LD	(HL),HL		L
FD 44	LD	B,IYU		
FD 45	LD	B,IYL		
FD 46 12	LD	B,(IY+12H)	I	
FD 4C	LD	C,IYU		
FD 4D	LD	C,IYL		
FD 4E 12	LD	C,(IY+12H)	I	
FD 54	LD	D,IYU		
FD 55	LD	D,IYL		
FD 56 12	LD	D,(IY+12H)	I	
FD 5C	LD	E,IYU		
FD 5D	LD	E,IYL		
FD 5E 12	LD	E,(IY+12H)	I	
FD 60	LD	IYU,B		
FD 61	LD	IYU,C		
FD 62	LD	IYU,D		
FD 63	LD	IYU,E		
FD 64	LD	IYU,IYU		
FD 65	LD	IYU,IYL		
FD 66 12	LD	H,(IY+12H)	I	
FD 67	LD	IYU,A		
FD 68	LD	IYL,B		
FD 69	LD	IYL,C		
FD 6A	LD	IYL,D		
FD 6B	LD	IYL,E		
FD 6C	LD	IYL,IYU		
FD 6D	LD	IYL,IYL		
FD 6E 12	LD	L,(IY+12H)	I	
FD 6F	LD	IYL,A		
FD 70 12	LD	(IY+12H),B	I	
FD 71 12	LD	(IY+12H),C	I	
FD 72 12	LD	(IY+12H),D	I	
FD 73 12	LD	(IY+12H),E	I	L
FD 74 12	LD	(IY+12H),H	I	
FD 75 12	LD	(IY+12H),L	I	
FD 77 12	LD	(IY+12H),A	I	
FD 79 34 12	OUTW	(C),1234H		
FD 7C	LD	A,IYU		
FD 7D	LD	A,IYL		
FD 7E 12	LD	A,(IY+12H)	I	
FD 84	ADD	A,IYU		
FD 85	ADD	A,IYL		
FD 86 12	ADD	A,(IY+12H)	I	
FD 87	ADDW	HL,IY		
FD 87	ADDW	IY		
FD 8C	ADC	A,IYU		
FD 8D	ADC	A,IYL		
FD 8E 12	ADC	A,(IY+12H)	I	
FD 8F	ADCW	HL,IY		
FD 8F	ADCW	IY		
FD 94	SUB	A,IYU		
FD 95	SUB	A,IYL		
FD 96 12	SUB	A,(IY+12H)	I	
FD 97	SUBW	HL,IY		

Object Code	Source Code		Mode	
FD 97	SUBW	IY		
FD 9C	SBC	A,IYU		
FD 9D	SBC	A,IYL		
FD 9E 12	SBC	A,(IY+12H)	I	
FD 9F	SBCW	HL,IY		
FD 9F	SBCW	IY		
FD A4	AND	A,IYU		
FD A4	AND	IYU		
FD A5	AND	A,IYL		
FD A5	AND	IYL		
FD A6 12	AND	(IY+12H)	I	
FD A6 12	AND	A,(IY+12H)	I	
FD A7	ANDW	HL,IY		
FD A7	ANDW	IY		
FD AC	XOR	A,IYU		
FD AC	XOR	IYU		
FD AD	XOR	A,IYL		
FD AD	XOR	IYL		
FD AE 12	XOR	(IY+12H)	I	
FD AE 12	XOR	A,(IY+12H)	I	
FD AF	XORW	HL,IY		
FD AF	XORW	IY		
FD B4	OR	A,IYU		
FD B4	OR	IYU		
FD B5	OR	A,IYL		
FD B5	OR	IYL		
FD B6 12	OR	(IY+12H)	I	
FD B6 12	OR	A,(IY+12H)	I	
FD B7	ORW	HL,IY		
FD B7	ORW	IY		
FD BC	CP	A,IYU		
FD BC	CP	IYU		
FD BD	CP	A,IYL		
FD BD	CP	IYL		
FD BE 12	CP	(IY+12H)	I	
FD BE 12	CP	A,(IY+12H)	I	
FD BF	CPW	HL,IY		
FD BF	CPW	IY		
FD C0	DDIR	LW		
FD C1	DDIR	IB,LW		
FD C2	DDIR	IW,LW		
FD C3	DDIR	IW		
FD C4 56 34 12	CALR	NZ,123456H		X
FD C6 12	ADDW	(IY+12H)	I	
FD C6 12	ADDW	HL,(IY+12H)	I	
FD CB 12 02	RLCW	(IY+12H)	I	
FD CB 12 03	LD	BC,(IY+12H)	I	L
FD CB 12 06	RLC	(IY+12H)	I	
FD CB 12 0A	RRCW	(IY+12H)	I	
FD CB 12 0B	LD	(IY+12H),BC	I	L
FD CB 12 0E	RRC	(IY+12H)	I	
FD CB 12 12	RLW	(IY+12H)	I	
FD CB 12 13	LD	DE,(IY+12H)	I	L
FD CB 12 16	RL	(IY+12H)	I	

C

ZiLOG

Appendix D

APPENDIX D

INSTRUCTIONS AFFECTED BY NORMAL/ EXTENDED MODE, AND LONG WORD MODE

This Appendix has two sets of tables. Each table is a subset of the Table in the Appendix B. The Table D-1 has the instructions which works differently in the Native and Extended mode of operation, and the Table D-2 has the instructions which works differently in Word/Long Word mode of operation.

Source Code		Object Code		
RETN		ED	45	
RST	00H	C7		
RST	08H	CF		
RST	10H	D7		
RST	18H	DF		
RST	20H	E7		
RST	28H	EF		
RST	30H	F7		
RST	38H	FF		

Table D-2. Instructions operates different in Long Word Modes.

Source Code		Object Code		
EX	(SP),HL	E3		
EX	(SP),IX	DD	E3	
EX	(SP),IY	FD	E3	
EX	BC,BC'	ED	CB	30
EX	BC,DE	ED	05	
EX	BC,HL	ED	0D	
EX	BC,IX	ED	03	
EX	BC,IY	ED	0B	
EX	DE,DE'	ED	CB	31
EX	DE,HL	EB		
EX	DE,IX	ED	13	
EX	DE,IY	ED	1B	
EX	HL,HL'	ED	CB	33
EX	HL,IX	ED	33	
EX	HL,IY	ED	3B	
EX	IX,IX'	ED	CB	34
EX	IX,IY	ED	2B	
EX	IY,IY'	ED	CB	35
EXTS	A	ED	65	
EXTS		ED	65	
LD	(BC),BC	FD	0C	
LD	(BC),DE	FD	1C	
LD	(BC),HL	FD	3C	
LD	(BC),IX	DD	01	
LD	(BC),IY	FD	01	
LD	(DE),BC	FD	0D	
LD	(DE),DE	FD	1D	
LD	(DE),HL	FD	3D	
LD	(DE),IX	DD	11	
LD	(DE),IY	FD	11	
LD	(HL),BC	FD	0F	
LD	(HL),DE	FD	1F	
LD	(HL),HL	FD	3F	
LD	(HL),IX	DD	31	
LD	(HL),IY	FD	31	
LD	BC,(BC)	DD	0C	
LD	BC,(DE)	DD	0D	
LD	BC,(HL)	DD	0F	
LD	BC,BC	ED	02	

Source Code		Object Code	
LD	BC,DE	DD	02
LD	BC,HL	FD	02
LD	BC,IX	DD	0B
LD	BC,IY	FD	0B
LD	DE,(BC)	DD	1C
LD	DE,(DE)	DD	1D
LD	DE,(HL)	DD	1F
LD	DE,BC	ED	12
LD	DE,DE	DD	12
LD	DE,HL	FD	12
LD	DE,IX	DD	1B
LD	DE,IY	FD	1B
LD	HL,(BC)	DD	3C
LD	HL,(DE)	DD	3D
LD	HL,(HL)	DD	3F
LD	HL,BC	ED	32
LD	HL,DE	DD	32
LD	HL,HL	FD	32
LD	HL,I	DD	57
LD	HL,IX	DD	3B
LD	HL,IY	FD	3B
LD	I,HL	DD	47
LD	IX,(BC)	DD	03
LD	IX,(DE)	DD	13
LD	IX,(HL)	DD	33
LD	IX,BC	DD	07
LD	IX,DE	DD	17
LD	IX,HL	DD	37
LD	IX,IY	DD	27
LD	IY,(BC)	FD	03
LD	IY,(DE)	FD	13
LD	IY,(HL)	FD	33
LD	IY,BC	FD	07
LD	IY,DE	FD	17
LD	IY,HL	FD	37
LD	IY,IX	FD	27
LD	SP,HL	F9	
LD	SP,IX	DD	F9
LD	SP,IY	FD	F9

ZILOG

APPENDIX E

INSTRUCTIONS AFFECTED BY DDIR IM INSTRUCTIONS

This Appendix has instructions which can be used with the Decoder Directive(s) Extend Immediate. There are eight tables (E1-E8) which are the subset of the Table A, sorted by the category of the instruction.

Note that the instructions listed here does not have the DDIR Decoder Directive in front of the instructions listed below, and notation used here may be different by the assembler to be used.

