



**Z8<sup>PLUS</sup>**  
**USER'S MANUAL**

---

---

---

© 1999 by ZiLOG, Inc. All rights reserved. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of ZiLOG, Inc. The information in this document is subject to change without notice. Devices sold by ZiLOG, Inc. are covered by warranty and patent indemnification provisions appearing in ZiLOG, Inc. Terms and Conditions of Sale only.

ZiLOG, Inc. makes no warranty, express, statutory, implied or by description, regarding the information set forth herein or regarding the freedom of the described devices from intellectual property infringement. ZiLOG, Inc. makes no warranty of merchantability or fitness for any purpose.

The software described herein is provided on an as-is basis and without warranty. ZiLOG accepts no liability for incidental or consequential damages arising from use of the software.

ZiLOG, Inc. shall not be responsible for any errors that may appear in this document. ZiLOG, Inc. makes no commitment to update or keep current the information contained in this document.

ZiLOG's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the customer and ZiLOG prior to use. Life support devices or systems are those which are intended for surgical implantation into the body, or which sustains life whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

**ZiLOG, Inc.**

**910 East Hamilton Ave., Suite 110**

**Campbell, CA 95008**

**Telephone: (408) 558-8500**

**FAX: (408) 558-8300**

**Internet: <http://www.zilog.com>**

---

---

---



The following conventions have been adopted to provide clarity and ease of use:

- Courier Font For Executables

Commands, variables, icon names, entry field names, selection buttons, code examples, and other executable items are distinguished by the use of the Courier font. Where the use of the font is not possible, like in the Index, the name of the entity is capitalized. For example, a procedure may contain an instruction which appears as: Click on `File`. However, an Index entry would appear as `FILE`.

- Grouping of Actions Within A Procedure Step

Actions in a procedure step are all performed on the same window or dialog box. Actions performed on different windows or dialog boxes appear in separate steps.

- Sequencing Words Within A Procedure Step

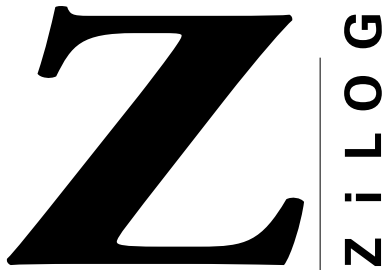
When an item in a procedure contains a series of actions, the second action is preceded by the word *then*, and the third and subsequent actions are preceded by the word *and*. For example: Click on `View`, *then* `Memory`, *and* `Z8 Code Memory`.

- Unavailable menu items are presented in gray.

## **ADDITIONAL SOURCES OF INFORMATION**

In addition to this manual, you should have access to and be familiar with the following documentation:

- *Z8 Microcontrollers User's Manual*, UM95Z800103
- Data Sheet for each product with which you work.



*Totally Logical*

**TABLE OF CONTENTS**

<b>Chapter Title and Subsections</b>	<b>Page</b>
<b>Chapter 1. Address Space</b>	
Introduction .....	1-1
Register File Space .....	1-1
General-Purpose Registers .....	1-5
Working Register Groups .....	1-6
Precautions .....	1-8
Control and Peripheral Registers .....	1-10
Control Registers .....	1-10
Peripheral Registers .....	1-10
Program Memory .....	1-11
Stack .....	1-13
<b>Chapter 2. Addressing Modes</b>	
Addressing Modes .....	2-1
Register Addressing (R) .....	2-2
Indirect Register Addressing (IR) .....	2-3
Indexed Addressing (X) .....	2-5
Direct Addressing (DA) .....	2-7
Relative Addressing (RA) .....	2-8
Immediate Data Addressing (IM) .....	2-9

---

Chapter Title and Subsections	Page
<b>Chapter 3. Instruction Set</b>	
Functional Summary .....	3-1
Processor Flags .....	3-5
Condition Codes .....	3-7
Notation And Binary Encoding .....	3-10
Assembly Language Syntax .....	3-12
Z8Plus Instruction Summary .....	3-12
Opcode Map .....	3-18
Instruction Description and Formats .....	3-19
ADC—Add with Carry .....	3-20
ADC—Add with Carry .....	3-22
ADD—Add .....	3-23
AND—Logical AND .....	3-25
CALL—Call Procedure .....	3-27
CCF—Complement Carry Flag .....	3-29
CLR—Clear .....	3-30
COM—Complement .....	3-31
CP—Compare .....	3-32
DA—Decimal Adjust .....	3-34
DEC—Decrement .....	3-37
DECW—Decrement Word .....	3-38
DI—Disable Interrupts .....	3-39
DJNZ—Decrement And Jump If Non-zero .....	3-40
EI—Enable Interrupts .....	3-42
HALT—Halt .....	3-43
INC—Increment .....	3-44
INCW—Increment Word .....	3-46
IRET—Interrupt Return .....	3-47
JP—Jump .....	3-48

---



Chapter Title and Subsection	Page
JR—Jump Relative .....	3-50
LD—Load .....	3-51
LDC—Load Constant .....	3-55
LDCl—Load Constant Auto Increment .....	3-57
NOP—No Operation .....	3-59
OR—Logical OR .....	3-60
POP—Pop .....	3-62
PUSH—Push .....	3-63
RCF—Reset Carry Flag .....	3-64
RET—Return .....	3-65
RL—Rotate Left .....	3-66
RLC—Rotate Left Through Carry .....	3-68
RLC—Rotate Left Through Carry .....	3-69
RR—Rotate Right .....	3-70
RRC—Rotate Right Through Carry .....	3-72
RRC—Rotate Right Through Carry .....	3-73
SBC—Subtract with Carry .....	3-74
SCF—Set Carry Flag .....	3-76
SRA—Shift Right Arithmetic .....	3-77
SRP—Set Register Pointer .....	3-79
STOP—Stop .....	3-81
SUB—Subtract .....	3-82
SWAP—Swap Nibbles .....	3-84
TCM—Test Complement Under Mask .....	3-85
TM—Test Under Mask .....	3-87
WDT—Watch-Dog Timer .....	3-89
XOR—Logical Exclusive OR .....	3-90

---

<b>Chapter Title and Subsections</b>	<b>Page</b>
--------------------------------------	-------------

---

**Chapter 4. Interrupts**

Introduction .....	4-1
Interrupt Sources .....	4-3
External Interrupt Sources .....	4-3
Internal Interrupt Sources .....	4-4
Interrupt Request (IREQ) Register Logic And Timing .....	4-4
Interrupt Mask Register (IMASK) Initialization .....	4-5
Interrupt Request (IREQ) Register Initialization .....	4-7
IREQ Software Interrupt Generation .....	4-9
Vectored Processing .....	4-9
Nesting of Vectored Interrupts .....	4-11
Polled Processing .....	4-12
Reset Conditions .....	4-12

**Appendix A. Accessing the ZBBS/Internet**

Bulletin Board Information

    How to Access the ZBBS

ZiLOG On The Internet

**Problem/Suggestion Report Form**

**Index**



*Totally Logical*

**LIST OF FIGURES**

<b>Chapter Title and Subsections</b>	<b>Page</b>
<b>Chapter 1. Address Space</b>	
Figure 1-1. Complete Register File RAM Space . . . . .	1-2
Figure 1-2. 16-Bit Register Addressing . . . . .	1-5
Figure 1-3. Accessing Individual Bits (Example) . . . . .	1-5
Figure 1-4. Working Register Addressing (Example) . . . . .	1-7
Figure 1-5. Register Pointer . . . . .	1-8
Figure 1-6. Program Memory Map . . . . .	1-12
Figure 1-7. Stack Pointer . . . . .	1-13
Figure 1-8. Stack Operations . . . . .	1-14
<b>Chapter 2. Addressing Modes</b>	
Figure 2-1. 8-Bit Register Addressing . . . . .	2-2
Figure 2-2. 4-Bit Register Addressing . . . . .	2-3
Figure 2-3. Indirect Addressing of Register File Memory . . . . .	2-4
Figure 2-4. Indirect Register Addressing to Program Memory . . . . .	2-5
Figure 2-5. Indexed Register Addressing . . . . .	2-6
Figure 2-6. Direct Addressing . . . . .	2-7
Figure 2-7. Retrieve Addressing . . . . .	2-8
Figure 2-8. Immediate Data Addressing . . . . .	2-9
<b>Chapter 3. Instruction Set</b>	
Figure 3-1. Flag Register . . . . .	3-5
Figure 3-2. Op Code Map . . . . .	3-18
<b>Chapter 4. Interrupts</b>	
Figure 4-1. Interrupt Control Register Addresses and Identifiers . . . . .	4-1
Figure 4-2. Interrupt Block Diagram . . . . .	4-2
Figure 4-3. Interrupt Service Sequence . . . . .	4-4
Figure 4-4. Interrupt Mask Register . . . . .	4-5
Figure 4-5. Interrupt Mask 2 Register . . . . .	4-6

---

Figure 4-6. Interrupt Request Register. . . . .	4-7
Figure 4-7. Interrupt Request Register 2 . . . . .	4-8
Figure 4-8. Stacks Before and After Interrupt . . . . .	4-10
Figure 4-9. Interrupt Vector Table Location . . . . .	4-11



*Totally Logical*

**LIST OF TABLES**

<b>Chapter Title and Subsections</b>	<b>Page</b>
<b>Chapter 1. Address Space</b>	
Table 1-1 Z8 <sup>PLUS</sup> Core Control Registers . . . . .	1-3
Table 1-1 Page 0 Register File Organization . . . . .	1-4
<b>Chapter 3. Instruction Set</b>	
Table 3-1 Load Instructions . . . . .	3-2
Table 3-2 Arithmetic Instructions . . . . .	3-2
Table 3-3 Logical Instructions . . . . .	3-2
Table 3-4 Program Control Instructions . . . . .	3-3
Table 3-5 Bit Manipulation Instructions . . . . .	3-3
Table 3-6 Block Transfer Instructions . . . . .	3-3
Table 3-7 Rotate and Shift Instructions . . . . .	3-4
Table 3-8 CPU Control Instructions . . . . .	3-4
Table 3-9 Flag Definitions . . . . .	3-7
Table 3-10 Flag Settings Definitions . . . . .	3-8
Table 3-11 Condition Codes . . . . .	3-8
Table 3-12 Notational Shorthand . . . . .	3-10
Table 3-13 Additional Symbols . . . . .	3-11
Table 3-14 Instruction Summary . . . . .	3-13
Table 3-15 Lower Nibble Values . . . . .	3-17
Table 3-16 DA Operation Reference . . . . .	3-34
Table 3-17 Register Pointers, Working Register Groups, and Actual Registers . . . . .	3-79
<b>Chapter 4. Interrupts</b>	
Table 4-1 Z8E001 Interrupt Types, Sources, and Vectors . . . . .	4-3

---

---



## CHAPTER 1

### ADDRESS SPACE

---

#### INTRODUCTION

Two address spaces are available for the Z8<sup>PLUS</sup> MCU:

- Register file RAM contains addresses for all the control registers and all the general purpose registers.
- Program memory contains addresses for all memory locations where executable code and/or data are stored.

---

#### REGISTER FILE SPACE

The on-chip register file RAM is organized into 16 pages, where each page has 256 addressable memory locations. The first page (page 0) contains both control registers and general purpose registers. All the remaining pages (pages 1 through 15) contain only general purpose registers. Figure 1-1 illustrates the complete register file RAM space. As shown, control registers are located in the upper half of page 0. Any specific implementation of the Z8<sup>PLUS</sup> core may use only a subset of the complete register file RAM space.

Table 1-1 describes the Core Control Registers and Table 1-2 shows the Page 0 Register File organization.

All registers on the Z8<sup>PLUS</sup>-family products are fully read/writable. Hardware may write lock certain registers or bits under some conditions. The TCTLHI register is one such example.

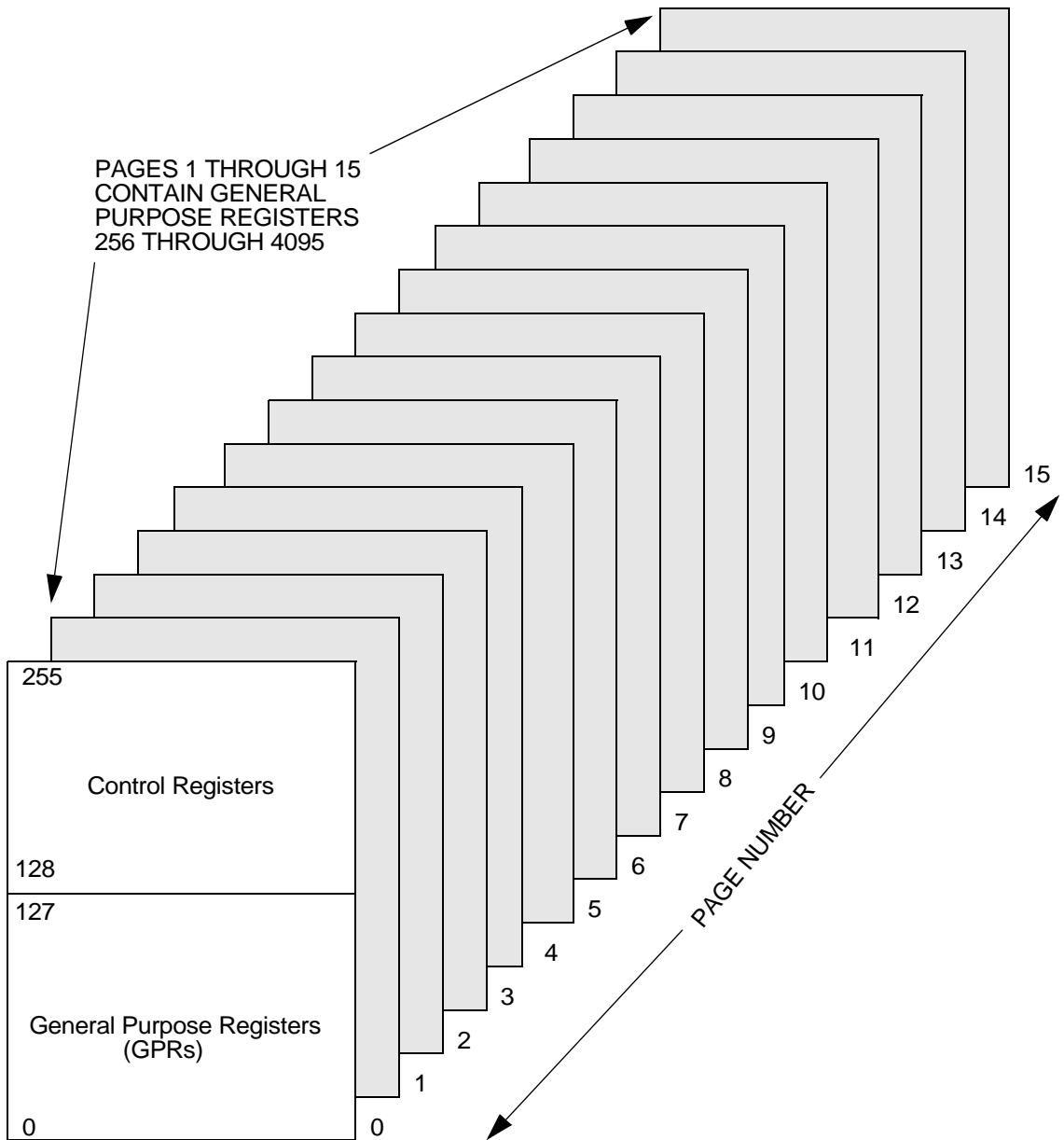


Figure 1-1. Complete Register File RAM Space



Table 1-1. Z8<sup>PLUS</sup> Core Control Registers

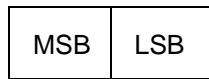
Hex Address	Register Name	Register Description	Comments
0FFH	STKPTR (SPL)	Stack Pointer Low	LSB of Stack Pointer
0FEH	SPH	Stack Pointer High	MSB of Stack Pointer
0FDH	REGPTR(RP)	Register Pointer	
0FCH	FLAGS	Flags	
0FBH	IMASK	Interrupt Mask 1	Ints. 0 - 6
0FAH	IREQ	Interrupt Request 1	Ints. 0 - 6
0F9H	IMASK2	Interrupt Mask 2	Ints. 7 - 14
0F8H	IREQ2	Interrupt Request 2	Ints. 7 - 14
0F7H			Reserved
0F6H			Reserved
0F5H			Reserved
0H4H			Reserved
0F3H			Reserved
0F2H			Reserved
0F1H			Reserved
0F0H			Reserved

The Stack Pointer High register (0FEH), the interrupt mask register 2 (0F9H), and the interrupt request register 2 (0F8H) are optional and are reserved if not implemented.