Table E-1. Valid with DDIR IB in Extended mode. LW bit status does not affect the operation

ADD	HL,(123456H)	ED	C6	56	34	12
ADD	SP,123456H	ED	82	56	34	12
CALL	123456H	CD	56	34	12	
CALL	C,123456H	DC	56	34	12	
CALL	M,123456H	FC	56	34	12	
CALL	NC,123456H	D4	56	34	12	
CALL	NZ,123456H	C4	56	34	12	
CALL	P,123456H	F4	56	34	12	
CALL	PE,123456H	EC	56	34	12	
CALL	PO,123456H	E4	56	34	12	
CALL	Z,123456H	CC	56	34	12	
JP	123456H	C3	56	34	12	
JP	C,123456H	DA	56	34	12	
JP	M,123456H	FA	56	34	12	
JP	NC,123456H	D2	56	34	12	
JP	NS,123456H	F2	56	34	12	
JP	NV,123456H	E2	56	34	12	
JP	NZ,123456H	C2	56	34	12	
JP	P,123456H	F2	56	34	12	
JP	PE,123456H	EA	56	34	12	
JP	PO,123456H	E2	56	34	12	
JP	S,123456H	FA	56	34	12	
JP	V,123456H	EA	56	34	12	
JP	Z,123456H	CA	56	34	12	
SUB	HL,(123456H)	ED	D6	56	34	12
SUB	SP,123456H	ED	92	56	34	12

Table E-2. Valid with DDIR IB. XM bit status does not affect the operation. Transfer size determined by LW bit. (Either with DDIR IB, DDIR IB,LW or DDIR IB,W)

LD	(123456H),BC	ED	43	56	34	12
LD	(123456H),DE	ED	53	56	34	12
LD	(123456H),HL	22	56	34	12	
LD	(123456H),HL	ED	63	56	34	12
LD	(123456H),IX	DD	22	56	34	12
LD	(123456H),IY	FD	22	56	34	12
LD	(123456H),SP	ED	73	56	34	12
LD	(IX+1234H),BC	DD	CB	34	12	0B
LD	(IX+1234H),DE	DD	CB	34	12	1B
LD	(IX+1234H),HL	DD	CB	34	12	3B
LD	(IX+1234H),IY	DD	CB	34	12	2B
LD	(IY+1234H),BC	FD	CB	34	12	0B
LD	(IY+1234H),E	FD	73	34	12	
LD	(IY+1234H),HL	FD	CB	34	12	3B
LD	(IY+1234H),IX	FD	CB	34	12	2B
LD	(SP+1234H),BC	DD	CB	34	12	09
LD	(SP+1234H),DE	DD	CB	34	12	19
LD	(SP+1234H),HL	DD	CB	34	12	39
LD	(SP+1234H),IX	DD	CB	34	12	29
LD	(SP+1234H),IY	FD	CB	34	12	29
LD	BC,(123456H)	ED	4B	56	34	12
LD	BC,(IX+1234H)	DD	CB	34	12	03
LD	BC,(IY+1234H)	FD	CB	34	12	03
LD	BC,(SP+1234H)	DD	CB	34	12	01
LD	DE,(123456H)	ED	5B	56	34	12
LD	DE,(IX+1234H)	DD	CB	34	12	13
LD	DE,(IY+1234H)	FD	CB	34	12	13
LD	DE,(SP+1234H)	DD	CB	34	12	11
LD	HL,(123456H)	2A	56	34	12	
LD	HL,(123456H)	ED	6B	56	34	12
LD	HL,(IX+1234H)	DD	CB	34	12	33
LD	HL,(IY+1234H)	FD	CB	34	12	33
LD	HL,(SP+1234H)	DD	CB	34	12	31
LD	IX,(123456H)	DD	2A	56	34	12
LD	IX,(IY+1234H)	FD	CB	34	12	23
LD	IX,(SP+1234H)	DD	CB	34	12	21
LD	IY,(123456H)	FD	2A	56	34	12
LD	IY,(IX+1234H)	DD	CB	34	12	23
LD	IY,(SP+1234H)	FD	CB	34	12	21
LD	SP,(123456H)	ED	7B	56	34	12
LDW	(BC),123456H	ED	06	56	34	12
LDW	(DE),123456H	ED	16	56	34	12
LDW	(HL),123456H	ED	36	56	34	12

OR	(IY+1234H)	FD	B6	34	12	
OR	A,(IX+1234H)	DD	B6	34	12	
OR	A,(IY+1234H)	FD	B6	34	12	
ORW	(IX+1234H)	DD	F6	34	12	
ORW	(IY+1234H)	FD	F6	34	12	
ORW	HL,(IX+1234H)	DD	F6	34	12	
ORW	HL,(IY+1234H)	FD	F6	34	12	
OUTA	(123456H),A	ED	D3	56	34	12
OUTAW	(123456H),HL	FD	D3	56	34	12
RES	0,(IX+1234H)	DD	CB	34	12	86
RES	0,(IY+1234H)	FD	CB	34	12	86
RES	1,(IX+1234H)	DD	CB	34	12	8E
RES	1,(IY+1234H)	FD	CB	34	12	8E
RES	2,(IX+1234H)	DD	CB	34	12	96
RES	2,(IY+1234H)	FD	CB	34	12	96
RES	3,(IX+1234H)	DD	CB	34	12	9E
RES	3,(IY+1234H)	FD	CB	34	12	9E
RES	4,(IX+1234H)	DD	CB	34	12	A6
RES	4,(IY+1234H)	FD	CB	34	12	A6
RES	5,(IX+1234H)	DD	CB	34	12	AE
RES	5,(IY+1234H)	FD	CB	34	12	AE
RES	6,(IX+1234H)	DD	CB	34	12	B6
RES	6,(IY+1234H)	FD	CB	34	12	B6
RES	7,(IX+1234H)	DD	CB	34	12	BE
RES	7,(IY+1234H)	FD	CB	34	12	BE
RL	(IX+1234H)	DD	CB	34	12	16
RL	(IY+1234H)	FD	CB	34	12	16
RLC	(IX+1234H)	DD	CB	34	12	06
RLC	(IY+1234H)	FD	CB	34	12	06
RLCW	(IX+1234H)	DD	CB	34	12	02
RLCW	(IY+1234H)	FD	CB	34	12	02
RLW	(IX+1234H)	DD	CB	34	12	12
RLW	(IY+1234H)	FD	CB	34	12	12
RR	(IX+1234H)	DD	CB	34	12	1E
RR	(IY+1234H)	FD	CB	34	12	1E
RRC	(IX+1234H)	DD	CB	34	12	0E
RRC	(IY+1234H)	FD	CB	34	12	0E
RRCW	(IX+1234H)	DD	CB	34	12	0A
RRCW	(IY+1234H)	FD	CB	34	12	0A
RRW	(IX+1234H)	DD	CB	34	12	1A
RRW	(IY+1234H)	FD	CB	34	12	1A
SBC	A,(IX+1234H)	DD	9E	34	12	
SBC	A,(IY+1234H)	FD	9E	34	12	
SBCW	(IX+1234H)	DD	DE	34	12	
SBCW	(IY+1234H)	FD	DE	34	12	
SET	0,(IX+1234H)	DD	CB	34	12	C6
SET	0,(IY+1234H)	FD	CB	34	12	C6
SET	1,(IX+1234H)	DD	CB	34	12	CE
SET	1,(IY+1234H)	FD	CB	34	12	CE
SET	2,(IX+1234H)	DD	CB	34	12	D6
SET	2,(IY+1234H)	FD	CB	34	12	D6
SET	3,(IX+1234H)	DD	CB	34	12	DE
SET	3,(IY+1234H)	FD	CB	34	12	DE
SET	4,(IX+1234H)	DD	CB	34	12	E6
SET	4,(IY+1234H)	FD	CB	34	12	E6
SET	5,(IX+1234H)	DD	CB	34	12	EE
SET	5,(IY+1234H)	FD	CB	34	12	EE
SET	6,(IX+1234H)	DD	CB	34	12	F6
SET	6,(IY+1234H)	FD	CB	34	12	F6
SET	7,(IX+1234H)	DD	CB	34	12	FE
SET	7,(IY+1234H)	FD	CB	34	12	FE
SLA	(IX+1234H)	DD	CB	34	12	26
SLA	(IY+1234H)	FD	CB	34	12	26
SLAW	(IX+1234H)	DD	CB	34	12	22
SLAW	(IY+1234H)	FD	CB	34	12	22
SRA	(IX+1234H)	DD	CB	34	12	2E
SRA	(IY+1234H)	FD	CB	34	12	2E
SRAW	(IX+1234H)	DD	CB	34	12	2A
SRAW	(IY+1234H)	FD	CB	34	12	2A
SRL	(IX+1234H)	DD	CB	34	12	3E
SRL	(IY+1234H)	FD	CB	34	12	3E
SRLW	(IX+1234H)	DD	CB	34	12	3A
SRLW	(IY+1234H)	FD	CB	34	12	3A
SUB	A,(IX+1234H)	DD	96	34	12	
SUB	A,(IY+1234H)	FD	96	34	12	
SUBW	HL,(IX+1234H)	DD	D6	34	12	
SUBW	HL,(IY+1234H)	FD	D6	34	12	
XOR	(IX+1234H)	DD	AE	34	12	
XOR	(IY+1234H)	FD	AE	34	12	
XOR	A,(IX+1234H)	DD	AE	34	12	
XOR	A,(IY+1234H)	FD	AE	34	12	
XORW	(IX+1234H)	DD	EE	34	12	
XORW	(IY+1234H)	FD	EE	34	12	
XORW	HL,(IX+1234H)	DD	EE	34	12	
XORW	HL,(IY+1234H)	FD	EE	34	12	

Table E-7. Valid with DDIR IW in Long Word mode. XM bit status does not affect the operation. (Either with DDIR IW,LW or DDIR IW with LW bit set.)