**Table 1-2. Page 0 Register File Organization**

Hex Address Range	Register Description
F0 - FF	Core Control Registers
E0 - EF	Virtual Copy of the Current Working Register Set
D0 - DF	Port Logic Control Registers
C0 -CF	Timer Peripherals Control Registers
B0 - BF	Reserved for Future Extensions
A0 - AF	Reserved for Future Extensions
90 - 9F	Reserved for Future Extensions
80 - 8F	Reserved for Future Extensions
70 - 7F	General Purpose Registers
60 - 6F	General Purpose Registers
50 - 5F	General Purpose Registers
40 - 4F	General Purpose Registers
30 - 3F	General Purpose Registers
20 - 2F	General Purpose Registers
10 -1F	General Purpose Registers
00 - 0F	General Purpose Registers

Registers can be accessed as either 8-bit or 16-bit registers using Direct, Indirect, or Indexed Addressing. All general-purpose registers can be referenced or modified by any instruction that accesses an 8-bit register, without the need for special instructions. Registers accessed as 16 bits are treated as even-odd register pairs. In this case, the data's Most Significant Byte (MSB) is stored in the even numbered register, while the Least Significant Byte (LSB) goes into the next higher odd numbered register (Figure 1-2).

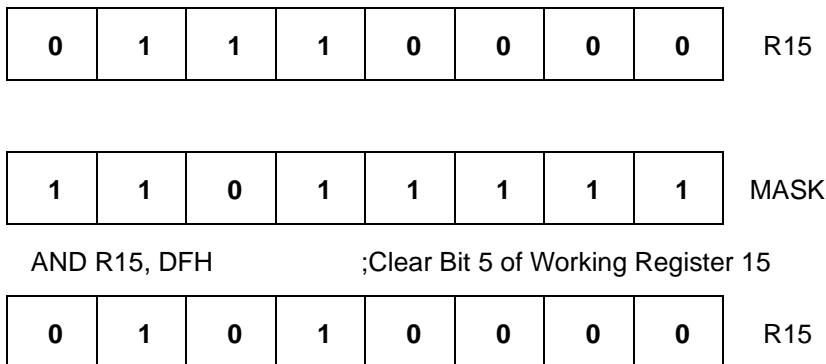


Rn      Rn+1

n = Even  
Address

**Figure 1-2. 16-Bit Register Addressing**

By using a logical instruction and a mask, individual bits within registers can be accessed for bit set, bit clear, bit complement, or bit test operations. For example, the instruction `AND R15, MASK` performs a bit clear operation. Figure 1-3 shows this example.



**Figure 1-3. Accessing Individual Bits (Example)**

When instructions are executed, registers are only read, not written, when defined as sources; and read and/or written when defined as destinations. All General-Purpose Registers function as accumulators, address pointers, index registers, stack areas, or scratch pad memory.

## General-Purpose Registers

General-Purpose Registers (GPR) are undefined after the device is powered up. The registers keep their last value after any reset, as long as the reset occurs in the  $V_{CC}$  voltage-specified operating range. It does not keep its last state from a  $V_{LV}$  reset if  $V_{CC}$  drops below 1.8V.

## Working Register Groups

Instructions can access 8-bit registers and register pairs (16-bit words) using either 4-, 8-, or 12-bit address fields. Eight-bit address fields refer to the actual address of the register within the current page. For example, Register 58H is accessed by calling upon its 8-bit address, 01011000 (58H). The lower nibble of the Register Pointer specifies the current RAM page.

With 4-bit addressing, the register file is logically divided into 16 Working Register Groups of 16 registers each, as shown in Table 1-3. These 16 registers are known as Working Registers. A Register Pointer (one of the control registers, FDH) contains the base address of the active Working Register Group. The High nibble of the Register Pointer determines the current Working Register Group.

When accessing one of the Working Registers, the 4-bit address of the Working Register is combined with the upper four bits (High nibble) of the Register Pointer, thus forming the 8-bit actual address. Figure 1-4 illustrates this operation. Since working registers are typically specified by short format instructions, there are fewer bytes of code needed. In addition, when processing interrupts or changing tasks, the Register Pointer (see Figure 1-5) speeds context switching. A special Set Register Pointer (SRP) instruction sets the contents of the Register Pointer.

Data transfer across RAM page boundaries can be accomplished via 12-bit addressing. Using certain instruction modes, data can be moved from the current page and working group into any register on the chip by specifying the absolute 12-bit address, including page. Not all family members support 12-bit addressing. See the applicable product specification for specific information.

**Table 1-3. Working Register Groups**

Register Pointer (FDH) High Nibble (Binary)	Working Register Group (HEX)	Actual Registers (HEX)
1111	F	F0 - FF
1110	E	E0 - EF
1101	D	D0 - DF
1100	C	C0 - CF
1011	B	B0 - BF
1010	A	A0 - AF
1001	9	90 - 9F
1000	8	80 - 8F
0111	7	70 - 7F

Table 1-3. Working Register Groups (Continued)

Register Pointer (FDH) High Nibble (Binary)	Working Register Group (HEX)	Actual Registers (HEX)
0110	6	60 - 6F
0101	5	50 - 5F
0100	4	40 - 4F
0011	3	30 - 3F
0010	2	20 - 2F
0001	1	10 - 1F
0000	0	00 - 0F

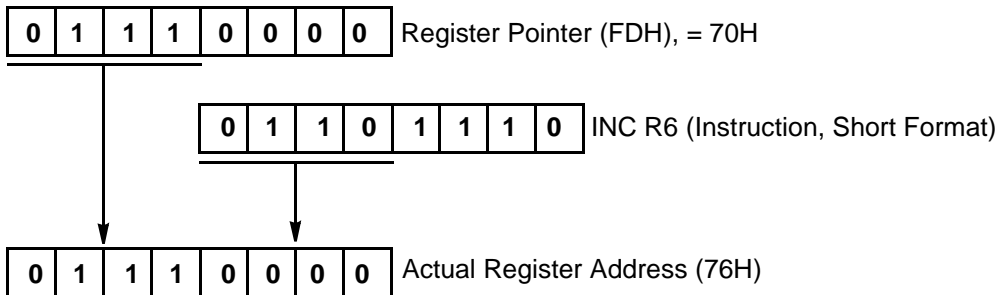


Figure 1-4. Working Register Addressing (Example)

The upper nibble of the register file address, provided by the register pointer, specifies the active working-register group.

The lower nibble specifies the current page of RAM.

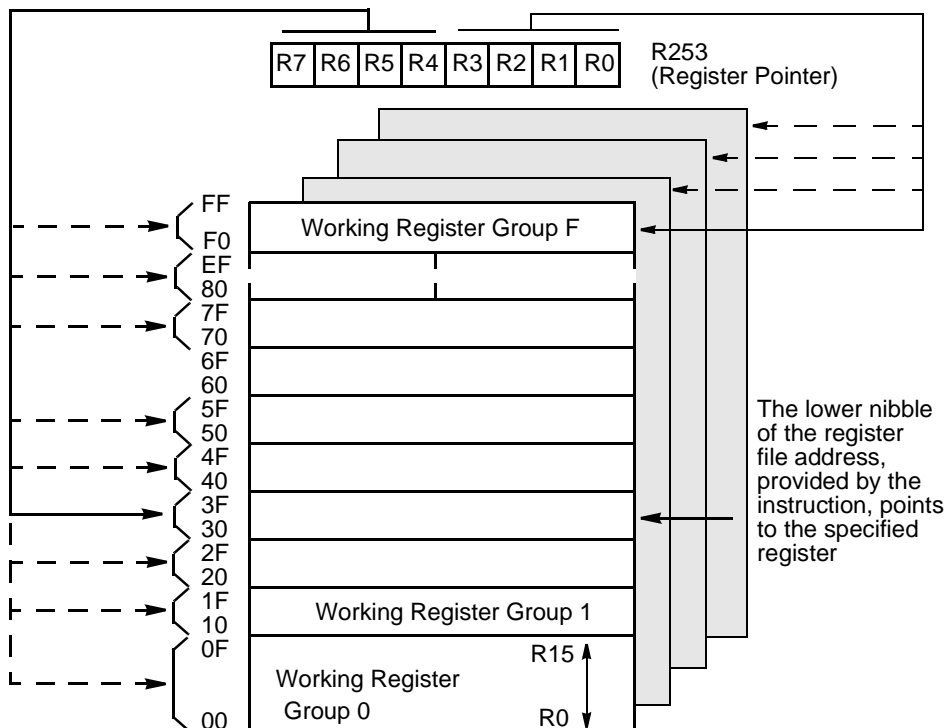


Figure 1-5. Register Pointer

## Precautions

Registers in the Standard Register File must be correctly used or certain conditions produce inconsistent results.

- The watch-dog timer can only be disabled via software if the first instruction out of RESET performs this function. During the execution of the first instruction after the Z8<sup>PLUS</sup> leaves RESET, the upper five bits of the TCTLHI register can be written. After the first instruction, hardware does not allow the upper five bits of this register to be written.
- Some control registers, including the port inputs and timer count registers, may be updated by hardware. Writing these registers from software always overrides the hardware update from the same cycle, but with unpredictable results. For example, writing into the count value register of a running timer can cause

unexpected results if the hardware was in the process of decrementing the timer for the terminal count and generating an interrupt.

- The register space from 0E0H–0EFH is special. The MCU uses these addresses to flag accesses via 4-bit addressing mode to the current working register group. There are no physical registers at that location. Care must be taken that the Register Pointer never points at Group E on the first page (be loaded with E0H). This is an undefined case. Also, indirect addressing does *not* redirect a second time and find the working registers. This is also an undefined case. As an example, in the code below, R0 does *not* find the data in register 08. It returns garbage. R2 correctly contains a copy of register 08.

```
SRP          #%00

LD           R1, #%E8

LD           R0, @R1

LD           R2, %E8
```

---

## CONTROL AND PERIPHERAL REGISTERS

### Control Registers

The standard control registers govern the operation of the CPU. Any instruction which references the register file can access these control registers. Available control registers are:

- Stack Pointer Low (SPL or STKPTR)
- Stack Pointer High (SPH)
- Register Pointer (RP or REGPTR)
- Flags (FLAGS)
- Interrupt Mask 1 (IMASK)
- Interrupt Request 1 (IREQ)
- Interrupt Mask 2 (IMASK2)
- Interrupt Request 2 (IREQ2)

A 16-bit Program Counter (PC) to determine the sequence of current program instructions. The PC is not an addressable register.

### Peripheral Registers

Peripheral registers are used to transfer data, configure the operating mode, and control the operation of the on-chip peripherals. Any instruction that references the register file can access the peripheral registers. Possible peripheral registers can include:

- Timer Count Value Register for Timer  $n$
- Auto-Initialization Value Register(s) for Timer  $n$
- Timer Control Registers (High and Low Byte)
- Watch-Dog Timer Registers (High and Low Byte)

In addition, the port registers are considered to be peripheral registers. Ports generally have at least the following four dedicated registers which are readable and writable by software:

- Port Input Value Register
- Port Output Value Register
- Port Control Register
- Port Special Function Register



## **PROGRAM MEMORY**

The program memory map is shown in Figure 1-6. The first two bytes of program memory are reserved for the PC rollover vector. When the PC wraps around to 0000H, bytes 0000H and 0001H are executed as instructions, enabling a user defined behavior for this occurrence. For example, a JR instruction in 0000H and a corresponding displacement in 0001H could be defined for the PC rollover vector. The next 30 bytes of Program Memory are reserved for the interrupt vectors. These locations contain 16-bit vectors that correspond to the available interrupts. Address 0020H through the end of the populated memory (0FFFFh, 64 KB maximum) consists of on-chip mask-programmable ROM or EPROM or Flash. The first byte of program memory executed following a RESET is located at 0020H. See the product data sheet for the exact program, data, register memory size, and address range available.

The internal program memory may be one-time programmable (OTP) or mask programmable dependent on the specific device. A ROM protect feature prevents dumping of the ROM contents. The ROM Protect option is mask-programmable and is selected by the customer when the ROM code is submitted. For programmable memory devices, the ROM Protect option is an OTP programming option.

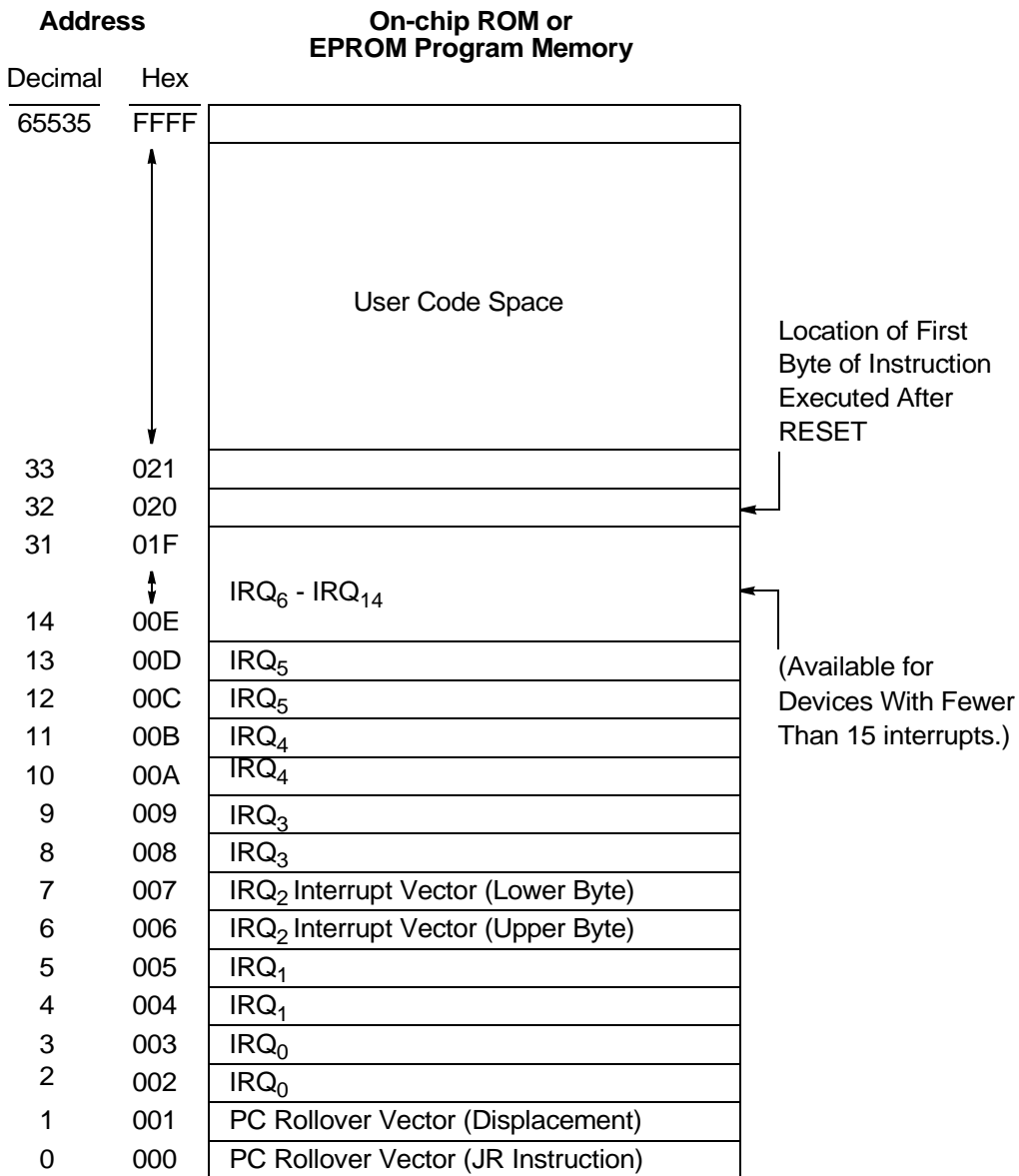
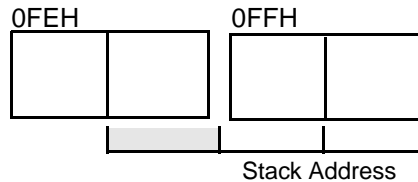


Figure 1-6. Program Memory Map

## STACK

The stack always resides in the general purpose registers of the on-chip register file RAM. The stack pointer register (SP) contains an address into the standard register file that is the address of the operand that is currently on the top of the stack. The register 0FEH is the 8-bit stack pointer (SP), that is used for all stack operations (see Figure 1-7).

Some devices prepend the lower nibble of register 0FEH to form a 12-bit stack pointer. Otherwise, register 0FEH is reserved.

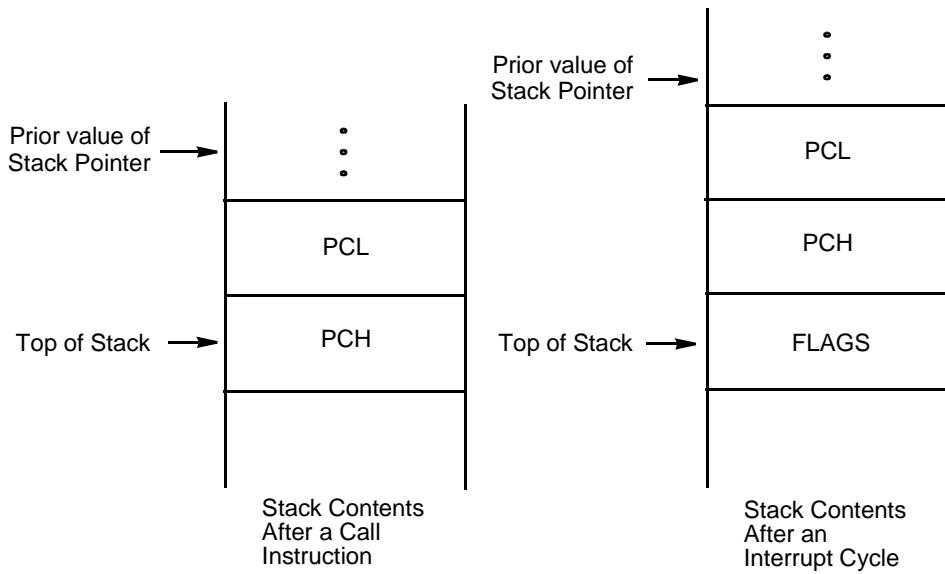


**Figure 1-7. Stack Pointer**

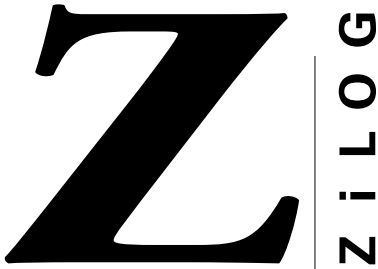
The stack address is decremented prior to a PUSH operation and incremented after a POP operation. The stack address always points to the data stored on the top of the stack. The stack is a return stack for CALL instructions and interrupts, as well as a data stack.

During a CALL instruction, the contents of the Program Counter are saved on the stack. The PC is restored during a RET instruction. Interrupts cause the contents of the PC and FLAGS registers to be saved on the stack. The IRET instruction restores them (see Figure 1-8).

An overflow or underflow can occur when the stack address is incremented or decremented during normal stack operations. The programmer must prevent this occurrence or unpredictable operation may result. The stack must not encroach into the control registers.



**Figure 1-8. Stack Operations**



## CHAPTER 2

### ADDRESSING MODES

---

#### ADDRESSING MODES

The Z8<sup>PLUS</sup> microcontroller provides six addressing modes:

- Register (R)
- Indirect Register (IR)
- Indexed (X)
- Direct Address (DA)
- Relative Address (RA)
- Immediate Data (IM)

With the exception of immediate data and condition codes, all operands are expressed as register file or Program Memory addresses. Registers are accessed using 12-bit addresses in the range of 000H–FFFH. The Program Memory is accessed using 16-bit addresses (or register pairs) in the range of 0000H–FFFFH.

Generally, registers are accessed, within the current page, by specifying an 8-bit address. The upper 4 bits of the absolute address is specified by pre-pending the lower 4 bits of the Register Pointer (0FDH) (the Page Pointer) to the 8-bit address to form a 12-bit address.

Working Registers are accessed using 4-bit addresses in the range of 0-15 (0H–FH). The address of the register being accessed is formed by the combination of the lower 4 bits of the RP (Page Pointer), the upper four bits in the Register Pointer (Group Pointer) and the 4-bit working register address supplied by the instruction.

Registers can be used in pairs to designate 16-bit values or memory addresses. A Register Pair must be specified as an even-numbered address in the range of 0–14 for Working Registers, or 0–4094 for general purpose registers.

In the following definitions of Z8<sup>PLUS</sup> Addressing Modes, the use of register can also imply register pair, working register, or working register pair, depending on the context.

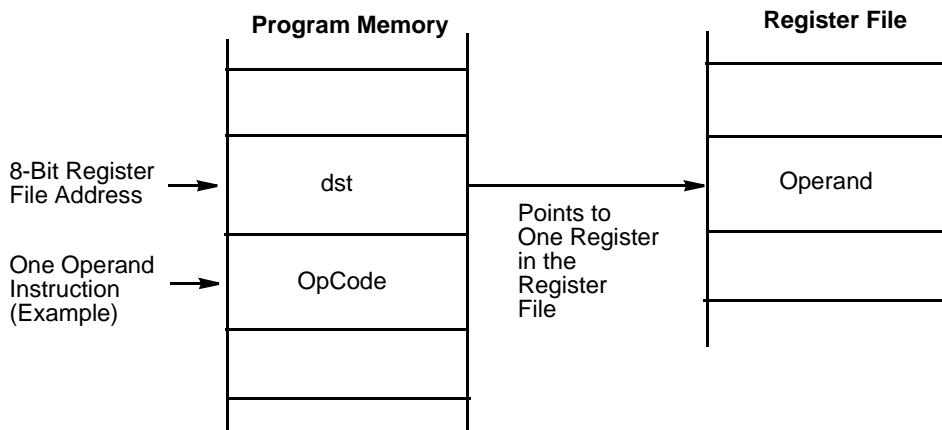
**NOTE:** See the product data sheet for exact program and register memory types and address ranges available.

---

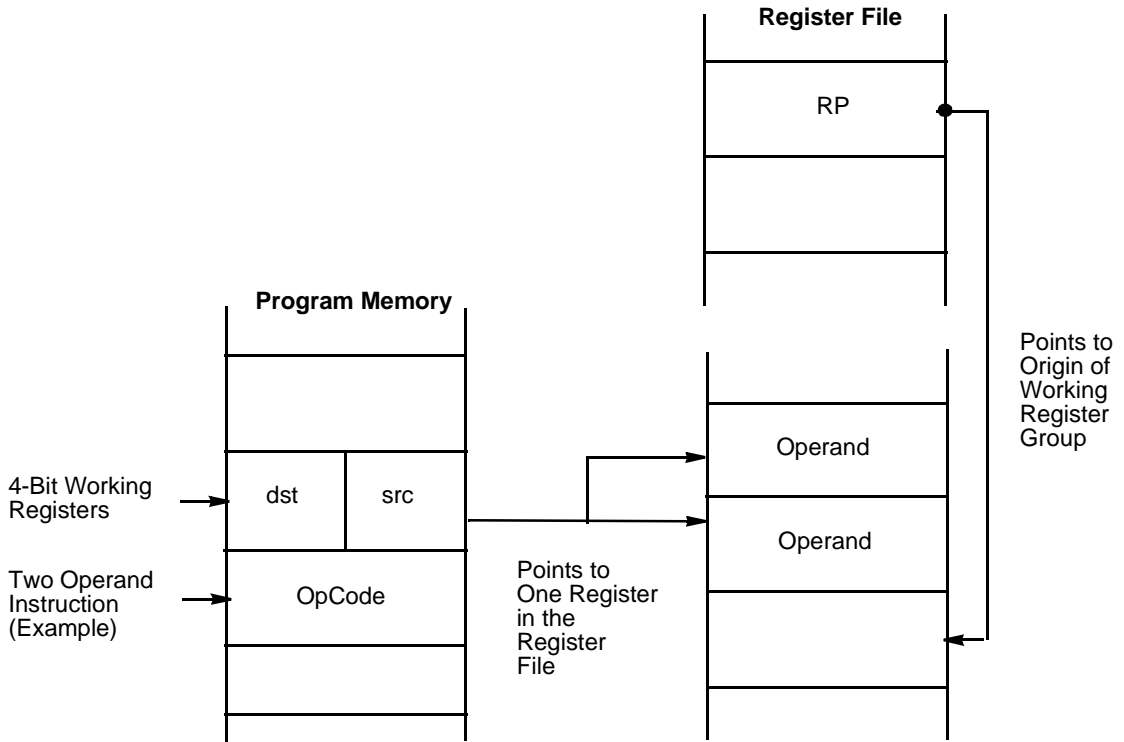
## **REGISTER ADDRESSING (R)**

In 8-bit Register Addressing mode, the operand value is equivalent to the contents of the specified register or register pair.

In the Register Addressing (see Figure 2-1), the destination and/or source address specified corresponds to the actual register in the current page of the register file.



**Figure 2-1. 8-Bit Register Addressing**



**Figure 2-2. 4-Bit Register Addressing**

In 4-bit Register Addressing (see Figure 2-2), the destination and/or source addresses point to the Working Register within the current Working Register Group. This 4-bit address is combined with the Register Pointer to form the actual 12-bit address of the affected register.

## INDIRECT REGISTER ADDRESSING (IR)

In the Indirect Register Addressing Mode, the contents of the specified register are equivalent to the address of the operand (see Figure 2-3 and Figure 2-4).

Depending upon the instruction selected, the specified register contents points to a Register or Program Memory location.

When accessing program memory, register pairs or Working Register pairs are used to hold the 16-bit addresses.

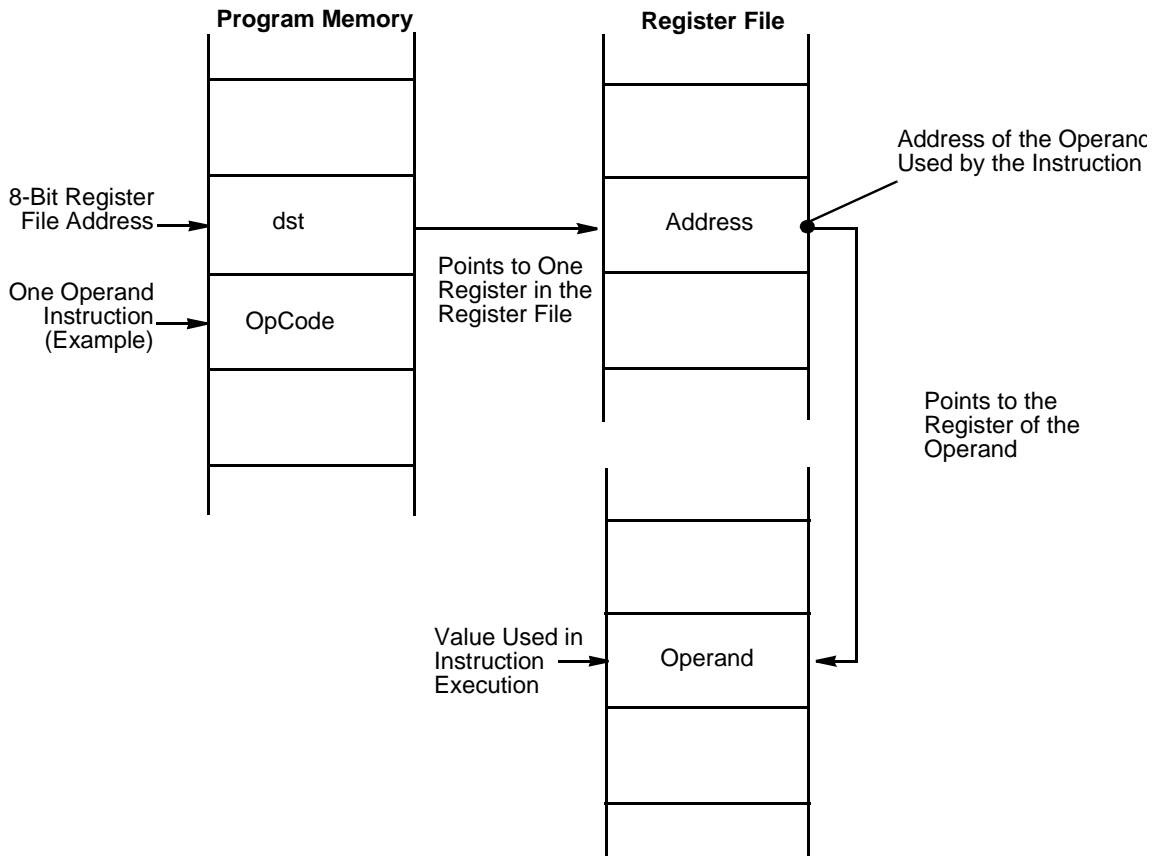


Figure 2-3. Indirect Addressing of Register File Memory



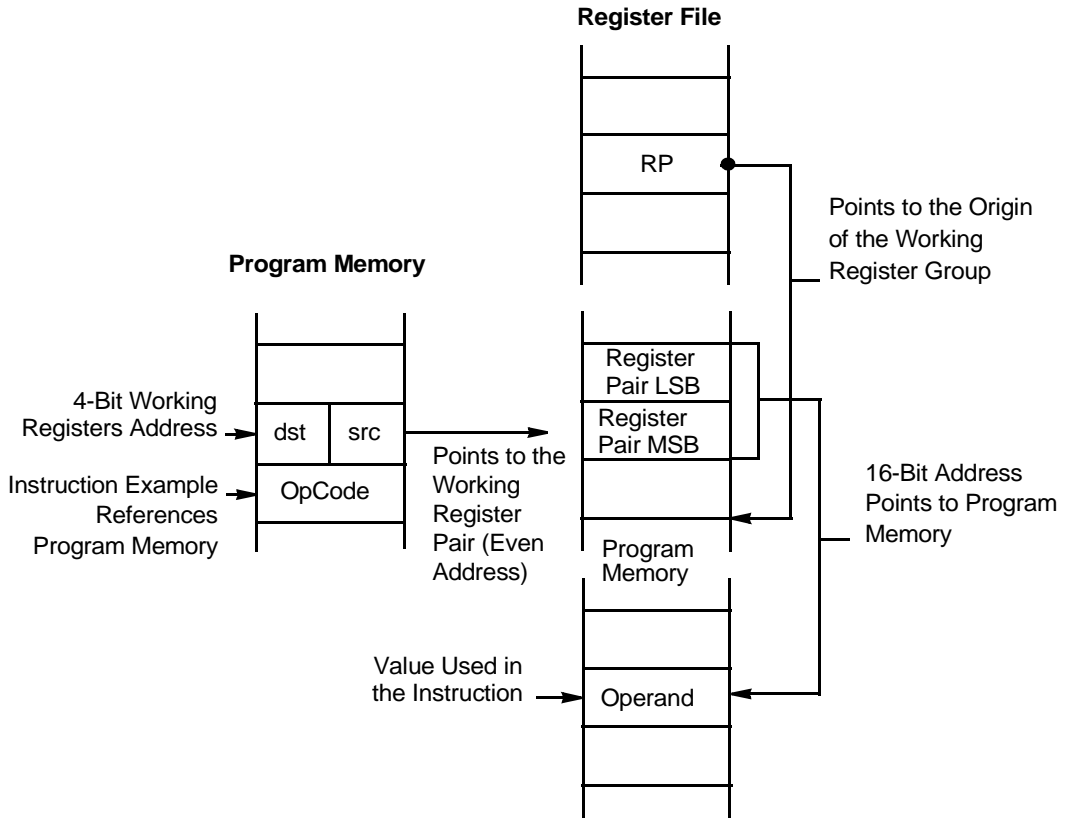


Figure 2-4. Indirect Register Addressing to Program Memory

## INDEXED ADDRESSING (X)

The Indexed Addressing Mode is used only by the Load (LD) instruction. An indexed address consists of a register address offset by the contents of a designated Working Register (the Index). This offset is added to the register address to obtain the address of the operand. Figure 2-5 illustrates this addressing convention.