LD	BC,12345678H	01	78	56	34	12	
LD	DE,12345678H	11	78	56	34	12	
LD	HL,12345678H	21	78	56	34	12	
LD	IX,12345678H	DD	21	78	56	34	12
LD	IY,12345678H	FD	21	78	56	34	12
LD	SP,12345678H	31	78	56	34	12	
PUSH	12345678H	FD	F5	78	56	34	12

Table E-8. Valid with DDIR IW. XM bit nor LW bit status do not affect the operation

ADC	A,(IX+123456H)	DD	8E	56	34	12	
ADC	A,(IY+123456H)	FD	8E	56	34	12	
ADCW	(IX+123456H)	DD	CE	56	34	12	
ADCW	(IY+123456H)	FD	CE	56	34	12	
ADCW	HL,(IX+123456H)	DD	CE	56	34	12	
ADCW	HL,(IY+123456H)	FD	CE	56	34	12	
ADD	A,(IX+123456H)	DD	86	56	34	12	
ADD	A,(IY+123456H)	FD	86	56	34	12	
ADDW	(IX+123456H)	DD	C6	56	34	12	
ADDW	(IY+123456H)	FD	C6	56	34	12	
ADDW	HL,(IX+123456H)	DD	C6	56	34	12	
ADDW	HL,(IY+123456H)	FD	C6	56	34	12	
AND	(IX+123456H)	DD	A6	56	34	12	
AND	(IY+123456H)	FD	A6	56	34	12	
AND	A,(IX+123456H)	DD	A6	56	34	12	
AND	A,(IY+123456H)	FD	A6	56	34	12	
ANDW	(IX+123456H)	DD	E6	56	34	12	
ANDW	(IY+123456H)	FD	E6	56	34	12	
ANDW	HL,(IX+123456H)	DD	E6	56	34	12	
ANDW	HL,(IY+123456H)	FD	E6	56	34	12	
BIT	0,(IX+123456H)	DD	CB	56	34	12	46
BIT	0,(IY+123456H)	FD	CB	56	34	12	46
BIT	1,(IX+123456H)	DD	CB	56	34	12	4E
BIT	1,(IY+123456H)	FD	CB	56	34	12	4E
BIT	2,(IX+123456H)	DD	CB	56	34	12	56
BIT	2,(IY+123456H)	FD	CB	56	34	12	56
BIT	3,(IX+123456H)	DD	CB	56	34	12	5E
BIT	3,(IY+123456H)	FD	CB	56	34	12	5E
BIT	4,(IX+123456H)	DD	CB	56	34	12	66
BIT	4,(IY+123456H)	FD	CB	56	34	12	66
BIT	5,(IX+123456H)	DD	CB	56	34	12	6E
BIT	5,(IY+123456H)	FD	CB	56	34	12	6E
BIT	6,(IX+123456H)	DD	CB	56	34	12	76
BIT	6,(IY+123456H)	FD	CB	56	34	12	76
BIT	7,(IX+123456H)	DD	CB	56	34	12	7E
BIT	7,(IY+123456H)	FD	CB	56	34	12	7E
CP	(IX+123456H)	DD	BE	56	34	12	
CP	(IY+123456H)	FD	BE	56	34	12	
CP	A,(IX+123456H)	DD	BE	56	34	12	
CP	A,(IY+123456H)	FD	BE	56	34	12	
CPW	(IX+123456H)	DD	FE	56	34	12	
CPW	(IY+123456H)	FD	FE	56	34	12	
CPW	HL,(IX+123456H)	DD	FE	56	34	12	
CPW	HL,(IY+123456H)	FD	FE	56	34	12	
DEC	(IX+123456H)	DD	35	56	34	12	
DEC	(IY+123456H)	FD	35	56	34	12	
DIVUW	(IX+123456H)	DD	CB	56	34	12	BA
DIVUW	(IY+123456H)	FD	CB	56	34	12	BA
DIVUW	HL,(IX+123456H)	DD	CB	56	34	12	BA
DIVUW	HL,(IY+123456H)	FD	CB	56	34	12	BA
INA	A,(123456H)	ED	DB	56	34	12	
INAW	HL,(123456H)	FD	DB	56	34	12	
INC	(IX+123456H)	DD	56	34	12		
INC	(IY+123456H)	FD	56	34	12		
LD	(12345678H),A	32	78	56	34	12	
LD	(IX+123456H),56H	DD	36	56	34	12	56
LD	(IX+123456H),A	DD	77	56	34	12	
LD	(IX+123456H),B	DD	70	56	34	12	
LD	(IX+123456H),C	DD	71	56	34	12	
LD	(IX+123456H),D	DD	72	56	34	12	
LD	(IX+123456H),E	DD	73	56	34	12	
LD	(IX+123456H),H	DD	74	56	34	12	
LD	(IX+123456H),L	DD	75	56	34	12	
LD	(IY+123456H),78H	FD	36	56	34	12	78
LD	(IY+123456H),A	FD	77	56	34	12	
LD	(IY+123456H),B	FD	70	56	34	12	
LD	(IY+123456H),C	FD	71	56	34	12	
LD	(IY+123456H),D	FD	72	56	34	12	
LD	(IY+123456H),DE	FD	CB	56	34	12	1B
LD	(IY+123456H),H	FD	74	56	34	12	
LD	(IY+123456H),L	FD	75	56	34	12	
LD	A,(12345678H)	3A	78	56	34	12	
LD	A,(IX+123456H)	DD	7E	56	34	12	
LD	A,(IY+123456H)	FD	7E	56	34	12	
LD	B,(IX+123456H)	DD	46	56	34	12	
LD	B,(IY+123456H)	FD	46	56	34	12	
LD	C,(IX+123456H)	DD	4E	56	34	12	
LD	C,(IY+123456H)	FD	4E	56	34	12	
LD	D,(IX+123456H)	DD	56	56	34	12	
LD	D,(IY+123456H)	FD	56	56	34	12	
LD	E,(IX+123456H)	DD	5E	56	34	12	
LD	E,(IY+123456H)	FD	5E	56	34	12	
LD	H,(IX+123456H)	DD	66	56	34	12	
LD	H,(IY+123456H)	FD	66	56	34	12	
LD	L,(IX+123456H)	DD	6E	56	34	12	
LD	L,(IY+123456H)	FD	6E	56	34	12	
MULTUW	(IX+123456H)	DD	CB	56	34	12	9A
MULTUW	(IY+123456H)	FD	CB	56	34	12	9A
MULTUW	HL,(IX+123456H)	DD	CB	56	34	12	9A
MULTUW	HL,(IY+123456H)	FD	CB	56	34	12	9A
MULTW	(IX+123456H)	DD	CB	56	34	12	92
MULTW	(IY+123456H)	FD	CB	56	34	12	92
MULTW	HL,(IX+123456H)	DD	CB	56	34	12	92
MULTW	HL,(IY+123456H)	FD	CB	56	34	12	92
OR	(IX+123456H)	DD	B6	56	34	12	
OR	(IY+123456H)	FD	B6	56	34	12	

E

ZILOG

INDEX

Symbols

A

B

C

D

J

L

M

N

O

T

U

V

W

Z

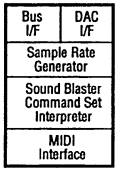
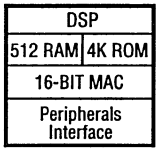
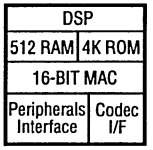
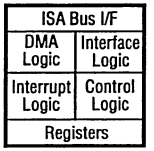
I

ZILOG

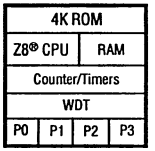
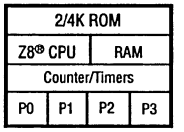
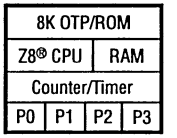
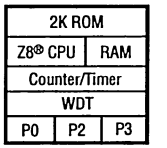
BLOCK DIAGRAM	ROM UART 8611 / CPU COUNTER/TIMERS / RAM P0 P1 P2 P3	ROM CPU WDT 236 RAM P1 P2 P3 P0	Z8 DSP 24K ROM 4K ROM A/D D/A 31 or 47 DIGITAL I/O	Z8 DSP 24K ROM 6K ROM A/D D/A 31 or 47 DIGITAL I/O
PART NUMBER	**Z8600/Z8611**	**Z86C30/E30/C31/E31**	**Z89C65/Z89C66**	**Z89165/Z89166**
DESCRIPTION	Z8® NMOS (CCP™) Z8600 = 2K ROM Z8611 = 4K ROM	Z8® Consumer Controller Processor (CCP™) Z86C30 = 28-Pin, 4K ROM Z86C31 = 28-Pin, 2K ROM Z86C40 = 40-Pin, 4K ROM Z86E30, Z86E31, Z86E40 = OTP Version	Telephone Answering Controller Z89C66 = ROMLess with 31 I/O Pins	Low-Cost DTAD Controller Z89166 = ROMLess with 31 I/O Pins
PROCESS/SPEED	NMOS: 8,12 MHz	CMOS: 12 MHz	CMOS: 20 MHz	CMOS: 20 MHz
FEATURES	■ 2K/4K ROM ■ 128 Bytes RAM ■ 22/32 I/O Lines ■ On-Chip Oscillator ■ Two Counter/Timers ■ Six Vectored, Priority Interrupts ■ UART (Z8611 Only)	■ 4K ROM/236 RAM ■ Two Standby Modes ■ Two Counter/Timers ■ ROM/RAM Protect ■ Four Ports (Z86C40/E40) ■ Three Ports (Z86C30/E30/C31/E31) ■ Low-Voltage Protection ■ Two Analog Comparators ■ Low-EMI Option ■ Watch-Dog Timer (WDT) ■ Auto Power-On Reset ■ Low-Power Option	■ 24K ROM (Z89C65 Only) ■ 16-Bit DSP ■ 4K Word ROM ■ 8-Bit A/D with Automatic Gain Control (AGC) ■ DTMF Macro Available ■ LPC Macro Available ■ 10-Bit PWM D/A ■ Other DSP Software Options Available ■ 47 I/O Pins (Z89C65 Only)	■ 24K ROM (Z89165 Only) ■ 16-Bit DSP ■ 6K Word DSP ROM ■ 8-Bit A/D with Automatic Gain Control (AGC) ■ DTMF Macro Available ■ LPC Macro Available ■ 10-Bit PWM D/A ■ Other DSP Software Options Available ■ 47 I/O Pins (Z89165 Only)
PACKAGE	28-Pin DIP 40-Pin DIP 44-Pin PLCC	28-Pin DIP 40-Pin DIP 44-Pin PLCC, QFP	68-Pin PLCC	68-Pin PLCC 80-Pin QFP
SUPPORT PRODUCTS	Z86C1200ZEM - Emulator Z0860000ZCO - Evaluation Board Z0860000ZDP - Adaptor Kit	Z86CCP00ZEM - Emulator Z86CCP00ZAC - Emulator Z86C5000ZEM - Emulator Z86E3000ZDP - Adaptor Kit Z86E4000ZDP - Program Adaptor Kit	Z89C6501ZEM - Emulator Z89C6500ZDB - Emulator	Z89C6501ZEM - Emulator Z89C6500ZDB - Emulator Z8916500ZCO - Evaluation Board