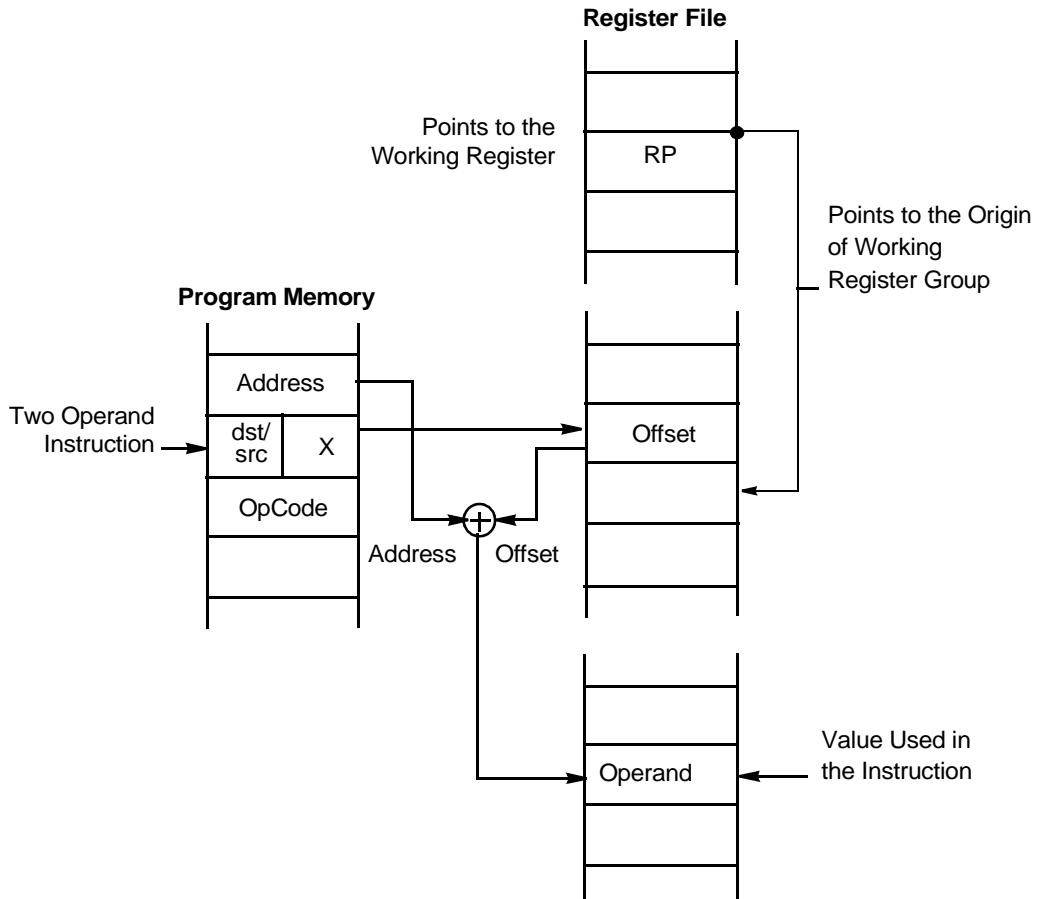
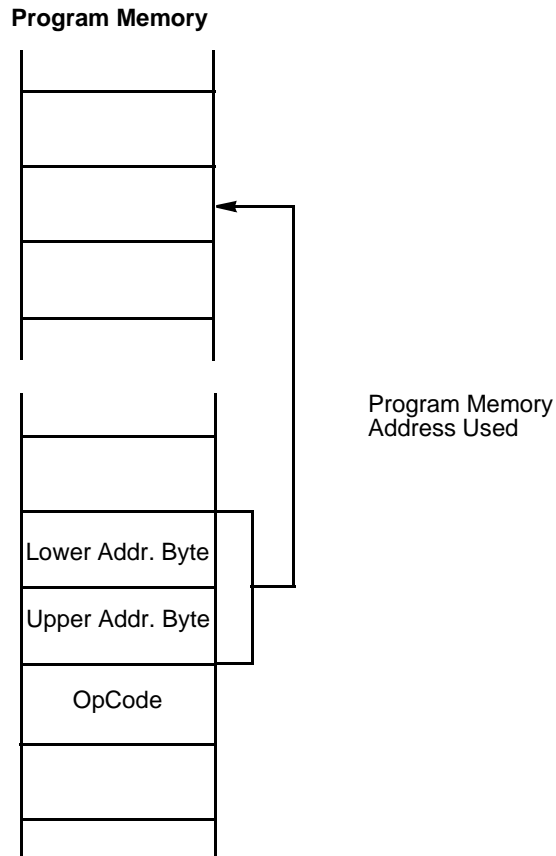


Figure 2-5. Indexed Register Addressing

## DIRECT ADDRESSING (DA)

The Direct Addressing mode, as shown in Figure 2-6, specifies the address of the next instruction to be executed. Only the Conditional Jump (JP) and Call (CALL) instructions use this addressing mode.



**Figure 2-6. Direct Addressing**

## RELATIVE ADDRESSING (RA)

In the Relative Addressing mode, illustrated in Figure 2-7, the instruction specifies a two's-complement signed displacement in the range of  $-128$  to  $+127$ . This is added to the contents of the Program Counter to obtain the address of the next instruction to be executed. The PC (prior to the add) consists of the address of the instruction following the Jump Relative (JR) or Decrement and Jump if Non-Zero (DJNZ) instruction. JR and DJNZ are the only instructions which use this addressing mode.

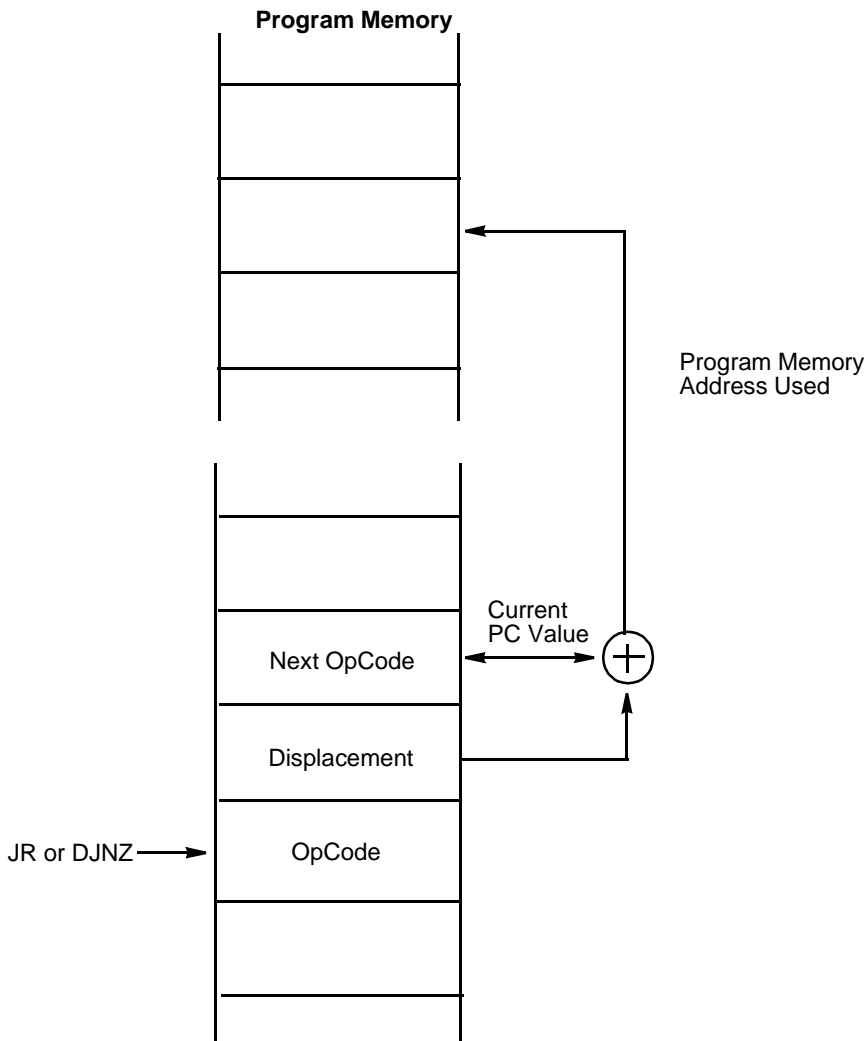
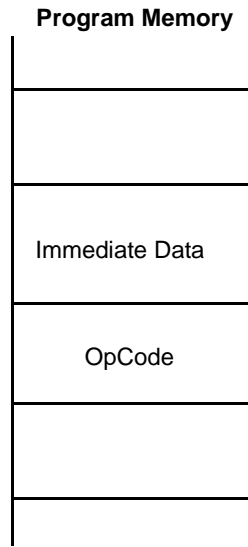


Figure 2-7. Retrieve Addressing

## IMMEDIATE DATA ADDRESSING (IM)

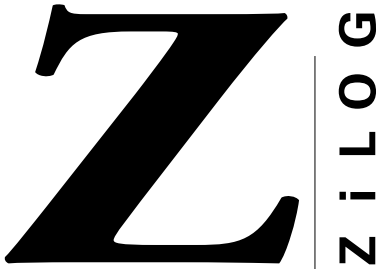
Immediate data is considered to be an addressing mode for the purposes of this discussion. It is the only addressing mode that does not indicate a register or memory address as the source operand. The operand value used by the instruction is the value supplied in the operand field itself. Because an immediate operand is part of the instruction, it is always located in the Program Memory address space (see Figure 2-8).



**Figure 2-8. Immediate Data Addressing**

---

---



*Totally Logical*

## CHAPTER 3 INSTRUCTION SET

---

### FUNCTIONAL SUMMARY

Z8<sup>PLUS</sup> instructions can be divided into the following eight functional groups:

- Load
- Arithmetic
- Logical
- Program Control
- Bit Manipulation
- Block Transfer
- Rotate and Shift
- CPU Control

Table 3-1 through Table 3-8 show the instructions belonging to each group and the number of operands required for each. The source operand is *src*, the destination operand is *dst*, and a condition code is *cc*.

When instructions are executed, registers defined as sources are read only. All General-Purpose Registers function as:

- accumulators
- address pointers
- index registers
- stack areas
- scratch pad memory

**Table 3-1. Load Instructions**

Mnemonic	Operands	Instruction
CLR	dst	Clear
LD	dst, src	Load
LDC	dst, src	Load Constant
POP	dst	Pop
PUSH	src	Push

**Table 3-2. Arithmetic Instructions**

Mnemonic	Operands	Instruction
ADC	dst, src	Add with Carry
ADD	dst, src	Add
CP	dst, src	Compare
DA	dst	Decimal Adjust
DEC	dst	Decrement
DECW	dst	Decrement Word
INC	dst	Increment
INCW	dst	Increment Word
SBC	dst, src	Subtract with Carry
SUB	dst, src	Subtract

**Table 3-3. Logical Instructions**

Mnemonic	Operands	Instruction
AND	dst, src	Logical AND
COM	dst	Complement
OR	dst, src	Logical OR
XOR	dst, src	Logical Exclusive OR



**Table 3-4. Program Control Instructions**

<b>Mnemonic</b>	<b>Operands</b>	<b>Instruction</b>
CALL	dst	Call Procedure
DJNZ	dst, src	Decrement and Jump Non-Zero
IRET		Interrupt Return
JP	cc, dst	Jump
JR	cc, dst	Jump Relative
RET		Return

**Table 3-5. Bit Manipulation Instructions**

<b>Mnemonic</b>	<b>Operands</b>	<b>Instruction</b>
TCM	dst, src	Test Complement Under Mask
TM	dst, src	Test Under Mask
AND	dst, src	Bit Clear
OR	dst, src	Bit Set
XOR	dst, src	Bit Complement

**Table 3-6. Block Transfer Instructions**

<b>Mnemonic</b>	<b>Operands</b>	<b>Instruction</b>
LDCI	dst, src	Load Constant Auto Increment

**Table 3-7. Rotate and Shift Instructions**

<b>Mnemonic</b>	<b>Operands</b>	<b>Instruction</b>
RL	dst	Rotate Left
RLC	dst	Rotate Left Through Carry
RR	dst	Rotate Right
RRC	dst	Rotate Right Through Carry
SRA	dst	Shift Right Arithmetic
SWAP	dst	Swap Nibbles

**Table 3-8. CPU Control Instructions**

<b>Mnemonic</b>	<b>Operands</b>	<b>Instruction</b>
CCF		Complement Carry Flag
DI		Disable Interrupts
EI		Enable Interrupts
HALT		Halt
NOP		No Operation
RCF		Reset Carry Flag
SCF		Set Carry Flag
SRP	src	Set Register Pointer
STOP		Stop
WDT		Refresh WDT

## PROCESSOR FLAGS

The Flag Register (FCH) informs the user of the processor's current status. The flags and their bit positions in the Flag Register are shown in Figure 3-1.

The Flag Register contains eight bits of status information which are set or cleared by CPU operations. Four of the bits (C, V, Z and S) can be tested for use with conditional Jump instructions. Two flags (H and D) are used for BCD arithmetic. The two remaining bits in the Flag Register are the watch-dog timer reset flag and the stop mode recovery flag. Both of these flag bits may be tested and must be explicitly cleared by software.

As with bits in the other control registers, the Flag Register bits can be set or reset by instructions; however, only those instructions that do not affect the flags as an outcome of the execution should be assigned a value.

**Figure 3-1. Flag Register**

### Flag Register (FCH: Read/Write) R252 Flags

Bit	7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	U	U	U	U	U	U	*	*
R = Read W = Write X = Indeterminate U = Unchanged								

Bit/Field	Bit Position	R/W	Value	Description
Carry Flag (C)	7	R/W		<p>The Carry Flag is set to 1 whenever the result of an arithmetic operation generates a carry out of or a borrow into the high order bit 7. Otherwise, the Carry Flag is cleared to 0. Following Rotate and Shift instructions, the Carry Flag contains the last value shifted out of the specified register.</p> <p>An instruction can set (I), reset(O), or complement the Carry Flag.</p> <p>The carry flag is not effected by RESET.</p>
Zero Flag (Z)	6	R/W		<p>For arithmetic and logical operations, the Zero Flag is set to 1 if the result is 0. Otherwise, the Zero Flag is cleared to 0.</p> <p>If the result of testing bits in a register is 00H, the Zero Flag is set to 1. Otherwise the Zero Flag is cleared to 0.</p> <p>If the result of a Rotate or Shift operation is 00H, the Zero Flag is set to 1.</p> <p>The Zero Flag is not effected by a RESET command.</p>

SignFlag (S)	5	R/W	<p>The Sign Flag stores the value of the most significant bit of a result following an arithmetic, logical, rotate, or shift operation.</p> <p>When performing arithmetic operations on signed numbers, binary two's-complement notation is used to represent and process information. A positive number is identified by a 0 in the most significant bit position (bit 7); therefore, the Sign Flag is also 0.</p> <p>A negative number is identified by a 1 in the most significant bit position (bit 7); therefore, the Sign Flag is also 1.</p> <p>The Sign Flag is not effected by RESET.</p>
Overflow (V)	4	R/W	<p>For signed arithmetic, rotate, and shift operations, the Overflow Flag is set to 1 when the result is greater than the maximum possible number (&gt;127) or less than the minimum possible number (&lt;-128) that can be represented in two's-complement form. The Overflow Flag is cleared to 0 if no overflow occurs.</p> <p>Following logical operations the Overflow Flag is cleared to 0.</p> <p>The Overflow Flag is not effected by RESET.</p>
Decimal Adjust Flag (D)	3	R/W	<p>The Decimal Adjust Flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag specifies what type of instruction was last executed so that the subsequent Decimal Adjust (DA) operation can function properly. Normally, the Decimal Adjust Flag cannot be used as a test condition.</p> <p>After a subtraction, the Decimal Adjust Flag is set to 1. Following an addition it is cleared to 0.</p> <p>The Decimal Adjust Flag is not effected by RESET.</p>
Half-Carry Flag (H)	2	R/W	<p>The Half Carry Flag is set to 1 whenever an addition generates a carry out of bit 3 (Overflow) or a subtraction generates a "borrow into" bit 3. The Half Carry Flag is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. As in the case of the Decimal Adjust Flag, the user does not normally access this flag.</p> <p>The Half Carry flag is not effected by RESET.</p>

Watch-Dog Timer (WDT)	1	R/W	<p>The Watch-Dog Timer reset flag is set by a watchdog timer timeout. This permits software to determine if a timeout of the watchdog timer has occurred.</p> <p>The WDT flag is cleared by the <math>\overline{\text{RESET}}</math> pin. The WDT and SMR flags are the only flags effected by RESET. This behavior permits software to determine if a RESET occurred, if a WDT timeout occurred, or if a return from STOP mode occurred.</p> <p>Software must explicitly clear this flag after detecting the timeout condition.</p> <p>Failure to clear this flag may result in undefined behavior.</p>
Stop Mode Recovery Flag (SMR)	0	R/W	<p>The Stop Mode Recovery (SMR) flag is set upon the execution of a STOP instruction. This permits software to determine if a return from stop mode has occurred upon returning to active status.</p> <p>The SMR flag is cleared by the <math>\overline{\text{RESET}}</math> pin. The WDT and SMR flags are the only flags effected by RESET. This behavior permits software to determine if a RESET occurred, if a WDT timeout occurred, or if a return from STOP mode occurred.</p> <p>Software must explicitly clear this flag after detecting the SMR condition.</p> <p>Failure to clear this flag may result in undefined behavior.</p>

## CONDITION CODES

The C, Z, S, and V Flags control the operation of the conditional JUMP instructions. Sixteen frequently useful functions of the flag settings are encoded in a 4-bit field called the condition code (CC), which forms bits 4-7 of the conditional instructions.

Flag Definitions, Flag Settings and Condition Codes are summarized in Table 3-9, Table 3-10, and Table 3-11.

**Table 3-9. Flag Definitions**

Flag	Description
C	Carry Flag
Z	Zero Flag
S	Sign Flag
V	Overflow Flag

**Table 3-10. Flag Settings Definitions**

Symbol	Definition
0	Cleared to 0
1	Set to 1
*	Set or cleared according to operation
–	Unaffected
X	Undefined

**Table 3-11. Condition Codes**

Binary	HEX	Mnemonic	Definition	Flag Settings
0000	0	F	Always False	–
1000	8	(blank)	Always True	–
0111	7	C	Carry	C = 1
1111	F	NC	No Carry	C = 0
0110	6	Z	Zero	Z = 1
1110	E	NZ	Non-Zero	Z = 0
1101	D	PL	Plus	S = 0
0101	5	MI	Minus	S = 1
0100	4	OV	Overflow	V = 1
1100	C	NOV	No Overflow	V = 0

Table 3-11. Condition Codes (Continued)

Binary	HEX	Mnemonic	Definition	Flag Settings
0110	6	EQ	Equal	$Z = 1$
1110	E	NE	Not Equal	$Z = 0$
1001	9	GE	Greater Than or Equal	$(S \text{ XOR } V) = 0$
0001	1	LT	Less Than	$(S \text{ XOR } V) = 1$
1010	A	GT	Greater Than	$(Z \text{ OR } (S \text{ XOR } V)) = 0$
0010	2	LE	Less Than or Equal	$(Z \text{ OR } (S \text{ XOR } V)) = 1$
1111	F	UGE	Unsigned Greater Than or Equal	$C = 0$
0111	7	ULT	Unsigned Less Than	$C = 1$
1011	B	UGT	Unsigned Greater Than	$(C = 0 \text{ AND } Z = 0) = 1$
0011	3	ULE	Unsigned Less Than or Equal	$(C \text{ OR } Z) = 1$

## NOTATION AND BINARY ENCODING

The operands and status flags use a notational shorthand. Operands, condition codes, address modes, and their notations are described in Table 3-12.

**Table 3-12. Notational Shorthand**

Notation	Address Mode	Operand	Range*
cc	Condition Code		See Table 3-11, condition codes
r	Working Register	Rn	n = 0 – 15
R	Register or Working Register	Reg  Rn	Reg. represents a number in the range of 00H to FFH n = 0 – 15
RR	Indirect Register Pair or Working Register Pair	Reg  RRp	p = 0, 2, 4, 6, 8, 10, 12, or 14
Ir	Indirect Working Register	@Rn	n = 0 – 15
IR	Indirect Register or Indirect Working Register	@Reg  @Rn	Reg. represents a number in the range of 00H to FFH n = 0– 15
Irr	Indirect Working Register Pair	@RRp	p = 0, 2, 4, 6, 8, 10, 12, or 14
IRR	Indirect Register Pair or Working Register Pair	@Reg  @RRp	Reg. represents an even number in the range 00H to FFH p=0, 2, 4, 6, 8, 10, 12, or 14
X	Indexed	Reg (Rn)	Reg. represents a number in the range of 00H to FFH n = 0 – 15
DA	Direct Address	Addr <sub>s</sub>	Addr <sub>s</sub> . represents a number in the range of 0000H to FFFFH
RA	Relative Address	Addr <sub>s</sub>	Addr <sub>s</sub> . represents a number in the range of +127 to –128 which is an offset relative to the address of the next instruction
IM	Immediate	#Data	Data is a number between 00H to FFH

\*See the device product specification to determine the exact register file range available. The register file size varies by the device type.



Table 3-13, which follows, describes additional symbols used.

**Table 3-13. Additional Symbols**

Symbol	Definition
dst	Destination Operand
src	Source Operand
@	Indirect Address Prefix
SP	Stack Pointer
PC	Program Counter
FLAGS	Flag Register (FCH)
RP	Register Pointer (FDH)
IMR	Interrupt Mask Register (FBH)
#	Immediate Operand Prefix
%	Hexadecimal Number Prefix
H	Hexadecimal Number Suffix
B	Binary Number Suffix
OPC	op code

Assignment of a value is indicated by the symbol  $\leftarrow$ , for example:

```
dst ← dst + src
```

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

The notation `addr (n)` is used to refer to bit 'n' of a given location. The following example refers to bit 7 of the destination operand.

```
dst (7)
```

Some instructions operate with several addressing modes. This situation is indicated by an op code number written like `x[ ]`. The brackets are filled by a nibble indicating the addressing mode in use. For example, `ADD 0[ ]` indicates that the ADD instruction works identically for more than one addressing mode.

## Assembly Language Syntax

For proper instruction execution, assembly language syntax requires that the destination and source be specified as *dst*, *src* (in that order). The following instruction descriptions show the format of the object code produced by the assembler. This binary format should be followed by users who prefer manual program coding or who intend to implement their own assembler. Other third party assemblers can differ. Please consult the software user's manual for detailed information.

**Example:** The contents of registers 43H and 08H are added, and the result is stored in 43H. The assembly syntax and resulting object code are:

```
ASM:      ADD    43H,    08H    (ADD dst, src)
OBJ:      04     08     43     (OPC src, dst)
```

In general, whenever an instruction format requires an 8-bit register address, that address can specify any register location in the range 0 - 255. When using working registers (R0-R15), a 4-bit address is used. If a working register is used and an 8-bit address is required by the assembler, an E is pre-pended to the 4-bit working register address. If, in the above example, the source register is a working register, the assembly syntax and resulting object code are:

```
ASM:      ADD    43H,    R8     (ADD dst, src)
OBJ:      04     E8     43     (OPC src, dst)
```

### NOTES:

1. Note that the 4-bit address R8 was expanded to 8-bits by pre-pending EH. This expansion occurs any time a 4-bit address is specified for an instruction that takes 8-bit operands.
2. See the device product specification to determine the exact register file range available. The register file size varies by device type

## Z8<sup>PLUS</sup> INSTRUCTION SUMMARY

The instructions marked with this symbol (†) have an identical set of addressing modes, which are encoded for brevity. The upper nibble is described in Table 3-14, and the lower nibble is represented by [ ]. The second nibble's value is described in Table 3-15, and is found beside the applicable addressing mode pair. For example, the op code of an ADC instruction using the addressing modes *r* (destination) and *Ir* (source) is 13H.

Table 3-14. Instruction Summary

Instruction and Operation	Address Mode		op code Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
<b>ADC</b> dst, src dst ← dst + src +C	†		1[ ]	*	*	*	*	0	*
<b>ADD</b> dst, src dst ← dst + src	†		0[ ]	*	*	*	*	0	*
<b>AND</b> dst, src dst ← dst AND src	†		5[ ]	–	*	*	0	–	–
<b>CALL</b> src SP ← SP – 2 PC ← src		DA	D6	–	–	–	–	–	–
<b>CALL</b> src SP ← SP – 2 PC ← @src		IRR	D4	–	–	–	–	–	–
<b>CCF</b> C ← NOT C			EF	*	–	–	–	–	–
<b>CLR</b> dst dst ← 0	R IR		B0 B1	–	–	–	–	–	–
<b>COM</b> dst dst ← NOT dst	R IR		60 61	–	*	*	0	–	–
<b>CP</b> dst, src dst – src	†		A[ ]	*	*	*	*	–	–
<b>DA</b> dst dst ← DA dst	R IR		40 41	*	*	*	–	–	–
<b>DEC</b> dst dst ← dst – 1	R IR		00 01	–	*	*	*	–	–
<b>DECW</b> dst dst ← dst – 1	RR IR		80 81	–	*	*	*	–	–

Table 3-14. Instruction Summary (Continued)

Instruction and Operation	Address Mode		op code Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
<b>DI</b> IMR(7) ← 0			8F	-	-	-	-	-	-
<b>DJNZ</b> , dst, src r RA dst ← dst - 1 if dst ≠ 0 then PC ← PC + src Range: -128 ≤ src ≤ 127	RA		rA (r = 0 - F)	-	-	-	-	-	-
<b>EI</b> IMR(7) ← 1			9F	-	-	-	-	-	-
<b>HALT</b>			7F	-	-	-	-	-	-
<b>INC</b> dst dst ← dst + 1	r R IR		rE (r = 0 - F) 20 21	-	*	*	*	-	-
<b>INCW</b> dst dst ← dst + 1	RR IR		A0 A1	-	*	*	*	-	-
<b>IRET</b> FLAGS ← @SP; SP ← SP + 1 PC ← @SP; SP ← SP + 2; IMR(7) ← 1			BF	*	*	*	*	*	*
<b>JP</b> cc, src if cc is true, then PC ← src		DA	ccD (cc = 0 - F)	-	-	-	-	-	-
<b>JP</b> src PC ← @src		IRR	30	-	-	-	-	-	-
<b>JR</b> cc, src if cc is true, then PC ← PC + src Range: -128 ≤ src ≤ 127		RA	ccB c = 0 - F	-	-	-	-	-	-

Table 3-14. Instruction Summary (Continued)

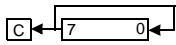
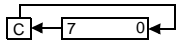
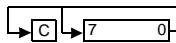
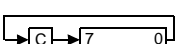
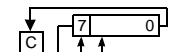
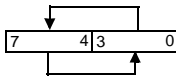
Instruction and Operation	Address Mode		op code Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
<b>LD</b> dst, src dst ← src	r r R	Im R r	r C r 8 r 9 (r = 0 – F)	–	–	–	–	–	–
	r X	X r	C7 D7						
	r Ir	Ir r	E3 F3						
	R R	R IR	E4 E5						
	R IR	IM IM	E6 E7						
	IR	R	F5						
<b>LDC</b> dst, src dst ← src	r lrr	Irr r	C2 D2	–	–	–	–	–	–
<b>LDCI</b> dst, src @dst ← @src dst ← dst + 1 src ← src + 1	Ir lrr	Irr r	C3 D3	–	–	–	–	–	–
<b>NOP</b>			FF	–	–	–	–	–	–
<b>OR</b> dst, src dst ← dst OR src	†		4[ ]	–	*	*	0	–	–
<b>POP</b> dst dst ← @SP SP ← SP + 1	R IR		50 51	–	–	–	–	–	–
<b>PUSH</b> src SP ← SP – 1 @SP ← src	R IR		70 71	–	–	–	–	–	–
<b>RCF</b> C ← 0			CF	0	–	–	–	–	–
<b>RET</b> PC ← @SP; SP ← SP + 2			AF	–	–	–	–	–	–
<b>RL</b> dst 	R IR		90 91	*	*	*	*	–	–

Table 3-14. Instruction Summary (Continued)

Instruction and Operation	Address Mode		op code Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
<b>RLC</b> dst 	R IR		10 11	*	*	*	*	-	-
<b>RR</b> dst 	R IR		E0 E1	*	*	*	*	-	-
<b>RRC</b> dst 	R IR		C0 C1	*	*	*	*	-	-
<b>SBC</b> dst, src dst ← dst - src - C	†		3[ ]	*	*	*	*	1	*
<b>SCF</b> C ← 1			DF	1	-	-	-	-	-
<b>SRA</b> dst 	R IR		D0 D1	*	*	*	0	-	-
<b>SRP</b> src RP ← src	Im		31	-	-	-	-	-	-
<b>STOP</b>			6F	-	-	-	-	-	-
<b>SUB</b> dst, src dst ← dst - src	†		2[ ]	*	*	*	*	1	*
<b>SWAP</b> dst 	R IR		F0 F1	-	*	*	-	-	-
<b>TCM</b> dst, src (NOT dst) AND src	†		6[ ]	-	*	*	0	-	-
<b>TM</b> dst, src dst AND src	†		7[ ]	-	*	*	0	-	-
<b>WDT</b>			5F	-	-	-	-	-	-
<b>XOR</b> dst, src dst ← dst XOR src	†		7[ ]	-	*	*	0	-	-

**Table 3-15. Lower Nibble Values**

Address Mode		Lower
dst	src	op code Nibble
r	r	[2]
r	Ir	[3]
R	R	[4]
R	IR	[5]
R	IM	[6]
IR	IM	[7]

Figure 3-2, which follows, illustrates the Op Code map.

# OP CODE MAP

		LOWER NIBBLE (HEX)																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
UPPER NIBBLE (HEX)	0	DEC R1	DEC IR1	ADD r1, r2	ADD r1, lr2	ADD R2, R1	ADD IR2, R1	ADD R1, IM	ADD IR1, IM	LD r1, R2	LD r2, R1	DJNZ r1, RA	JR cc, RA	LD r1, IM	JP cc, DA	INC r1		
	1	RLC R1	RLC IR1	ADC r1, r2	ADC r1, lr2	ADC R2, R1	ADC IR2, R1	ADC R1, IM	ADC IR1, IM									
	2	INC R1	INC IR1	SUB r1, r2	SUB r1, lr2	SUB R2, R1	SUB IR2, R1	SUB R1, IM	SUB IR1, IM									
	3	JP IRR1	SRP IM	SBC r1, r2	SBC r1, lr2	SBC R2, R1	SBC IR2, R1	SBC R1, IM	SBC IR1, IM									
	4	DA R1	DA IR1	OR r1, r2	OR r1, lr2	OR R2, R1	OR IR2, R1	OR R1, IM	OR IR1, IM									
	5	POP R1	POP IR1	AND r1, r2	AND r1, lr2	AND R2, R1	AND IR2, R1	AND R1, IM	AND IR1, IM									WDT
	6	COM R1	COM IR1	TCM r1, r2	TCM r1, lr2	TCM R2, R1	TCM IR2, R1	TCM R1, IM	TCM IR1, IM									STOP
	7	PUSH R2	PUSH IR2	TM r1, r2	TM r1, lr2	TM R2, R1	TM IR2, R1	TM R1, IM	TM IR1, IM									HALT
	8	DECW RR1	DECW IR1															DI
	9	RL R1	RL IR1															EI
	A	INCW RR1	INCW IR1	CP r1, r2	CP r1, lr2	CP R2, R1	CP IR2, R1	CP R1, IM	CP IR1, IM									RET
	B	CLR R1	CLR IR1	XOR r1, r2	XOR r1, lr2	XOR R2, R1	XOR IR2, R1	XOR R1, IM	XOR IR1, IM									IRET
	C	RRC R1	RRC IR1	LDC r1, lr2	LDCI lr1, lr2					LD r1,x,R2								RCF
	D	SRA R1	SRA IR1	LDC lr1, r2	LDCI lr1, lr2	CALL* IRR1		CALL DA	LD r2,x,R1									SCF
	E	RR R1	RR IR1		LD r1, IR2	LD R2, R1	LD IR2, R1	LD R1, IM	LD IR1, IM									CCF
	F	SWAP R1	SWAP IR1		LD lr1, r2		LD R2, IR1											NOP

2
3
2
3
1

**BYTES PER INSTRUCTION**

**Notes:**

All Z8<sup>PLUS</sup> instructions execute in ten XTAL clock cycles, (1 μS at 10 MHz).

Blank areas are reserved and execute as NOP.

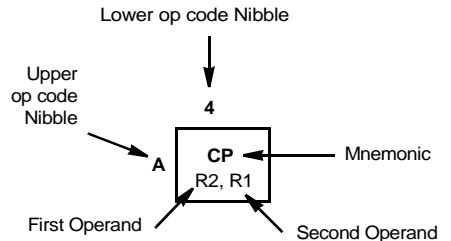
\* 2-byte instruction appears as a 3-byte instruction.

**Legend:**

R = 8-bit Addr  
r = 4-bit Addr  
R1 or r1 = Dst Addr  
R2 or r2 = Src Addr

**Sequence:**

op code,  
First Operand,  
Second Operand



**Figure 3-2. Op Code Map**

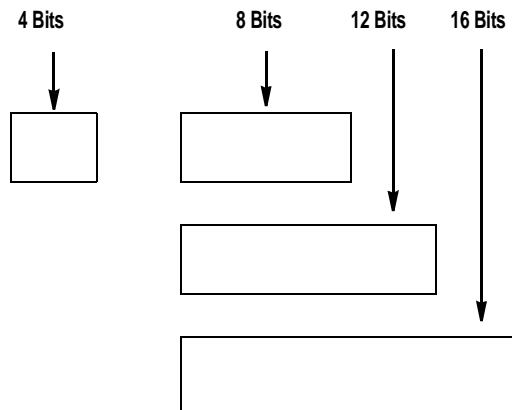


## INSTRUCTION DESCRIPTION AND FORMATS

The following section lists each instruction set, and describes the:

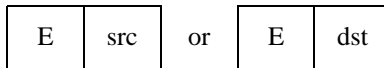
- Instruction Format
- Operation performed
- Flag Conditions
- Examples of the code

The format for the instruction uses the following conventions:

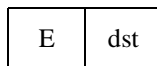


**NOTE:** The bytes shown in the boxes are in machine code order. The ZiLOG assembler always requires the format `OPC, dst, src`.

Address modes R or IR can be used to specify a 4-bit working register. In this format, the source or destination working-register operand is specified by adding 1110B (EH) to the High nibble of the operand. For example, if working register R12 (CH) is the destination operand, then ECH is used as the destination operand in the Op Code.



Address mode IRR can be used to specify a 4-bit working register Pair. In this format, the destination working register Pair operand is specified by adding 1110B (EH) to the High nibble of the operand. For example, if working register Pair RR12 (CH) is the destination operand, then ECH is used as the destination operand in the Op Code.



## ADC Add with Carry

### Instruction Format:

ADC dst, src

			OPC (Hex)	Address Mode dst	src
OPC	dst	src	12	r	r
			13	r	Ir
OPC	src	dst	14	R	R
			15	R	IR
OPC	dst	src	16	R	IM
			17	IR	IM

### Operation:

$dst \leftarrow dst + src + C$

The source operand, along with the setting of the Carry (C) Flag, is added to the destination operand. Two's complement addition is performed. The sum is stored in the destination operand. The contents of the source operand are not changed. In multiple precision arithmetic, this instruction permits the carry from the addition of low order operands to be carried into the addition of high order operands.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: 1 if a value is carried from the most significant bit of the result; otherwise, 0.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if the result is a negative value; otherwise, 0.
- V: 1 if an arithmetic overflow occurs (both operands have the same sign and the result has the opposite sign; otherwise, 0).
- D: 0.
- H: 1 if a value is carried from the most significant bit of the low-order four bits of the result; otherwise, 0.