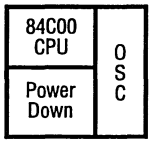
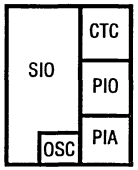
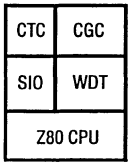
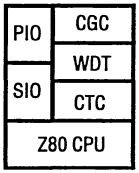
BLOCK DIAGRAM	16/8K ROM 4K CHAR ROM Z8 CPU | RAM OSD 13 PWM | TIMER WDT | 5 PORTS	6K ROM 3K CHAR ROM Z8 CPU | RAM OSD 7 PWM | TIMER WDT | 3 PORTS	CHAR ROM COMMAND INTERPRETER ANALOG SYNC/DATA SLICER | OSD CTRL	1K/6K ROM Z8 CPU WDT | 124 RAM P2 | P3	2K/8K/16K ROM Z8 CPU WDT | 128,256, 768 RAM P0 | P1 | P2 | P3
PART NUMBER	**Z86C27/127/97/47/E47**	**Z86227**	**Z86128/Z86228/Z86129**	**Z86L06/Z86L29**	**Z86L70/71/72/73/74 75/76/77/78**
DESCRIPTION	Digital Television Controller (DTC™) Television, VCRs, and Cable Z86E47 = OTP Version	Standard DTC™ Features with Reduced ROM, RAM, PWM Outputs for Greater Economy	Z86128/228 = Line 21 Closed Caption Controller (L21C™) Z86129/228 = Line 21 Closed Caption and EDS Controller	Z86L06 = Low-Voltage CMOS Consumer Controller Processor Z86L29 = 6K Infrared Remote Controller	Zilog Infrared Remote Controllers (ZIRC™) for IR Remote/Battery Operated Applications Ranging in ROM: L70=2K, L71=8K, L72&78=16K, L73&74=32K, L75=4K, L76=12K, L77=24K
PROCESS/SPEED	CMOS: 4 MHz	CMOS: 4 MHz	CMOS: 12 MHz	Low-Voltage CMOS: 8 MHz	Low-Voltage CMOS: 8 MHz
FEATURES	■ 8K/16K/OTP ROM ■ 256 Byte RAM ■ 160x7-Bit Video RAM ■ On-Screen Display (OSD) Video Controller ■ Programmable – Color – Size – Position Attributes ■ 13 PWMs for D/A Conversion ■ 128-Character Set ■ 4Kx6-Bit Char. Gen. ROM ■ Watch-Dog Timer (WDT) ■ Low-Voltage Protection ■ Five Ports/36 Pins ■ Two Standby Modes ■ Low-EMI Mode	■ 6K ROM, 256 Byte RAM ■ 120x7-Bit Video RAM ■ OSD On-Board Programmable – Color – Size – Position Attributes ■ 7 PWMs ■ 96 Character Set ■ 3Kx6-Bit Char. Gen. ROM ■ Watch-Dog Timer (WDT) ■ Low-Voltage Protection ■ Three Ports/20 Pins ■ Two Standby Modes ■ Low-EMI Mode	■ Conforms to FCC Line 21 Format ■ Parallel or Serial Modes ■ Stand-Alone Operation ■ On-Board Data Sync and Slicer ■ On-Board Character Generator – Color – Blinking – Italic – Underline – Extended Data Services	■ 1K ROM and 6K ROM ■ Watch-Dog Timer (WDT) ■ Two Analog Comparators with Output Option ■ Two Standby Modes ■ Two Counter/Timers ■ Auto Power-On Reset ■ 2V Operation ■ RC Oscillator Option ■ Low-Noise Option ■ Low-Voltage Protection ■ High-Current Drivers (2, 4)	■ Watch-Dog Timer (WDT) ■ Two Analog Comparators with Output Option ■ Two Standby Modes ■ Two Enhanced Counter/Timers – Auto Pulse – Reception/Generation ■ Auto Power-On Reset ■ 2V Operation ■ RC Oscillator Option ■ Low-Voltage Protection ■ High-Current Drivers – Three OTP Versions Available – Z86E72/73/74
PACKAGE	64-Pin DIP	40-Pin DIP	18-Pin DIP	18-Pin DIP 18-Pin SOIC	Z86L71=20-Pin DIP/SOIC Z86L70/L75=18-Pin DIP, SOIC Z86L72/L76/L77=40,44-Pin DIP, PLCC, QFP Z86L74=64/68-Pin
SUPPORT PRODUCTS	Z86C2700ZCO - Evaluation Board Z86C2700ZDB - Emulator Z86C2700ZEM - Emulator	Z86C2700ZDB - Emulator Z86C2702ZEM - Emulator Z86C2700ZCO - Evaluation Board	Support Documentation Provided with the device	Z86C5000ZEM - Emulator	Z86L7200TSC - Emulator Z86L7100ZEM - Emulator Z86L7100ZDB - Emulator

BLOCK DIAGRAM	512 Byte ROM Z8® CPU WDT / 64 RAM P2 / P3	1K ROM Z8® CPU WDT / 128 RAM P0 / P2	1K ROM Z8® CPU WDT / 128 RAM SPI P2 / P3
PART NUMBER	**Z86C03**	**Z86C04/Z86E04**	**Z86C06**
DESCRIPTION	Consumer Controller Processor (CCP™) with 512 Byte ROM	Z86C04 = 8-Bit Low Cost 1 Kbyte ROM MCU Z86E04 = OTP Version	Consumer Controller Processor (CCP™) with 1 Kbyte ROM
PROCESS/SPEED	CMOS: 8 MHz	CMOS: 8 MHz	CMOS: 12 MHz
FEATURES	■ 512 Byte ROM ■ 64 Byte RAM ■ Two Standby Modes ■ One Counter/Timer ■ ROM Protect ■ Two Analog Comparator ■ Auto Power-On Reset ■ Low-Voltage Protection ■ 14 I/O ■ RC Oscillator Option ■ Low-Noise Option	■ 1 Kbyte ROM ■ 128 Byte RAM ■ Two Standby Modes ■ Two Counter/Timer ■ ROM Protect ■ Two Analog Comparator ■ Auto Power-On Reset ■ Low-Voltage Protection (ROM Only) ■ 14 I/O ■ Low-Noise Option	■ 1 Kbyte ROM ■ 128-Byte RAM ■ Two Standby Modes ■ Two Counter/Timer ■ ROM Protect ■ Two Analog Comparator ■ Auto Power-On Reset ■ Low-Voltage Protection (ROM Only) ■ 14 I/O ■ RC Oscillator Option ■ Serial Peripheral Interface (SPI)
PACKAGE	18-Pin DIP 18-Pin SOIC	18-Pin DIP 18-Pin SOIC	18-Pin DIP 18-Pin SOIC
SUPPORT PRODUCTS	Z86CCP00ZEM - Emulator Z86CCP00ZAC - Emulator	Z86C0800ZCO - Evaluation Board Z86C0800ZDP - Adaptor Kit Z86C1200ZEM - Emulator Z86C1200ZPD - Adaptor Kit Z86CCP00ZEM - Emulator Z86CCP00ZAC - Emulator	Z86E0600ZDP - Adaptor Kit Z86C5000ZEM - Emulator Z86C5000ZDP - Adaptor Kit Z86CCP00ZEM - Emulator Z86CCP00ZAC - Emulator

BLOCK DIAGRAM				
PART NUMBER	**Z86321**	**Z89320**	**Z89321/Z89371**	**Z5380**
DESCRIPTION	8-Bit Digital Audio Processor	16-Bit Digital Signal Processor	16-Bit Digital Signal Processor Z89371= OTP Version	Small Computer System Interface (SCSI)
PROCESS/SPEED	CMOS: 12 MHz	CMOS: 10 MHz	CMOS: 20 MHz	Clock: 1.5 Mb/s
FEATURES	■ Sound Blaster™ Compatible ■ ADPCM Decompression ■ 8-Bit DAC Interface ■ Successive Approximation ADC Algoritant ■ MIDI Interface	■ 16-Bit Multiply/Accumulate ■ 100 ns ■ 512 Word RAM ■ 4K Word RAM ■ Peripherals Interface Bus ■ 74 Instruction Set	■ 16-Bit Multiply/Accumulate ■ 50 µs ■ 512 Word RAM ■ 4K Word ROM ■ Peripherals Interface Bus ■ CODEC Interface	■ Compatible 5380 Pin-out ■ CMOS ■ Asynchronous I/F Supports 1.5 Mb/s ■ 48 mA Drivers ■ Arbitration Support ■ Support Normal or Block Mode DMA
PACKAGE	40-Pin DIP 44-Pin PLCC	40-Pin DIP 44-Pin PLCC	40-Pin DIP 44-Pin PLCC	40-Pin DIP 44-Pin PLCC
SUPPORT PRODUCTS	Support Documentation Provided with Device	Z89C0000ZEM - Emulator	Z8937100ZEM - Emulator	Support Documentation Provided with Device

SoundBlaster™ is a Trademark of Creative Labs, Inc.