**ADC**  
**Add with Carry**

**Example:** Working register R3 contains 16H. The C flag is set to 1. Working register R11 contains 20H. The following statement leaves the value 37H in working register R3, and the C, Z, S, V, D, and H flags are set to 0.

```
ADC R3, R11  
Op Code: 12 3B.
```

**Example:** Working register R16 contains 16H. The C flag is not set. Working register R10 contains 20H. Register 20H contains 11H. The following statements leave the value 27H in working register R16; the C, Z, S, V, D, and H flags are set to 0.

```
ADC R16, @R10  
Op Code: 13 FA
```

**Example:** Register 34H contains 2EH. The C flag is set. Register 12H contains 1BH. The following statement leaves the value 4AH in register 34H. The H flag is set, and the C, Z, S, V, and D flags are set to 0.

```
ADC 34H, 12H  
Op Code: 14 12 34
```

**Example:** Register 4BH contains 82H. The C flag is set. Working register R3 contains 10H. Register 10H contains 01H. The following statement leaves the value 84H in register 4BH. The S flag is set to 1, and the C, Z, V, D, and H flags are set to 0.

```
ADC 4BH, @R3  
Op Code: 15 E3 4B
```

## ADC Add with Carry

**Example:** Register 6CH contains 2AH. The C flag is not set. The following statement leaves the value 2DH in register 6CH. The C, Z, S, V, D, and H flags are set to 0.

```
ADC 6CH, #03H  
Op Code: 16 6C 03
```

**Example:** Register D4H contains 5FH. Register 5FH contains 4CH. The C flag is set. The following statement leaves the value 4FH in register 5FH. The C, Z, S, V, D, and H flags are set to 0.

```
ADC @D4H, #02H  
Op Code: 17 D4 02
```

**ADD  
Add**

**Instruction Format:**

ADD dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	02	r	r
			03	r	Ir
OPC	src		04	R	R
			05	R	IR
OPC	dst	src		06	R
			07	IR	IM

**Operation:**

dst ← dst + src

The source operand is added to the destination operand. Two's complement addition is performed. The sum is stored in the destination operand. The contents of the source operand are not changed.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: 1 if a value is carried from the most significant bit of the result; otherwise, 0.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if the result is negative; otherwise, 0.
- V: 1 if an arithmetic overflow occurs(both operands have the same sign and the result has the opposite sign); otherwise, 0.
- D: 0.
- H: 1 if a value is carried from the most significant bit of the result's low-order four bits; otherwise, 0.

## ADD Add

**Example:** Working register R3 contains 16H. Working register R11 contains 20H. The following statement leaves the value 36H in working register R3. The C, Z, S, V, D, and H flags are set to 0.

```
ADD R3, R11  
Op Code: 02 3B
```

**Example:** Working register R16 contains 16H. Working register R10 contains 20H. Register 20H contains 11H. The following statement leaves the value 27H in working register R16. The C, Z, S, V, D, and H flags are set to 0.

```
ADD R16, @R10  
Op Code: 03 FA
```

**Example:** Register 34H contains 2EH. Register 12H contains 1BH. The following statement leaves the value 49H in register 34H. The H flag is set to 1, and the C, Z, S, V, and D flags are set to 0.

```
ADD 34H, 12H  
Op Code: 04 12 34
```

**Example:** Register 4BH contains 82H. Working register R3 contains 10H. Register 10H contains 01H. The following statement leaves the value 83H in register 4BH. The S flag is set, and the C, Z, V, D, and H flags are set to 0.

```
ADD 3EH, @R3  
Op Code: 05 E3 4B
```

**Example:** Register 6CH contains 2AH. The following statement leaves the value 2DH in register 6CH. The C, Z, S, V, D, and H flags are set to 0.

```
ADD 6CH, #03H  
Op Code: 06 6C 03
```

**Example:** Register D4H contains 5FH. Register 5FH contains 4CH. The following statement leaves the value 4EH in register 5FH. The C, Z, S, V, D, and H flags are set to 0.

```
ADD @D4H, #02H  
Op Code: 07 D4 02
```

## AND Logical AND

### Instruction Format:

AND dst, src

			OPC (Hex)	Address Mode dst	src
OPC	dst	src	52	r	r
			53	r	Ir
OPC	src	dst	54	R	R
			55	R	IR
OPC	dst	src	56	R	IM
			57	IR	IM

### Operation:

dst ← dst AND src

The source operand and the destination operand are processed with a logical AND operation. The result is a 1 stored whenever the corresponding bits in the two operands are both 1; otherwise, a 0 is stored. The result is stored in the destination operand. The contents of the source register are unchanged.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 0
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Working register R1 contains 34H (00111000B) and working register R14 contains 4DH (10001101). The following statement leaves the value 04H (00001000) in working register R1. The Z, V, and S flags are set to 0.

```
AND R1, R14
Op Code: 52 1E
```

## AND Logical AND

**Example:** Working register R4 contains F9H (11111001B). Working register R13 contains 7BH. Register 7BH contains 6AH (01101010B). The following statement leaves the value 68H (01101000B) in working register R4. The Z, V, and S flags are set to 0.

```
AND R4, @R13  
Op Code: 53 4D
```

**Example:** Register 3AH contains the value F5H (11110101B). Register 42H contains the value 0AH (00001010). The following statement leaves the value 00H (00000000B) in register 3AH. The Z flag is set to 1, and the V and S flags are cleared.

```
AND 3AH, 42H  
Op Code: 54 42 3A
```

**Example:** If working register R5 contains F0H (11110000B). Register 45H contains 3AH. Register 3AH contains 7FH (01111111B). The following statement leaves the value 70H (01110000B) in working register R5. The Z, V, and S flags are set to 0.

```
AND R5, @45H  
Op Code: 55 45 E5
```

**Example:** Register 7AH contains the value F7H (11110111B). The following statement leaves the value F0H (11110000B) in register 7AH. The S flag is set to 1, and the Z and V flags are set to 0.

```
AND 7AH, #F0H  
Op Code: 56 7A F0
```

**Example:** Working register R3 contains the value 3EH. Register 3EH contains the value ECH (11101100B). The following statement leaves the value 04H (00000100B) in register 3EH. The Z, V, and S flags are set to 0.

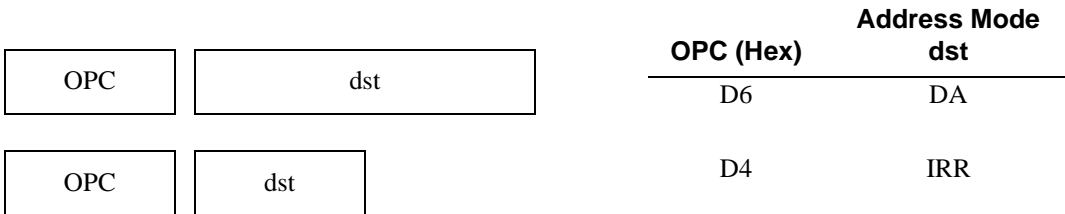
```
AND @R3, #05H  
Op Code: 57 E3 05
```



## CALL Call Procedure

**Instruction Format:**

CALL dst



**Operation:**

SP ← SP - 2  
 @SP ← PC  
 PC ← dst

The Stack pointer (SP) is decremented by 2. The current contents of the program counter (PC) (the address of the first instruction following the CALL instruction) are pushed onto the top of the Stack. The specified destination address is then loaded into the PC, which points to the first instruction of the procedure.

At the end of the procedure a return (RET) instruction can be used to return to the original program flow. RET pops the top of the Stack and replaces the original value into the PC.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

## CALL

### Call Procedure

**Example:** The contents of the PC are 1A47H and the contents of the SP (registers FEH and FFH) are 3002H. The following statements cause the SP to be decremented to 3000H, 1A4AH. The address following the CALL instruction is stored in external data memory at addresses 3000 and 3001H. The PC is loaded with 3521H and now points to the address of the first statement in the procedure to be executed.

```
CALL 3521H  
Op Code: D6 35 21
```

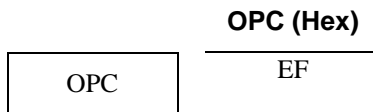
**Example:** The contents of the PC are 1A47H. The contents of the SP (register FFH) are 72H. The contents of register A4H are 34H. The contents of register pair 34H are 3521H. The following statements cause the SP to be decremented to 70H, 1A4AH. The address following the CALL instruction is stored in R70H and 71H. The PC is loaded with 3521H and now points to the address of the first statement in the procedure to be executed

```
CALL @A4H  
Op Code: D4 A4
```

## CCF Complement Carry Flag

### Instruction Format:

CCF



### Operation:

$$C \leftarrow \text{NOT } C$$

The C flag is complemented. If  $C = 1$ , then it is changed to  $C = 0$ ; or, if  $C = 0$ , then it is changed to  $C = 1$ .

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction is complemented.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The C flag contains a 0. The following statement changes the C flag from  $C = 0$  to  $C = 1$ .

CCF

Op Code: EF

## CLR Clear

### Instruction Format:

CLR dst

		OPC (Hex)	Address Mode dst
OPC	dst	B0	R
		B1	IR

### Operation:

dst ← 0

The destination operand is set to 00H.

### Flags

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Working register R6 contains AFH. The following statement leaves the value 00H in working register R6 .

```
CLR R6  
Op Code: B0 E6
```

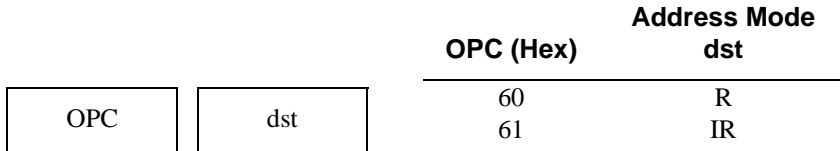
**Example:** Register A5H contains the value 23H. Register 23H contains the value FCH. The following statement leaves the value 00H in register 23H.

```
CLR @A5H  
Op Code: B1 A5
```

**COM  
Complement**

**Instruction Format:**

COM dst



**Operation:**

dst ← NOT dst

The contents of the destination operand are complemented (one's complement). All 1 bits are changed to 0, and all 0 bits are changed to 1.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if result bit 7 is set; otherwise, 0.
- V: 0
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Register 08H contains 24H (00100100B). The following statement leaves the value DBH (11011011) in register 08H. The S flag is set to 1, and the Z and V flags are set to 0.

```
COM 08
Op Code: 60 08
```

**Example:** Register 08H contains 24H, and register 24H contains FFH (11111111B). The following statement leaves the value 00H (00000000B) in register 24H. The Z flag is set to 1, and the V and S flags are set to 0.

```
COM @08H
Op Code: 61 08
```

## CP Compare

### Instruction Format:

CP dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	A2	r	r
			A3	r	Ir
OPC	src	dst	A4	R	R
			A5	R	IR
OPC	dst	src	A6	R	IM
			A7	IR	IM

### Operation:

dst - src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags are set accordingly. The contents of both operands are unchanged.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: 1 if a value is carried from the most significant bit of the result, otherwise, 0.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1 (negative); otherwise, 0.
- V: 1 if arithmetic overflow occurs; otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**CP  
Compare**

**Example:** Working register R3 contains 16H. Working register R11 contains 20H. The following statement sets the C and S flags to 1, and the Z and V flags are set to 0.

```
CP R3, R1  
Op Code: A2 3B
```

**Example:** Working register R15 contains 16H. Working register R10 contains 20H. Register 20H contains 11H. The following statement sets the C, Z, S, and V flags to 0.

```
CP R16, @R10  
Op Code: A3 FA
```

**Example:** Register 34H contains 2EH. Register 12H contains 1BH. The following statement sets the C, Z, S, and V flags to 0.

```
CP 34H, 12H  
Op Code: A4 12 34
```

**Example:** Register 4BH contains 82H. Working register R3 contains 10H. Register 10H contains 01H. The following statement sets the S flag to 1, and the C, Z, and V flags are set to 0.

```
CP 4BH, @R3  
Op Code: A5 E3 4B
```

**Example:** Register 6CH contains 2AH. The following statement sets the Z flag to 1, and the C, S, and V flags are set to 0.

```
CP 6CH, #2AH  
Op Code: A6 6C 2A
```

**Example:** Register D4H contains FCH. Register FCH contains 8FH. The following statement sets the V flag to 1, and the C, Z, and S flags are set to 0.

```
CP @D4H, 7FH  
Op Code: A7 D4 FF
```

## DA Decimal Adjust

### Instruction Format:

DA dst

		OPC (Hex)	Address Mode dst
OPC	dst	40	R
		41	IR

### Operation:

dst ← DA dst

The destination operand is adjusted to two 4-bit BCD digits following a binary addition or subtraction operation on BCD-encoded bytes. For addition (ADD and ADC) or subtraction (SUB and SBC), Table 3-14 indicates the operation performed.

**Table 3-16. DA Operation Reference**

Prior Instruction	Flags Before DA			Result Before		Adjustment Added	Result After		C Flag After
	C	H	D	[7...4]	[3...0]		[7...4]	[3...0]	
ADD or ADC	0	0	0	0-9	0-9	00	0-9	0-9	0
	0	0	0	0-8	A-F	06	1-9	0-5	0
	0	1	0	1-9	0-3	06	1-9	6-9	0
	0	0	0	A-F	0-9	60	0-5	0-9	1
	1	0	0	0-2	0-9	60	6-8	0-9	1
	0	0	0	9-F	A-F	66	0-5	0-5	1
	0	1	0	A-F	0-3	66	0-5	6-9	1
	1	0	0	0-2	A-F	66	6-9	0-5	1
	1	1	0	0-3	0-3	66	6-9	6-9	1
SUB or SBC	0	0	1	0-9	0-9	00	0-9	0-9	0
	0	1	1	0-8	6-F	FA	0-8	0-9	0
	1	0	1	7-F	0-9	A0	1-9	0-9	1
	1	1	1	6-F	6-F	9A	0-9	0-9	1

**Note:** If the destination operand is not the result of a valid addition or subtraction of BCD digits, the result is meaningless.



DA  
Decimal Adjust**Flags:**

When the instruction is executed, the flags are set as follows:

- C: 1 if a value is carried or borrowed during the prior addition or subtraction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1 (negative); otherwise, 0.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Addition is performed using the BCD values 15 and 27, the result should be 42. The sum actually obtained is incorrect, however, when the binary representations are added in the destination location using standard binary arithmetic.

$$\begin{array}{r}
 0001\ 0101 = 15\text{H} \\
 +\ 0010\ 0111 = 27\text{H} \\
 \hline
 0011\ 1100 = 3\text{CH}
 \end{array}$$

When the result of the addition is stored in Register 5FH, the following statement adjusts this result so the correct BCD representation is obtained.

```
DA 5FH
Op Code: 41 45
```

$$\begin{array}{r}
 0011\ 1100 = 3\text{CH} \\
 +\ 0000\ 0110 = 06\text{H} \\
 \hline
 0100\ 0010 = 42\text{H}
 \end{array}$$

Register 5F now contains the value 42H. The C, Z, and S flags are set to 0.

## DA Decimal Adjust

**Example:** A subtraction is performed on BCD values to subtract 17 from 25, the result should be 8. The result is incorrect when standard binary subtraction is performed on the binary representations of the BCD numbers.

$$\begin{array}{r} 0010\ 0101 = 25H \\ + \ 0001\ 0111 = 17H \\ \hline 0000\ 1110 = 0EH \end{array}$$

Register 45H contains the value 5FH. The result of the subtraction is stored in 5FH. The following statements adjust the result so the correct BCD representation is obtained.

```
DA @45H  
Op Code: 40 45
```

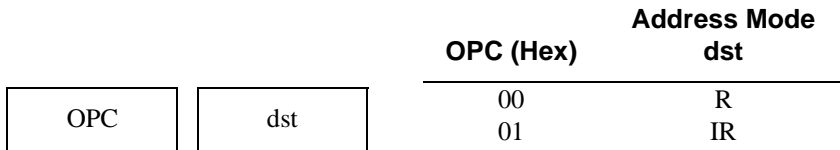
$$\begin{array}{r} 0000\ 1110 = 0EH \\ + \ 1111\ 1010 = FAH \\ \hline 0000\ 1000 = 08H \end{array}$$

Register 5FH now contains the value 08H. The C, Z, and S flags are set to 0.

**DEC**  
**Decrement**

**Instruction Format:**

DEC dst



**Operation:**

$dst \leftarrow dst - 1$

The contents of the destination operand are decremented by one.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1 (negative); otherwise, 0.
- V: 1 if arithmetic overflow occurs; otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Working register R10 contains 2AH. The following statement leaves the value 29H in working register R10. The Z, V, and S flags are set to 0.

```
DEC R10
Op Code: 00 EA
```

**Example:** Register B3H contains CBH. Register CBH contains 01H. The following statement leaves the value 00H in Register CBH. The Z flag is set to 1, and the V and S flags are set to 0.

```
DEC @B3H
Op Code: 01 B3
```

## DECW Decrement Word

### Instruction Format:

DECW dst

		OPC (Hex)	Address Mode dst
OPC	dst	80	RR
		81	IR

### Operation:

dst ← dst - 1

The contents of the destination (which must be an even address) operand are decremented by one. The destination operand can be a Register Pair or a working register Pair.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0
- S: 1 if bit 7 of the result is 1 (negative); otherwise, 0
- V: 1 if arithmetic overflow occurs; otherwise, 0
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Register pair 30H and 31H contain the value 0AF2H. The statement leaves the value 0AF1H in register pair 30H and 31H. The Z, V, and S flags are set to 0.

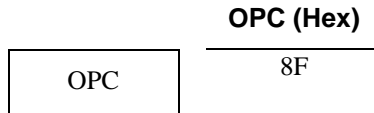
```
DECW 30H
Op Code: 80 30
```

**Example:** Working register R0 contains 30H. Register Pair 30H and 31H contain the value FAF3H. The following statement leaves the value FAF2H in Register Pair 30H and 31H. The S flag is set, and the Z and V flags are cleared.

```
DECW @R0
Op Code: 81 E0
```

DI  
Disable Interrupts**Instruction Format:**

DI

**Operation:**

IMASK (7) ← 0

Bit 7 of control register FBH (the Interrupt Mask Register) is reset to 0. All interrupts are disabled, although they remain potentially enabled. For example, the Global Interrupt Enable is cleared, but not the individual interrupt level enables.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Control register FBH contains 8AH (10001010B) (interrupts IRQ1 and IRQ3 are enabled). The following statement sets control register FBH to 0AH (00001010B) and disables all interrupts.

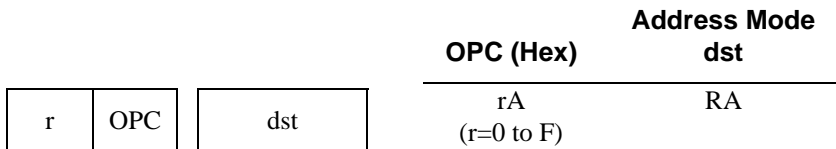
DI

Op Code: 8F

## DJNZ Decrement And Jump If Non-zero

### Instruction Format:

DJNZ r, dst



### Operation:

$r \leftarrow r - 1;$   
If  $r \neq 0$ ,  $PC \leftarrow PC + dst$

The specified working register serves as a counter and is decremented. If the contents of the specified working register are not 0 after decrementing, then the relative address is added to the Program Counter (PC) and control passes to the statement whose address is now in the PC. The range of the relative address is +127 to -128. The original value of the PC is the address of the instruction byte following the DJNZ statement. When the specified working register counter reaches 0, control falls through to the statement following the DJNZ instruction.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** DJNZ is typically used to control a loop of instructions. In this example, 12 bytes are moved from one buffer area in the register file to another. The steps involved are:

1. Load 12 into the counter (working register R6).
2. Set up the loop to perform the moves.
3. End the loop with a DJNZ instruction.

**DJNZ**  
**Decrement And Jump If Non-zero**

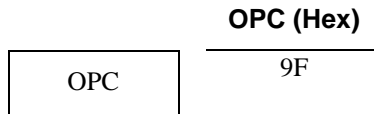
The assembly listing required for this routine is as follows:

	<b>Assembly</b>	<b>Op Code</b>
	LD R6, #12	6E 0C
LOOP:	LD R9 %20(R6)	C7 56 30
	LD %14(R6), R9	D7 56 10
	DJNZ R6, LOOP	6A F8

## EI Enable Interrupts

### Instruction Format:

EI



### Operation:

IMASK (7) ← 1

Bit 7 of Control Register FBH (the Interrupt Mask Register) is set to 1. This allows potentially enabled interrupts to become enabled.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Control Register FBH contains 0AH (00001010) (interrupts IRQ1 and IRQ3 are selected). The following statement sets Control Register FBH to 8AH (10001010B) enabling IRQ1 and IRQ3.

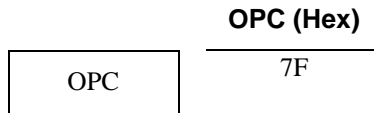
EI

Op Code: 9F



**HALT**  
**Halt****Instruction Format:**

HALT

**Operation:**

The HALT instruction turns off the internal CPU clock, but not the XTAL oscillation. The peripherals and interrupt logic remain active. Operation can be restarted by an interrupt or a reset.

**Flags:**

When the instruction is executed, the flags are set as follows

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Assuming the Z8 is in normal operation, the following statements place the Z8 into HALT mode.

HALT

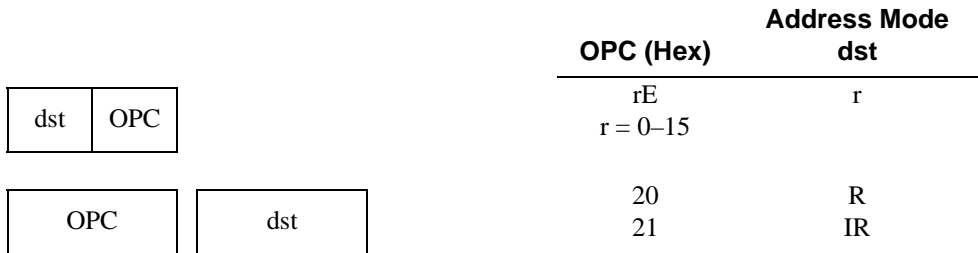
Op Codes: 7F

**NOTE:** Unlike the Z8, the Z8<sup>PLUS</sup> does not require a NOP before the HALT instruction.

## INC Increment

### Instruction Format:

INC dst



### Operation:

dst ← dst + 1

The contents of the destination operand are incremented by one.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1 (negative); otherwise, 0.
- V: 1 if arithmetic overflow occurs; otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**INC**  
**Increment**

**Example:** Working register R10 contains 2AH. The following statement leaves the value 2BH in working register R10. The Z, V, and S flags are set to 0.

```
INC R10  
Op Code: AE
```

**Example:** Register B3H contains CBH. The following statement leaves the value CCH in register CBH. The S flag is set to 1, and the Z and V flags are set to 0.

```
INC B3H  
Op Code: 20 B3
```

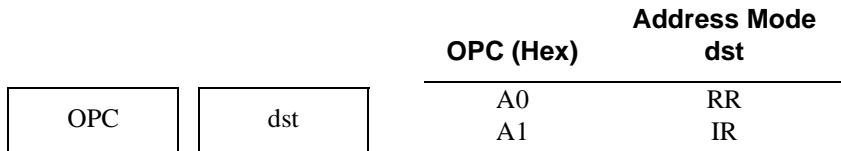
**Example:** Register B3H contains CBH. Register CBH contains FFH. The following statement leaves the value 00H in register CBH. The Z flag is set to 1, and the V and S flags are set to 0.

```
INC @B3H  
Op Code: 21 B3
```

## INCW Increment Word

### Instruction Format:

INCW dst



### Operation:

$dst \leftarrow dst + 1$

The contents of the destination (which must be an even address) operand is incremented by one. The destination operand can be a Register Pair or a working register Pair.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1 (negative); otherwise, 0.
- V: 1 if arithmetic overflow occurs; otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Register pairs 30H and 31H contain the value 0AF2H. The following statement leaves the value 0AF3H in register pair 30H and 31H. The Z, V, and S flags are set to 0.

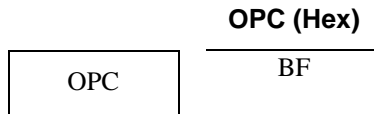
```
INCW 30H
Op Code: A0 30
```

**Example:** Working register R0 contains 30H. Register pairs 30H and 31H contain the value FAF3H. The following statement leaves the value FAF4H in register pair 30H and 31H. The S flag is set, and the Z and V flags are set to 0.

```
INCW @R0
Op Code: A1 E0
```

IRET  
Interrupt Return**Instruction Format:**

IRET

**Operation:**

```

FLAGS ← @SP
SP ← SP + 1
PC ← @SP
SP ← SP + 2
IMR (7) ← 1

```

This instruction is issued at the end of an interrupt service routine. It restores the Flag Register (Control Register FCH) and the PC. It also re-enables any interrupts that are potentially enabled.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value prior to the issuance of the interrupt.
- Z: The value prior to the issuance of the interrupt.
- S: The value prior to the issuance of the interrupt.
- V: The value prior to the issuance of the interrupt.
- D: The value prior to the issuance of the interrupt.
- H: The value prior to the issuance of the interrupt.

**Example:** Stack Pointer Low (register FFH) currently contains the value 45H. Register 45H contains the value 00H. Register 46H contains 6FH. Register 47 Contains E4H. The following statement restores the Flags Register (FCH) with the value 00H, restores the PC with the value 6FE4H, re-enables the interrupts, and sets the Stack Pointer Low to 48H. The next instruction to be executed is at location 6FE4H.

```

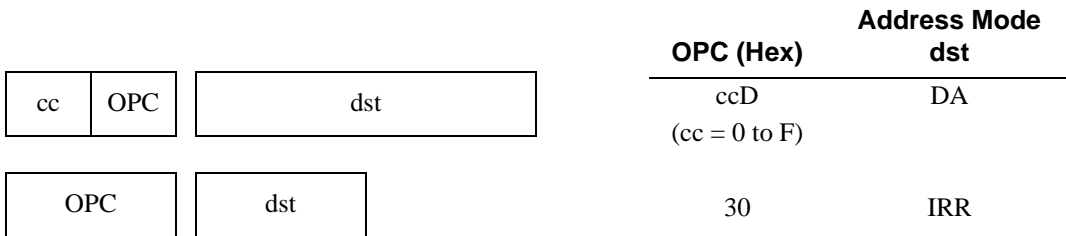
IRET
Op Code: BF

```

## JP Jump

### Instruction Format:

JP cc, dst



### Operation:

If condition code is true, then PC ← dst

A conditional jump (JP) transfers program control to the destination address if the condition specified by cc is true. Otherwise, the instruction following the JP instruction is executed. See page 3-8 for a list of condition codes.

**NOTE:** Op Code 30H (JP IRR) is *unconditional* only.

An unconditional jump simply replaces the contents of the Program Counter with the contents of the register pair specified by the destination operand. Program Control then passes to the instruction addressed by the PC.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**JP**  
**Jump**

**Example:** The Carry flag is 1. The following statement replaces the contents of the Program Counter with 1520H and transfers program control to that location. If the Carry flag had not been 1, control would have fallen through to the statement following the JP instruction.

```
JP C, 1520H  
Op Code: 7D 15 20
```

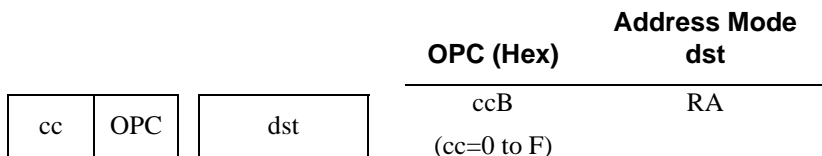
**Example:** Working register pair RR2 contains the value 3F45H. The following statement replaces the contents of the PC with the value 3F45H and transfers program control to that location.

```
JP @RR2  
Op Code: 30 E2
```

## JR Jump Relative

### Instruction Format:

JR cc, dst



### Operation:

If cc is true,  $PC \leftarrow PC + dst$

If the condition specified by the cc is true, the relative address is added to the PC and control passes to the instruction located at the address specified by the PC (See page 3-8 for a list of condition codes). Otherwise, the instruction following the JR instruction is executed. The range of the relative address is +127 to -128, and the original value of the PC is taken to be the address of the first instruction byte following the JR instruction.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The result of the last arithmetic operation executed is negative. The next nine bytes are skipped with the following statement. If the result is not negative, execution continues with the instruction following the JR instruction.

```
JR MI, 9
Op Code: 5B 09
```

**Example:** A short form of a jump -45 is:

```
JR -45
Op Code: 8B D3
```

The instruction jumps backwards 45 bytes, unconditionally. The condition code is blank in this case, and is assumed to be always true.



**LD  
Load**

**Instruction Format:**

LD dst, src

				OPC (Hex)	Address Mode	
					dst	src
dst	OPC	src		rC	r	IM
				r8	r	R
src	OPC	dst		r9	R*	r
				r=0 to F		
OPC		dst	src	E3	r	Ir
				F3	Ir	r
OPC	src		dst	E4	R	R
				E5	R	IR
OPC	dst		src	E6	R	IM
				E7	IR	IM
OPC	src		dst	F5	IR	R
OPC	dst	X	src	C7	r	X
OPC	src	X	dst	D7	X	r

\*For OPC r9H, only a full 8-bit register can be used. The ZiLOG assembler automatically uses the r8 Op Code for an instruction like:

LD R0,R1.

## LD Load

### Operation:

$dst \leftarrow src$

The contents of the source operand are loaded into the destination operand. The contents of the source operand are not changed.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The following statement loads the value 34H into working register R15.

```
LD R15, #34H  
Op Code: FC 34
```

**Example:** Register 34H contains the value FCH. The following statement loads the value FCH into working register R14. The contents of register 34H are not changed.

```
LD R14, 34H  
Op Code: F8 34
```

**Example:** Working register R14 contains the value 45H. The following statement loads the value 45H into register 34H. The contents of working register R14 are not changed.

```
LD 34H, R14  
Op Code: E9 34
```

**Example:** Working register R12 contains the value 34H. Register 34H contains the value FFH. The following statement loads the value FFH into working register R13. The contents of working register R12 and register 34H are not changed.

```
LD R13, @R12  
Op Code: E3 DC
```

**LD  
Load**

**Example:** Working register R13 contains the value 45H. Working register R12 contains the value 00H. The following statement loads the value 00H into register 45H. The contents of working register R12 and working register R13 are not changed.

```
LD @R13, R12  
Op Code: F3 DC
```

**Example:** Register 45H contains the value CFH. The following statement loads the value CFH into register 34H. The contents of register 45H are not changed.

```
LD 34H, 45H  
Op Code: E4 45 34
```

**Example:** Register 45H contains the value CFH. Register CFH contains the value FFH. The following statement loads the value FFH into register 34H. The contents of register 45H and register CFH are not changed.

```
LD 34H, @45H  
Op Code: E5 45 34
```

**Example:** The following statement loads the value A4H into Register 34H.

```
LD 34H, #0A4H  
Op Code: E6 34 A4
```

**Example:** Working register R14 contains the value 7FH. The following statement loads the value FCH into Register 7FH. The contents of working register R14 are not changed.

```
LD @R14, #0FCH  
Op Code: E7 EE FC
```

## LD Load

**Example:** Register 34H contains the value CFH. Register 45H contains the value FFH. The following statement loads the value FFH into register CFH. The contents of register 34H and register 45H are not changed.

```
LD @34H, 45H  
Op Code: F5 45 34
```

**Example:** Working register R0 contains the value 08H. Register 2CH (24H + 08H = 2CH) contains the value 4FH. The following statement loads working register R10 with the value 4FH. The contents of working register R0 and Register 2CH are not changed.

```
LD R10, 24H(R0)  
Op Code: C7 A0 24
```

**Example:** Working register R0 contains the value 0BH. Working register R10 contains 03H. The following statement loads the value 03H into register FBH (F0H + 0BH = FBH). Since this is the Interrupt Mask Register, the LOAD statement has the effect of enabling IRQ0 and IRQ1. The contents of working registers R0 and R10 are unchanged by the load.

```
LD F0H(R0), R10  
Op Code: D7 A0 F0
```

LDC  
Load Constant**Instruction Format:**

LDC dst, src

			OPC (Hex)	Address Mode dst	src
OPC	dst	src	C2	r	Irr
OPC	dst	src	D2	Irr	r

**Operation:**

dst ← src

This instruction is used to load a byte constant from program memory into a working register, or vice versa. The address of the program memory location is specified by a working register pair. The contents of the source operand are not changed.