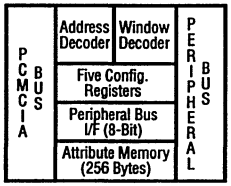
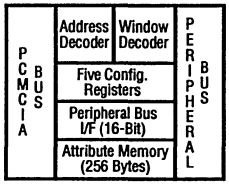
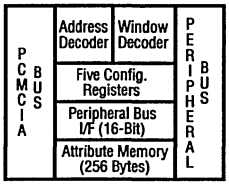
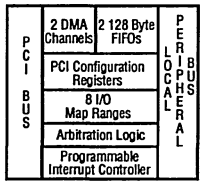
BLOCK DIAGRAM				
PART NUMBER	**Z8615**	**Z8614/Z8602**	**Z86E23**	**Z86C17**
DESCRIPTION	Keyboard MCU	Z8602 = 2K ROM Keyboard MCU Z8614 = 4K ROM Keyboard MCU	Keyboard OTP MCU	Mouse MCU
PROCESS/SPEED	NMOS: 4, 5 MHz	NMOS: 4 MHz	CMOS: 4 MHz	CMOS: 4 MHz
FEATURES	■ 4K ROM ■ 124-Byte RAM ■ 32 I/O Lines ■ Two Counter/Timers ■ Watch-Dog Timer (WDT) ■ RC Oscillator ■ Dedicated Row Column Pins ■ Data/Clock Pins ■ Direct Connect LED Pins	■ 4K ROM ■ 124 Byte RAM ■ 32 I/O Lines ■ Two Counter/Timers ■ Dedicated Row Column Pins	■ 8K ROM ■ 256 Byte RAM ■ 32 I/O Lines ■ Two Counter/Timers ■ Dedicated Row Column Pins	■ 2K ROM ■ 124 Byte RAM ■ 14 I/O Lines ■ Two Counter/Timers ■ Dedicated Opto-Transistor Pins ■ Integrated Pull-up Resistors ■ Power-Down Modes ■ Power-On Reset (POR) ■ Watch-Dog Timer (WDT)
PACKAGE	40-Pin DIP 44-Pin PLCC	40-Pin DIP 44-Pin PLCC	40-Pin DIP 44-Pin PLCC	18-Pin DIP 18-Pin SOIC
SUPPORT PRODUCTS	Z0861500ZCO - Evaluation Board Z86C1200ZEM -Emulator Z0861500ZDP - Adaptor Kit	Z0860200ZCO - Evaluation Board Z86C1200ZEM - Emulator Z0860200ZDP - Adaptor Kit Z86C1200ZPD - Emulator Pod	Z0860200ZCO - Evaluation Board Z86C1200ZEM - Emulator Z0860200ZDP - Adaptor Kit	Z86C1200ZEM - Emulator

BLOCK DIAGRAM				
PART NUMBER	**Z84C01**	**Z84C90**	**Z84013/Z84C13**	**Z84015/Z84C15**
DESCRIPTION	Z80® CPU with Clock Generator/Clock	Killer I/O (Three Z80® Peripherals)	Intelligent Peripheral Controller	Enhanced Intelligent Peripheral
PROCESS/SPEED	CMOS: 10 MHz	CMOS: 8, 10, 12 MHz	Z84013 = CMOS: 6, 10 MHz Z84C13 = CMOS: 6, 10 MHz	Z84015 = CMOS: 6, 10 MHz Z84C15 = CMOS: 16 MHz
FEATURES	■ Clock Generator/Controller ■ Four Power Down Modes	■ Serial Input/Output (SIO) ■ Counter/Timer Circuit (CTC) ■ Plus Eight I/O Lines ■ Three 8-Bit Ports	■ Serial Input/Output (SIO) ■ Counter/Timer Circuit (CTC) ■ Watch-Dog Timer (WDT) ■ Clock Generator Circuit (CGC) ■ Wait State Generator (WSG) ■ Power-On Reset (POR) ■ Two Chip Selects ■ Evaluation Mode	■ Serial Input/Output (SIO) ■ Counter/Timer Circuit (CTC) ■ Watch-Dog Timer (WDT) ■ Clock Generator Circuit (CGC) ■ Four Power-Down Modes ■ Power-On Reset ■ Two Chip Selects ■ 32-Bit CRC ■ Wait State Generator (WSG) ■ Evaluation Mode
PACKAGE	44-Pin QFP 44-Pin PLCC	84-Pin PLCC 80-Pin QFP	84-Pin PLCC	100-Pin QFP 100-Pin VQFP
SUPPORT PRODUCTS	Z84C9000ZCO - Evaluation Board	Z84C9000ZCO - Evaluation Board	Z84C1500ZCO - Evaluation Board	Z84C1500ZCO - Evaluation Board

BLOCK DIAGRAM	DSP 512 RAM | 4K ROM 16-BIT MAC DATA I/O | RAM I/O	Z8 | DSP 24K ROM | 4K WORD ROM 256 BYTES RAM | 512 WORD RAM 8-Bit A/D | 10-Bit D/A	Z8 | DSP ROMLess | 4K WORD ROM 256 BYTES RAM | 512 WORD RAM 8-Bit A/D | 10-Bit D/A	PCMCIA BUS Address Decoder | Window Decoder Five Config Registers Peripheral Bus I/F (16-Bit) Attribute Memory (256 Bytes) PERIPHERAL BUS
PART NUMBER	**Z89C00**	**Z89120**	**Z89920**	**Z86017**
DESCRIPTION	16-Bit Digital Signal Processor	Zilog Modem/Fax Controller	Zilog Modem/Fax Controller	PCMCIA Interface Adaptor
PROCESS/SPEED	CMOS: 10, 15 MHz	CMOS: 20 MHz	CMOS: 20 MHz	CMOS: 20 MHz
FEATURES	■ 16-Bit Multiply/Accumulate ■ 75 ns ■ Two Data RAMs (256 Words each) ■ 4K Word ROM ■ 64Kx16 Ext. ROM ■ 16-Bit I/O Port ■ 74 Instructions ■ Most Single Cycle ■ Two Conditional Branch Inputs, Two User Outputs ■ Library of Macros ■ Zero Overhead Pointers	■ Z8® with 24 Kbyte ROM ■ 16-Bit DSP with 4K Word ROM ■ 8-Bit A/D ■ 10-Bit D/A (PWM) □ Library of Macros □ 47 I/O Pins □ Two Comparators Independent Z8® and DSP Operations Power-Down Mode	■ Z8 with 64K External Memory ■ DSP with 4K Word ROM ■ 8-Bit A/D ■ 10-Bit D/A ■ Library of Macros ■ 47 I/O Pins ■ Two Comparators Independent Z8® and DSP Operations Power-Down Mode	■ 256 Bytes of Attribute Memory ■ Five Configuration Registers ■ EEPROM Sequencer or SPI Interface ■ PCMCIA to I/O, Memory or Both ■ PCMCIA to ATA/IDE ■ ATA/IDE to ATA/IDE ■ 3.0V to 5.5V Operation ■ 8- or 16-Bit Peripheral Support
PACKAGE	68-Pin PLCC 60-Pin VQFP	68-Pin PLCC	68-Pin PLCC	100-Pin VQFP
SUPPORT PRODUCTS	Z89C0000ZEM - Emulator Z89C0000ZCC - Emulator	Z89C6501ZEM - Emulator Z89C6500ZDP - Emulator	Z89C6501ZEM - Emulator Z89C6500ZDB - Emulator	Z8601700ZCO - Evaluation Board

BLOCK DIAGRAM	SCC	FIFO FIFO 85C30 SCC (2 CH)	SCC DMA DMA DMA DMA BIU	85C30 SCC 53C80 SCSI
PART NUMBER	**Z8030/Z80C30** **Z8530/Z85C30**	**Z85230/Z80230** **Z85233**	**Z16C35**	**Z85C80**
DESCRIPTION	Serial Communication Controller Z8030/Z80C30 = Multiplexed Bus Z8530/Z85C30 = Non-Multiplexed Bus	Enhanced Serial Communication Controller Z8230/Z80230 = Dual Channel Z85233 = Single Channel	Integrated Serial Communication Controller	SCSCI Serial Communication and Small Computer Interface
PROCESS/SPEED	Z8030/Z8530 = NMOS: 4, 6, 8 MHz Z80C30/Z85C30 = CMOS: 8,10 16 MHz Clock: 2, 2.5, 4 Mb/s	CMOS: 10, 16 20 MHz Clock: 2.5, 4.0, 5.0 Mb/s	CMOS: 10, 16 MHz Clock: 2.5, 4.0 Mb/s	CMOS: 10, 16 MHz Clock: 2.5 Mb/s
FEATURES	■ Two Independent Full-Duplex Channels ■ Enhanced DMA Support: ■ 10x19 Status FIFO ■ 14-Bit Byte Counter ■ NRZ/NRZI/FM Encoding Modes	■ Full Dual-Channel SCC Plus Deeper FIFOs: – 4 Bytes on Transmitters – 8 Bytes on Receivers ■ DPLL Counter Per Channel ■ Software Compatible to SCC	■ Full Dual-Channel SCC ■ Four DMA Controllers ■ Bus Interface Unit	■ Two Independent Full-Duplex Channels ■ Direct SCSI Bus Interface ■ Supports SCSI ANSI-X3.131-1986 Standard
PACKAGE	40-Pin DIP 44-Pin CERDIP 44-Pin PLCC	40-Pin DIP 44-Pin PLCC 44-Pin QFP (Z85233 Only)	68-Pin PLCC	68-Pin PLCC 100-Pin VQFP
SUPPORT PRODUCTS	Z8018600ZCO - Evaluation Board Z8523000ZCO - Evaluation Board Z8018100ZCO - Evaluation Board ZEPMD000002 - EPM™ Manual	Z8018600ZCO - Evaluation Board Z8S18000ZCO - Evaluation Board Z8038000ZCO - Evaluation Board Z8523000ZCO - Evaluation Board ZEPMDC00002 - EPM™ Manual	Z8018600ZCO - Evaluation Board	ZEPMD00002 - EPM™ Manual

BLOCK DIAGRAM	UART CPU OSC 256 RAM CLOCK P0 P1 P2 P3	8K PROM UART CPU 256 RAM P0 P1 P2 P3	DSP 512 RAM 4K ROM 16-BIT MAC DATA I/O RAM I/O	MULT DIV UART CPU OSC 256 RAM CLOCK P0 P1 P2 P3
PART NUMBER	**Z86C91/Z8691**	**Z86E21/Z86C21**	**Z89C00**	**Z86C93**
DESCRIPTION	ROMLess Z8®	Z86E21 = 8K OTP Z86C21 = 8K ROM	16-Bit Digital Signal Processor	ROMLess Enhanced Z8® Mult/Div
PROCESS/SPEED	Z86C91 = CMOS: 16 MHz Z8691 = NMOS: 12 MHz	CMOS: 12, 16 MHz	CMOS: 10, 15 MHz	CMOS: 20, 25, 33 MHz
FEATURES	■ Full-Duplex UART ■ Two Standby Modes (STOP and HALT) ■ 2x8 Bit ■ Counter/Timer	■ 256 Byte RAM ■ Full-Duplex UART ■ Two Standby Modes (STOP and HALT) ■ Two Counter/Timers ■ ROM Protect Option ■ RAM Protect Option ■ Low-EMI Option	■ 16-Bit Multiply/Accumulate ■ 75 ns ■ Two Data RAMs (256 Words Each) ■ 4K Word ROM ■ 64Kx16 Ext. ROM ■ 16-Bit I/O Port ■ 74 Instructions ■ Most Single Cycle ■ Two Conditional Branch Inputs, Two User Outputs ■ Library of Macros ■ Zero Overhead Pointers	■ 16x16 Multiply 17 Clocks ■ 32x16 Divide 20 Clocks ■ Full-Duplex UART ■ Two Standby Modes (STOP and HALT) ■ Three 16-Bit Counter/Timers
PACKAGE	40-Pin DIP 44-Pin PLCC 44-Pin QFP	40-Pin DIP 44-Pin PLCC 44-Pin QFP	68-Pin PLCC	40-Pin DIP 44-Pin PLCC 44-Pin QFP
SUPPORT PRODUCTS	Z0860000ZCO - Evaluation Board Z86C0000ZUSP064 - Signum Emulator Z86C1200ZPD - Signum Emulator Pod	Z0860000ZCO - Evaluation Board Z86C0000ZUSP064 - Signum Emulator Z86C1200ZPD - Signum Emulator Pod	Z89C00ZEM - Emulator	Z0860000ZCO - Evaluation Board Z86C0000ZUSP064 - Signum Emulator Z86C0001ZUSP064 - Signum Emulator Z86C9300ZPD - Signum Emulator Pod Z86C9301ZPD - Signum Emulator Pod

BLOCK DIAGRAM				
PART NUMBER	**Z86016**	**Z86017**	**Z86M17**	**Z86020**
DESCRIPTION	8-Bit PCMCIA Interface Adaptor	PCMCIA Interface Adaptor	PCMCIA Interface Adaptor	PCI/Multifunction Bridge
PROCESS/SPEED	CMOS: 20 MHz	CMOS: 20 MHz	CMOS: 20 MHz	CMOS: 33 MHz
FEATURES	■ Z86017 with 8-Bit Peripheral Bus Only	■ 256 Bytes of Attribute Memory ■ Five Configuration Registers ■ EEPROM Sequencer or SPI Interface ■ PCMCIA to I/O, Memory or Both ■ PCMCIA to ATA/IDE ■ ATA/IDE to ATA/IDE ■ 3.0V to 5.5V Operation ■ 8- or 16-Bit Peripheral Support	■ Mirror Image Pin-Out of Z86017 for Opposite PCB - Surface Layout	■ 256 Bytes of Configuration Memory ■ 64 PCI Configuration Registers ■ Eight Programmable Memory or I/O Map Ranges with Independent Timing Control ■ 128 Byte FIFO's ■ Two Full Featured DMA Channels ■ PCI Initiator/Target Operations ■ On-Chip Peripheral Bus Arbitration
PACKAGE	48-Pin VQFP 64-Pin VQFP	100-Pin VQFP	100-Pin VQFP	160-Pin QFP
SUPPORT PRODUCTS	Z8601600ZCO - Evaluation Board (Available Q494)	Z8601700ZCO -Evaluation Board	Z8601700ZCO - Evaluation Board	Available Q494

Z8® MICROCONTROLLERS - CONSUMER FAMILY OF PRODUCTS

Databooks By Market Niche	Part No	Unit Cost
Z8® Microcontrollers Databook	**DC-8305-02**	**$ 5.00**

Product Specifications

Z86C07 CMOS Z8 8-Bit Microcontroller
Z86C08 CMOS Z8 8-Bit Microcontroller
Z86E08 CMOS Z8 8-Bit OTP Microcontroller
Z86C11 CMOS Z8 Microcontroller
Z86C12 CMOS Z8 In-Circuit Microcontroller Emulator
Z86C21 8K ROM Z8 CMOS Microcontroller
Z86E21 CMOS Z8 8K OTP Microcontroller
Z86C61/62/96 CMOS Z8 Microcontroller
Z86C63/64 32K ROM Z8 CMOS Microcontroller
Z86C91 CMOS Z8 ROMless Microcontroller
Z86C93 CMOS Z8 Multiply/Divide Microcontroller

Support Product Specifications

Z0860000ZCO Development Kit
Z86C0800ZCO Applications Board
Z86C0800ZDP Adaptor Board
Z86E2100ZDF Adaptor Kit
Z86E2100ZDP Adaptor Kit
Z86E2100ZDV Adaptor Kit
Z86E2100ZDV Adaptor Kit
Z86E2101ZDF Conversion Kit
Z86E2101ZDV Conversion Kit
Z86C6100TSC Z86C61/63 MCU OTP Emulation Board
Z86C6200ZEM In-Circuit Emulator
Z86C1200ZEM Z8® In-Circuit Emulator -C12
Z8® S Series Emulators, Base Units and Pods

Additional Information

Zilog's Superintegration™ Products Guide
Literature Guide
Third Party Support Vendors
Zilog's Sales Offices, Representatives and Distributors

Infrared Remote (IR) Controllers Databook	**DC-8301-04**	**$ 5.00**

Product Specifications

Z86L06 Low Voltage CMOS Consumer Controller Processor (Preliminary)
Z86L29 6K Infrared (IR) Remote (ZIRC™) Controller (Advance Information)
Z86L70/L71/L72/L75/L76 Zilog IR (ZIRC™) CCP™ Controller Family (Preliminary)
Z86E72/E73/E74 Zilog IR (ZIRC™) CCP™ Controller Family (Preliminary)

Application Note

Beyond the 3 Volt Limit

Support Product Specifications

Z86L7100ZDB Emulator Board
Z86L7100ZEM ICEBOX™ In-Circuit Emulator Board

Additional Information

Zilog's Superintegration™ Products Guide
Literature Ordering Guide
Zilog's Sales Offices, Representatives and Distributors

Z8® MICROCONTROLLERS - CONSUMER FAMILY OF PRODUCTS

Databooks By Market Niche	Part No	Unit Cost
Telephone Answering Device Databook	**DC-8300-02**	**$ 5.00**

Product Specifications
- Z89C65, Z89C66 (ROMless) Dual Processor T.A.M. Controller (Preliminary)
- Z89C67, Z89C68/C69 (ROMless) Dual Processor Tapeless T.A.M. Controller (Preliminary)

Development Guides
- Z89C65 Software Development Guide
- Z89C67/C69 Software Development Guide

Technical Notes
- Using Samsung KT8554 Codec on the ZTAD Development Board
- Z89C67/C69 Design Guidelines
- Z89C67/C69 ARAM Bit-Rate Measurements
- Z89C67 Codec Interfacing (Preliminary)
- Controlling the Out –5V and Codec Clock Signals for Low-Power Halt Mode

Support Product Specifications
- Z89C5900ZEM Emulation Module
- Z89C6500ZDB Emulation Board
- Z89C6501ZEM ICEBOX™ In-Circuit Emulator
- Z89C6700ZDB Emulator Board
- Z89C6700ZEM ICEBOX™ Emulator Board

Additional Information
- Zilog's Superintegration™ Products Guide
- Literature Ordering Guide
- Zilog's Sales Offices, Representatives and Distributors

Z8® MICROCONTROLLERS - PERIPHERALS MULTIMEDIA FAMILY OF PRODUCTS

Databooks By Market Niche	Part No	Unit Cost
Keyboard/Mouse/Pointing Devices Databook	**DC-8304-00**	**$ 5.00**

Product Specifications

Z8602 NMOS Z8® 8-Bit Keyboard Controller
Z8614 NMOS Z8® 8-Bit Keyboard Controller
Z8615 NMOS Z8® 8-Bit Keyboard Controller
Z86E23 Z8® 8-Bit Keyboard Controller with 8K OTP
Z86C04 CMOS Z8® 8-Bit Microcontroller
Z86C08 CMOS Z8® 8-Bit Microcontroller
Z88C17 CMOS Z8® 8-Bit Microcontroller

Additional Information

Zilog's Superintegration™ Products Guide
Literature Guide

PC Audio Databook	**DC-8317-00**	**$ 5.00**

Product Specifications

Z86321 Digital Audio Processor (Preliminary)
Z89320 16-Bit Digital Signal Processor (Preliminary)
Z89321/371 16-Bit Digital Signal Processor (Preliminary)
Z89331 16-Bit PC ISA Bus Interface (Advance Information)
Z89341/42/43 Wave Synthesis Chip Set (Advance Information)
Z5380 Small Computer System Interface

Additional Information

Zilog's Superintegration™ Products Guide
Literature Guide

Z8® MICROCONTROLLERS LITERATURE (Continued)

Technical Manuals and Users Guides	Part No.	Unit Cost
Z8® Microcontrollers Technical Manual	DC-8291-02	5.00
Z86018 Preliminary User's Manual	DC-8296-00	N/C
Digital TV Controller User's Manual	DC-8284-01	5.00
Z89C00 16-Bit Digital Signal Processor User's Manual/DSP Software Manual	DC-8294-02	5.00
Z86C95 16-Bit Digital Signal Processor User Manual	DC-8595-00	5.00
Z86017 PCMCIA Adaptor Chip User's Manual and Databook	DC-8298-03	5.00
PLC Z89C00 Cross Development Tools Brochure	DC-5538-01	N/C

Z8® Application Notes	Part No	Unit Cost
The Z8 MCU Dual Analog Comparator	DC-2516-01	N/C
Z8 Applications for I/O Port Expansions	DC-2539-01	N/C
Z86E21 Z8 Low Cost Thermal Printer	DC-2541-01	N/C
Zilog Family On-Chip Oscillator Design	DC-2496-01	N/C
Using the Zilog Z86C06 SPI Bus	DC-2584-01	N/C
Interfacing LCDs to the Z8	DC-2592-01	N/C
X-10 Compatible Infrared (IR) Remote Control	DC-2591-01	N/C
Z86C17 In-Mouse Applications	DC-3001-01	N/C
Z86C40/E40 MCU Applications Evaluation Board	DC-2604-01	N/C
Z86C08/C17 Controls A Scrolling LED Message Display	DC-2605-01	N/C
Z86C95 Hard Disk Controller Flash EPROM Interface	DC-2639-01	N/C
Three Z8® Applications Notes: Timekeeping with Z8; DTMF Tone Generation; Serial Communication Using the CCP Software UART	DC-2645-01	N/C

Z80®/Z8000® DATACOMMUNICATIONS FAMILY OF PRODUCTS

Databooks	Part No	Unit Cost
Z80 Family Databook	**DC-8321-00**	**5.00**

Discrete Z80® Family

Z8400/C00 NMOS/CMOS Z80® CPU Product Specification
Z8410/C10 NMOS/CMOS Z80 DMA Product Specification
Z8420/C20 NMOS/CMOS Z80 PIO Product Specification
Z8430/C30 NMOS/CMOS Z80 CTC Product Specification
Z8440/C40 NMOS/CMOS Z80 SIO Product Specification

Embedded Controllers

Z84C01 Z80 CPU with CGC Product Specification
Z8470 Z80 DART Product Specification
Z84C90 CMOS Z80 KIO™ Product Specification
Z84013/015 Z84C13/C15 IPC/EIPC Product Specification

Application Notes and Technical Articles

Z80® Family Interrupt Structure
Using the Z80® SIO with SDLC
Using the Z80® SIO in Asynchronous Communications
Binary Synchronous Communication Using the Z80® SIO
Serial Communication with the Z80A DART
Interfacing Z80® CPUs to the Z8500 Peripheral Family
Timing in an Interrupt-Based System with the Z80® CTC
A Z80-Based System Using the DMA with the SIO
Using the Z84C11/C13/C15 in Place of the Z84011/013/015
On-Chip Oscillator Design
A Fast Z80® Embedded Controller
Z80® Questions and Answers

Additional Information

Zilog's Superintegration™ Products Guide
Literature Guide
Third Party Support Vendors
Zilog's Sales Offices, Representatives and Distributors

Z80®/Z8000® DATACOMMUNICATIONS FAMILY OF PRODUCTS

Databooks and User's Manuals	Part No	Unit Cost
Z8000 Family of Products	**DC-8319-00**	**5.00**
Z8000 Family Databook		
Zilog's Z8000 Family Architecture		
Z8001/Z8002 Z8000 CPU Product Specification		
Z8016 Z8000 Z-DTC Product Specification		
Z8036 Z8000 Z-CIO Product Specification		
Z8536 CIO Counter/Timer and Parallel I/O Unit Product Specification		
Z8038/Z8538 FIO FIFO Input/Output Interface Unit Product Specification		
Z8060/Z8560 FIFO Buffer Unit		
Z8581 Clock Generator and Controller Product Specification		
User's Manuals		
Z8000 CPU Central Processing Unit User's Manual		
Z8010 Memory Management Unit (MMU) User's Manual		
Z8036 Z-CIO/Z8536 CIO Counter/Timer and Parallel Input/Output User's Manual		
Z8038 Z8000 Z-FIO FIFO Input/Output Interface User's Manual		
Z8000 Application Notes and Military Products		
Application Notes		
Using SCC with Z8000 in SDLC Protocol		
SCC in Binary Synchronous Communication		
Zilog's Military Products Overview		
Additional Information		
Zilog's Superintegration™ Products Guide		
Literature Guide		
Zilog's Sales Offices, Representatives and Distributors		
Z80 Family Technical Manual	DC-8309-00	5.00
Z80180 Z180 MPU Microprocessor Unit Technical Manual	DC-8276-04	5.00
Z280 MPU Microprocessor Unit Technical Manual	DC-8224-03	5.00
Z380™ Preliminary Product Specification	DC-6003-03	N/C
Z380™ User's Manual	DC-8297-03	5.00
ZNW2000 User's Manual for PC WAN Adaptor Board Development Kit	DC-8315-00	N/C
SCC Serial Communication Controller User's Manual	DC-8293-02	5.00
High-Speed SCC, Z16C30 USC User's Manual	DC-8280-04	5.00
High-Speed SCC, Z16C32 IUSC User's Manual	DC-8292-02	5.00
Z16C35 ISCC Integrated Serial Communication Controller Technical Manual	DC-8286-01	5.00
Z16C35 ISCC Integrated Serial Communication Controller Addendum	DC-8286-01A	N/C

GENERAL LITERATURE

Catalogs, Handbooks, Product Flyers and Users Guides	Part No	Unit Cost
Superintegration Master Selection Guide 1994-1995	DC-5634-00	N/C
Superintegration Products Guide	DC-5676-00	N/C
Quality and Reliability Report	DC-8329-00	N/C
ZIA™ 3.3-5.5V Matched Chip Set for AT Hard Disk Drives Datasheet	DC-5556-01	N/C
ZIA ZIA00ZCO Disk Drive Development Kit Datasheet	DC-5593-01	N/C
Zilog Hard Disk Controllers - Z86C93/C95 Datasheet	DC-5560-01	N/C
Zilog Infrared (IR) Controllers - ZIRC™ Datasheet	DC-5558-01	N/C
Zilog V. Fast Modem Controller Solutions	DC-5525-02	N/C
Zilog Digital Signal Processing - Z89320 Datasheet	DC-5547-01	N/C
Zilog Keyboard Controllers Datasheet	DC-5600-01	N/C
Z380™ - Next Generation Z80®/Z180™ Datasheet	DC-5580-02	N/C
Fault Tolerant Z8® Microcontroller Datasheet	DC-5603-01	N/C
32K ROM Z8® Microcontrollers Datasheet	DC-5601-01	N/C
Zilog Datacommunications Brochure	DC-5519-00	N/C
Z89300 DTC Controller Family Brochure	DC-5608-01	N/C
Zilog Digital Signal Processing Brochure	DC-5536-02	N/C
Zilog ASSPs - Partnering With You Product Brochure	DC-5553-01	N/C
Zilog Wireless Products Datasheet	DC-5630-00	N/C
Zilog Z8604 Cost Efficient Datasheet	DC-5662-00	N/C
Zilog Chip Carrier Device Packaging Datasheet	DC-5672-00	N/C
Zilog Database of IR Codes Datasheet	DC-5631-00	N/C
Zilog PCMCIA Adaptor Chip Z86017 Datasheet	DC-5585-01	N/C
Zilog Television/Video Controllers Datasheet	DC-5567-01	N/C
Zilog TAD Controllers - Z89C65/C67/C69 Datasheet	DC-5561-02	N/C
Zilog Z87000 Z-Phone Datasheet	DC-5632-00	D/C
Zilog 1993 Annual Report	DC-1993-AR	N/C
Zilog 1994 First Quarter Financial Report	DC-1994-Q1	N/C

ZiLOG LITERATURE GUIDE

ORDERING INFORMATION

Complete the attached literature order form. Be sure to enclose the proper payment or supply a purchase order. Please reference specific order requirements.

MINIMUM ORDER REQUIREMENTS

Orders under $300.00 must be prepaid by check, money order or credit card. Canadian and foreign orders must be accompanied by a cashier's check in U.S. dollars, drawn on a correspondent U.S. bank only.
Orders over $300.00 may be submitted with a Purchase Order.

SHIPMENT

Orders will be shipped after your check is cashed or credit is checked via the most economical method. Please allow four weeks for delivery.

RETURNS ARE NOT ACCEPTED.

PLEASE PRINT OR TYPE

NAME

PHONE ( ) -

COMPANY

ADDRESS

Method of Payment (Check One)

☐ Check ☐ Money Order

Credit Card ☐ VISA ☐ M/C ☐ P.O. (over $300.00)

CITY | STATE | ZIP | COUNTRY

PART NUMBER	DOCUMENT TITLE	UNIT COST	QTY.	TOTAL
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$
		$		$

Mail To:

ZiLOG
210 E. HACIENDA AVE. M/S C1-0
CAMPBELL, CA 95008-6600

Phone: (408)370-8016
Fax: (408)370-8056

Credit Card or Purchase Order # _____ SUBTOTAL

Expiration Date _____ ADD APPLICABLE SALES TAX (CA ONLY)

Signature _____ ADD 10% SHIPPING AND HANDLING

TOTAL

ZILOG

ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS

CALIFORNIA
Agoura 818-707-2160
Campbell 408-370-8120
Irvine 714-453-9701
San Diego 619-658-0391

COLORADO
Boulder 303-494-2905

FLORIDA
Clearwater 813-725-8400

GEORGIA
Duluth 404-931-4022

ILLINOIS
Schaumburg 708-517-8080

MINNESOTA
Minneapolis 612-944-0737

NEW HAMPSHIRE
Nashua 603-888-8590

OHIO
Independence 216-447-1480

OREGON
Portland 503-274-6250

PENNSYLVANIA
Horsham 215-784-0805

TEXAS
Austin 512-343-8976
Dallas 214-987-9987

INTERNATIONAL SALES OFFICES

CANADA
Toronto 905-850-2377

CHINA
Shenzhen 86-755-2236089
Shanghai 86-21-4370050, x5204
86-21-4331020

GERMANY
Munich 49-8967-2045
Sömmerda 49-3634-23906

JAPAN
Tokyo 81-3-3587-0528

HONG KONG
Kowloon 852-7238979

KOREA
Seoul 82-2-577-3272

SINGAPORE
Singapore 65-2357155

TAIWAN
Taipei 886-2-741-3125

UNITED KINGDOM
Maidenhead 44-628-392-00

Zilog, Inc. 210 East Hacienda Ave.
Campbell, CA 95008-6600
Telephone (408) 370-8000
Telex 910-338-7621
FAX 408 370-8056

Z

SALES REPRESENTATIVES AND DISTRIBUTORS

U.S., CANADIAN & PUERTO RICAN REPRESENTATIVES

UTAH

Salt Lake City
Thorson Rocky Mountain ..... (801) 942-1683

WASHINGTON

Kirkland
Phase II Technical Sales ..... (206) 823-3874

WISCONSIN

Brookfield
Victory Sales, Inc. ..... (414) 789-5770

CANADA

British Columbia
BBD Electronics, Inc. ..... (604) 465-4907
Ontario
BBD Electronics, Inc. ..... (905) 821-7800
Ottawa
BBD Electronics, Inc. ..... (613) 764-1752
Quebec
BBD Electronics, Inc. ..... (514) 697-0801

PUERTO RICO

San Juan
Semtronic Associates, Inc. ..... (809) 766-0700

SOUTH AMERICAN REPRESENTATIVES

ARGENTINA

Buenos Aires
Parallax Sales & Distribution ..... (1) 372-7140

BRAZIL

Sao Paulo
Parallax Sales & Distribution ..... (11) 535-1755

SALES REPRESENTATIVES AND DISTRIBUTORS

U.S. AND CANADIAN DISTRIBUTORS

INDIANA

Indianapolis
Arrow Electronics ..... (317) 299-2071
Hamilton Hallmark Electronics ..... (317) 872-8875
(800) 829-0146

IOWA

Cedar Rapids
Arrow Electronics ..... (319) 395-7230

KANSAS

Lenexa
Arrow Electronics ..... (913) 541-9542
Hamilton Hallmark Electronics ..... (913) 888-4747
(800) 332-4375

KENTUCKY

Lexington
Hamilton Hallmark Electronics ..... (800) 235-6039
(800) 525-0069

MARYLAND

Columbia
Anthem Electronics ..... (410) 995-6640
Arrow Electronics ..... (410) 596-7000
Hamilton Hallmark Electronics ..... (410) 988-9800

MASSACHUSETTS

Peabody
Hamilton Hallmark Electronics ..... (508) 532-9808

Wilmington
Anthem Electronics ..... (508) 657-5170
Arrow Electronics ..... (508) 658-0900

MICHIGAN

Livonia
Arrow Electronics ..... (313) 462-2290

Nori
Hamilton Hallmark Electronics ..... (313) 347-4271

Plymouth
Hamilton Hallmark Electronics ..... (313) 416-5800
(800) 767-9654

MINNESOTA

Bloomington
Hamilton Hallmark Electronics ..... (612) 881-2600

Eden Prairie
Anthem Electronics ..... (612) 944-5454
Arrow Electronics ..... (612) 941-5280

MISSOURI

Earth City
Hamilton Hallmark Electronics ..... (314) 291-5350

St. Louis
Arrow Electronics ..... (314) 567-6888

NEVADA

Sparks
Arrow Electronics ..... (702) 331-5000

NEW JERSEY

Cherry Hill
Hamilton Hallmark Electronics ..... (609) 235-1900

Marlton
Arrow Electronics ..... (609) 596-8000

Pinebrook
Anthem Electronics ..... (201) 227-7960
Arrow Electronics ..... (201) 227-7880

Parsippany
Hamilton Hallmark Electronics ..... (201) 575-4415

NEW YORK

Commack
Anthem Electronics ..... (516) 864-6600

Hauppauge
Arrow Electronics ..... (516) 231-2500
Hamilton Hallmark Electronics ..... (516) 737-0600

Melville
Arrow Electronics ..... (516) 391-1300

Rochester
Arrow Electronics ..... (716) 427-0300
Hamilton Hallmark Electronics ..... (716) 475-9130

Ronkonkoma
Hamilton Hallmark Electronics ..... (516) 737-0600

NORTH CAROLINA

Raleigh
Arrow Electronics ..... (919) 876-3132
Hamilton Hallmark Electronics ..... (919) 872-0712

SALES REPRESENTATIVES AND DISTRIBUTORS

CENTRAL AND SOUTH AMERICA

MEXICO
Semiconductores Profesionales 525-524-6123
Proyeccion Electronica 525-264-7482

ARGENTINA
Buenos Aires
YEL SRL 011-541-440-1532

BRAZIL
Sao Paulo
Nishicom 011-55-11-535-1755

ASIA-PACIFIC

AUSTRALIA
R&D Electronics 61-3-558-0444
GEC Electronics Division 61-2-638-1888

CHINA
Beijing
Lestina International Ltd. ..... 86-1-849-8888 Rm. 20469
China Electronics Appliance Corp. 86-755-335-4214
TLG Electronics, Ltd. 85-2-388-7613

Guang Zhou
Lestina International Ltd. 86-20-885-0613
86-20-886-1615

HONG KONG
Lestina International Ltd. 852-735-1736
Electrocon Products Ltd. 852-481-6022

INDIA
Bangalore
Zenith Technologies Pvt. Ltd. 91-812-586782
Bombay
Zenith Technologies Pvt. Ltd. 91-22-4947457

INDONESIA
Jakarta
Cinergi Asiamaju 62-21-7982762

JAPAN
Tokyo
Teksel Co., Ltd. 81-3-5467-9000
Internix Incorporated 81-3-3369-1101
Kanematsu Elec. Components Corp. 81-3-3779-7811
Osaka
Teksel Co., Ltd. 81-6368-9000

KOREA
ENC-Korea 822-523-2220

MALAYSIA
Eltee Electronics Ltd. 60-3-7038498

NEW ZEALAND
GEC Electronics Division 64-25-971057

PHILIPPINES
Alexan Commercial 63-2-402223

SINGAPORE
Eltee Electronics Ltd. 65-2830888

TAIWAN (ROC)
Acer Sertek, Inc. 886-2-501-0055
Orchard Electronics Co. 886-2-504-7083
Promate Electronics Co. Ltd. 886-2-659-0303

THAILAND
Eltee Electronics Ltd. 66-2-538-4600

SALES REPRESENTATIVES AND DISTRIBUTORS

ISRAEL
RDT 972-36450707

ITALY
Milano
De Mico S.P.A.. 0039-295-343600
EBV Elektronik 0039-2-66017111
Firenze
EBV Elektronik 0039-55-350792
Roma
EBV Elektronik 0039-6-2253367
Modena
EBV Elektronik 0039-59-344752
Napoli
EBV Elektronik 0039-81-2395540
Torino
EBV Elektronik 0039-11-2161531
Vicenza
EBV Elektronik 0039-444-572366

NETHERLANDS
EBV Elektronik 313-46562353

NORWAY
Bexab Norge 47-63833800

POLAND
Warsaw
Gamma Ltd. 004822-330853

PORTUGAL
Amadora
Amitron-Arrow. 0035-1-4714806

RUSSIA
Woronesh
Thesys/Intertechna 0732593697
Vyborg
Gamma Ltd. 081278-31509
St. Petersburg
Gamma Ltd. 0812-5131402

SPAIN
Barcelona
Amitron-Arrow S.A. 0034-3-4907494
Madrid
Amitron-Arrow S.A. 0034-1-3043040

SWEDEN
Bexab Sweden AB 46-8-630-8800

SWITZERLAND
Dietikon
EBV Elektronik GMBH 0041-1-7401090
Lausanne
EBV Elektronik AG 0041-21-3112804
Regensdorf
Eurodis AG 0041-1-8433111

UKRAINE
Kiev
Thesys/Mikropribor 04434-9533

Z

Q3/94 DC 8297-03

Zilog, Inc.
210 East Hacienda Ave.
Campbell, CA 95008-6600
408-370-8000