**Flags**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Working register pairs R6 and R7 contain the value 30A2H and program memory location 30A2H contains the value 22H. The following statement loads the value 22H into working register R2. The value of program memory location 30A2H is unchanged by the load.

```
LDC R2, @RR6
Op Code: C2 26
```

## LDC Load Constant

**Example:** Working register R2 contains the value 22H. Working register pair R6 and R7 contains the value 10A2H. The following statement loads the value 22H into program memory location 10A2H. The value of working register R2 is unchanged by the load.

```
LDC @RR6, R2  
Op Code: D2 26
```

**NOTE:** This instruction format is valid only for MCUs which can write to program memory.

## LDCI Load Constant Auto Increment

**Instruction Format:**

```
LDCI dst, src
```

	OPC (Hex)	Address Mode				
		dst	src			
<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr><td style="padding: 5px;">OPC</td></tr> </table> <table border="1" style="display: inline-table; border-collapse: collapse; margin-left: 10px;"> <tr> <td style="padding: 5px; width: 40px;">dst</td> <td style="padding: 5px; width: 40px;">src</td> </tr> </table>	OPC	dst	src	C3	Ir	Irr
OPC						
dst	src					
<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr><td style="padding: 5px;">OPC</td></tr> </table> <table border="1" style="display: inline-table; border-collapse: collapse; margin-left: 10px;"> <tr> <td style="padding: 5px; width: 40px;">dst</td> <td style="padding: 5px; width: 40px;">src</td> </tr> </table>	OPC	dst	src	D3	Irr	Ir
OPC						
dst	src					

**Operation:**

```
dst ← src
r ← r + 1
rr ← rr + 1
```

This instruction is used for block transfers of data between program memory and the Register File. The address of the program memory location is specified by a working register Pair, and the address of the Register File location is specified by working register. The contents of the source location are loaded into the destination location. Both addresses in the working registers are then incremented automatically. The contents of the source operand are not changed.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

## LDCI Load Constant Auto-increment

**Example:** Working register pair R6-R7 contains 30A2H, program memory location 30A2H and 30A3H contain 22H and BCH respectively, and working register R2 contains 20H. The following statement loads the value 22H into Register 20H. working register Pair RR6 is incremented to 30A3H and working register R2 is incremented to 21H.

```
LDCI @R2, @RR6  
Op Code: C3 26
```

A second statement loads the value BCH into register 21H. working register pair RR6 is incremented to 30A4H and working register R2 is incremented to 22H.

```
LDCI @R2, @RR6  
Op Code: C3 26
```

**Example:** Working register R2 contains 20H. Register 20H contains 22H. Register 21H contains BCH. Working register pair R6-R7 contains 30A2H. The following statement loads the value 22H into program memory location 30A2H. working register R2 is incremented to 21H and working register Pair R6-R7 is incremented to 30A3H.

```
LDCI @RR6, @R2  
Op Code: D3 26
```

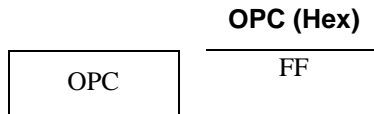
A second statement loads the value BCH into program memory location 30A3H. working register R2 is incremented to 22H and working register pair R6-R7 is incremented to 30A4H.

```
LDCI @RR6, @R2  
Op Code: D3 26
```



**NOP**  
**No Operation****Instruction Format:**

NOP

**Operation:**

No action is performed by this instruction. It is typically used for timing delays or clearing the pipeline.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

## OR Logical OR

### Instruction Format:

OR dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	42	r	r
			43	r	Ir
OPC	src	dst	44	R	R
			45	R	IR
OPC	dst	src	46	R	IM
			47	IR	IM

### Operation:

dst ← dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination operand. The contents of the source operand are not changed. The OR operation stores a 1 bit whenever either of the corresponding bits in the two operands is a 1. Otherwise, a 0 bit is stored.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**OR**  
**Logical OR**

**Example:** Working register R1 contains 34H (00111000B). Working register R14 contains 4DH (10001101). The following statement leaves the value BDH (10111101B) in working register R1. The S flag is set to 1, and the Z and V flags are set to 0.

```
OR R1, R14  
Op Code: 42 1E
```

**Example:** Working register R4 contains F9H (11111001B). Working register R13 contains 7BH. Register 7B contains 6AH (01101010B). The following statement leaves the value FBH (11111011B) in working register R4. The S flag is set to 1, and the Z and V flags are set to 0.

```
OR R4, @R13  
Op Code: 43 4D
```

**Example:** Register 3AH contains the value F5H (11110101B). Register 42H contains the value 0AH (00001010B). The following statement leaves the value FFH (11111111B) in register 3AH. The S flag is set to 1, and the Z and V flags are set to 0.

```
OR 3AH, 42H  
Op Code: 44 42 3A
```

**Example:** Working register R5 contains 70H (01110000B). Register 45H contains 3AH. Register 3AH contains 7FH (01111111B). The following statement leaves the value 7FH (01111111B) in working register R5. The Z, V, and S flags are set to 0.

```
OR R5, @45H  
Op Code: 45 45 E5
```

**Example:** Register 7AH contains the value F3H (11110111B). The following statement leaves the value F3H (11110111B) in register 7AH. The S flag is set to 1, and the Z and V flags are set to 0.

```
OR 7AH, #F0H  
Op Code: 46 7A F0
```

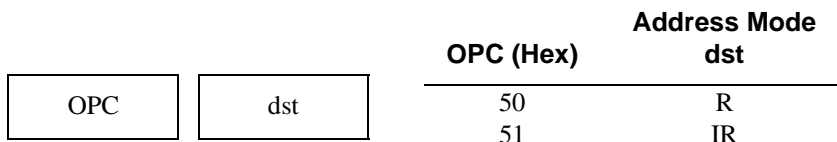
**Example:** Working register R3 contains the value 3EH. Register 3EH contains the value 0CH (00001100B). The following statement leaves the value 0DH (00001101B) in register 3EH. The Z, V, and S flags are set to 0.

```
OR @R3, #05H  
Op Code: 57 E3 05
```

## POP Pop

### Instruction Format:

POP dst



### Operation:

dst ← @SP  
SP ← SP + 1

The contents of the location specified by the Stack Pointer (SP) are loaded into the destination operand. The SP is then incremented automatically.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The SP (Control Registers FEH and FFH) contains the value 70H. Register 70H contains 44H. The following statement loads the value 44H into register 34H. After the POP operation, the SP contains 71H. The contents of register 70 are not changed.

```
POP 34H
Op Code: 50 34
```

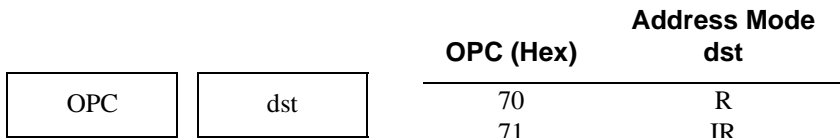
**Example:** The SP (Control Registers FEH and FFH) contains the value 1000H. Memory location 1000H contains 55H. Working register R6 contains 22H. The following statement loads the value 55H into register 22H. After the POP operation, the SP contains 1001H. The contents of working register R6 are not changed.

```
POP @R6
Op Code: 51 E6
```

**PUSH**  
Push

**Instruction Format:**

PUSH src



**Operation:**

SP ← SP - 1  
@SP ← src

The contents of the SP (stack pointer) are decremented by one. Then, the contents of the source operand are loaded into the location addressed by the updated SP, adding a new element to the stack.

**Flags**

:When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The SP contains 1001H. The following statement stores the contents of Register FCH (the Flag Register) in location 1000H. After the PUSH operation, the SP contains 1000H.

```
PUSH FCH
Op Code: 70 FC
```

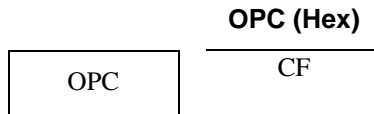
**Example:** The SP contains 61H. Working register R4 contains FCH. The following statement stores the contents of register FCH (the Flag Register) in location 60H. After the PUSH operation, the SP contains 60H.

```
PUSH @R4
Op Code: 71 E4
```

## RCF Reset Carry Flag

### Instruction Format:

RCF



### Operation:

$C \leftarrow 0$

The C flag is reset to 0, regardless of its previous value.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: 0
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

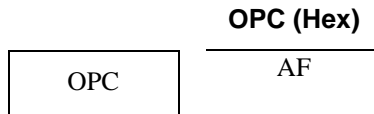
**Example:** The C flag is currently set to 1. The following statement resets the Carry flag to 0.

RCF

Op Code: CF

**RET**  
**Return****Instruction Format:**

RET

**Operation:**

PC ← @SP  
 SP ← SP + 2

This instruction is used to return from a procedure entered by a CALL instruction. The contents of the location addressed by the stack pointer (SP) are popped into the Program Control. The next statement executed is the one addressed by the new contents of the PC. The stack pointer is also incremented by 2.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**NOTE:** Each PUSH instruction executed within the subroutine should be countered with a POP instruction in order to guarantee the SP is at the correct location when the RET instruction is executed. Otherwise the wrong address is loaded into the PC and the program does not operate as desired.

**Example:** SP contains 200H. Memory location 200H contains 18H. Location 201H contains B5H. The following statement leaves the value 202H in the SP, and the PC contains 18B5H, the address of the next instruction to be executed.

RET  
 Op Code: AF

## RL Rotate Left

### Instruction Format:

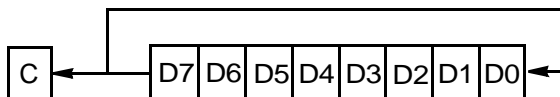
RL dst

		OPC (Hex)	Address Mode dst
OPC	dst	90	R
		91	IR

### Operation:

$C \leftarrow \text{dst}(7)$   
 $\text{dst}(0) \leftarrow \text{dst}(7)$   
 $\text{dst}(1) \leftarrow \text{dst}(0)$   
 $\text{dst}(2) \leftarrow \text{dst}(1)$   
 $\text{dst}(3) \leftarrow \text{dst}(2)$   
 $\text{dst}(4) \leftarrow \text{dst}(3)$   
 $\text{dst}(5) \leftarrow \text{dst}(4)$   
 $\text{dst}(6) \leftarrow \text{dst}(5)$   
 $\text{dst}(7) \leftarrow \text{dst}(6)$

The contents of the destination operand are rotated left by one bit position. The value from bit 7 is moved to the bit 0 position and also into the Carry flag.



### Flags:

When the instruction is executed, the flags are set as follows:

- C: 1 if the bit rotated from the most significant bit position was 1 (that is, bit 7 was previously set to 1).
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 1 if arithmetic overflow occurred (if the sign of the destination operand changed during rotation); otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.



**RL**  
**Rotate Left**

**Example:** The contents of register C6H are 88H (10001000B). The following statement leaves the value 11H (00010001B) in register C6H. The C and V flags are set to 1, and the S and Z flags are set to 0.

```
RL C6H  
Op Code: 80 C6
```

**Example:** The contents of register C6H are 88H. The contents of register 88H are 44H (01000100B). The following statement leaves the value 88H in register 88H (10001000B). The S and V flags are set to 1, and the C and Z flags are set to 0.

```
RL @C6H  
Op Code: 81 C6
```

## RLC Rotate Left Through Carry

### Instruction Format:

RLC dst

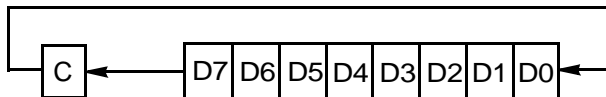
		OPC (Hex)	Address Mode dst
OPC	dst	10	R
		11	IR

### Operation:

```

C ← dst(7)
dst(0) ← C
dst(1) ← dst(0)
dst(2) ← dst(1)
dst(3) ← dst(2)
dst(4) ← dst(3)
dst(5) ← dst(4)
dst(6) ← dst(5)
dst(7) ← dst(6)
    
```

The contents of the destination operand along with the C flag are rotated left by one bit position. The initial value of bit 7 becomes the value of the C flag and the previous value of the C flag becomes the value of bit 0.



## RLC Rotate Left Through Carry

### Flags:

When the instruction is executed, the flags are set as follows:

- C: 1 if the bit rotated from the most significant bit position was 1 (that is, bit 7 was previously set to 1).
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 1 if arithmetic overflow occurred (if the sign of the destination operand changed during rotation); otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The C flag is reset. Register C6 contains 8F (10001111B). The following statement leaves register C6 with the value 1EH (00011110B). The C and V flags are set to 1, and S and Z flags are set to 0.

```
RLC C6  
Op Code: 10 C6
```

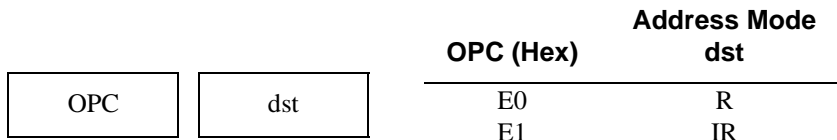
**Example:** The C flag is reset. Working register R4 contains C6H. Register C6 contains 8F (10001111B). The following statement leaves register C6 with the value 1EH (00011110B). The C and V flags are set to 1, and S and Z flags are set to 0.

```
RLC @R4  
Op Code: 11 E4
```

## RR Rotate Right

### Instruction Format:

RR dst

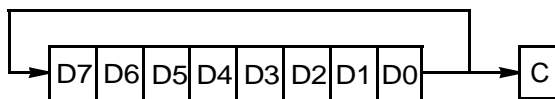


### Operation:

```

C ← dst(0)
dst(0) ← dst(1)
dst(1) ← dst(2)
dst(2) ← dst(3)
dst(3) ← dst(4)
dst(4) ← dst(5)
dst(5) ← dst(6)
dst(6) ← dst(7)
dst(7) ← dst(0)
    
```

The contents of the destination operand are rotated to the right by one bit position. The initial value of bit 0 becomes the value of bit 7 and the C flag.



### Flags:

When the instruction is executed, the flags are set as follows:

- C: 1 if the value rotated from the least significant bit position (bit 0) was 1.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 1 if arithmetic overflow occurred (if the sign of the destination operand changed during rotation); otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**RR**  
**Rotate Right**

**Example:** The contents of working register R6 are 31H (00110001B). The following statement leaves the value 98H (10011000B) in working register R6. The C, V, and S flags are set to 1, and the Z flag is set to 0.

```
RR R6  
Op Code: E0 E6
```

**Example:** The contents of register C6 are 31H. The contents of register 31H are 7EH (01111110B). The following statement leaves the value 4FH (00111111B) in register 31H. The C, Z, V, and S flags are set to 0.

```
RR @C6  
Op Code: E1 C6
```

## RRC Rotate Right Through Carry

### Instruction Format:

RRC dst

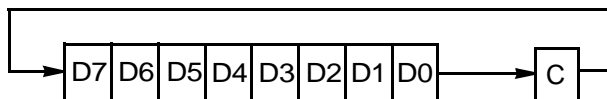
### Operation:

OPC	dst	OPC (Hex)	Address Mode dst
		C0	R
		C1	IR

```

C ← dst(0)
dst(0) ← dst(1)
dst(1) ← dst(2)
dst(2) ← dst(3)
dst(3) ← dst(4)
dst(4) ← dst(5)
dst(5) ← dst(6)
dst(6) ← dst(7)
dst(7) ← C
    
```

The contents of the destination operand with the C flag are rotated right by one bit position. The value of the C flag becomes the value of bit 7; the value of bit 0 becomes the value of the C flag.



## RRC Rotate Right Through Carry

### Flags:

When the instruction is executed, the flags are set as follows:

- C: 1 if the bit rotated from the least significant bit position was 1 (that is, bit 0 was 1).
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 1 if an arithmetic overflow occurs (the sign of the destination operand changed during rotation); otherwise, 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The contents of register C6H are DDH (11011101B). The C flag is 0. The following statement leaves the value 6EH (01101110B) in register C6H. The C and V flags are set to 1, and the Z and S flags are set to 0.

```
RRC C6H  
Op Code: C0 C6
```

**Example:** The contents of register 2C are EDH. The contents of register EDH is 02H (00000010B). The C flag is 0. The following statement leaves the value 01H (00000001B) in register EDH. The C, Z, S, and V flags are reset to 0.

```
RRC @2CH  
Op Code: C1 2C
```

## SBC Subtract with Carry

### Instruction Format:

SBC dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	32	r	r
			33	r	Ir
OPC	src	dst	34	R	R
			35	R	IR
OPC	dst	src	36	R	IM
			37	IR	IM

### Operation:

$$dst \leftarrow dst - src - C$$

The value of the source operand, and the value of the C flag, are subtracted from the destination operand. The result is stored in the destination operand. The contents of the source operand do not change. Subtraction is performed by adding the two's complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry (borrow) from the subtraction of low-order operands to be subtracted from the subtraction of high-order operands.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: 0 if a value is carried from the most significant bit of the result; otherwise, 1 (indicating a borrow).
- Z: 1 if the result is 0; otherwise, 0.
- V: 1 if an arithmetic overflow occurs (the operands have opposite signs, and the sign of the result is the same as the sign of the source); otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- H: 0 if a value is carried from the most significant bit of the low-order four bits of the result; otherwise, 1 (indicating a borrow).
- D: 1.



**SBC**  
**Subtract with Carry**

**Example:** Working register R3 contains 16H. The C flag is set to 1. Working register R11 contains 20H. The following statement leaves the value F5H in working register R3. The C, S, and D flags are set to 1, and the Z, V, and H flags are set to 0.

```
SBC R3, R11  
Op Code: 32 3B
```

**Example:** Working register R15 contains 16H. The C flag is not set. Working register R10 contains 20H. Register 20H contains 11H. The following statement leaves the value 05H in working register R15. The D flag is set to 1, and the C, Z, S, V, and H flags are set to 0.

```
SBC R16, @R10  
Op Code: 33 FA
```

**Example:** Register 34H contains 2EH. The C flag is set. Register 12H contains 1BH. The following statement leaves the value 12H in register 34H. The D flag is set, and the C, Z, S, V, and H flags are cleared.

```
SBC 34H, 12H  
Op Code: 34 12 34
```

**Example:** Register 4BH contains 82H. The C flag is set. Working register R3 contains 10H. Register 10H contains 01H. The following statement leaves the value 80H in register 4BH. The D and S flags are set to 1, and the C, Z, V, and H flags are set to 0.

```
SBC 4BH, @R3  
Op Code: 35 E3 4B
```

**Example:** Register 6CH contains 2AH. The C flag is not set. The following statement leaves the value 27H in register 6CH. The D flag is set to 1, and the C, Z, S, V, and H flags are set to 0.

```
SBC 6CH, #03H  
Op Code: 36 6C 03
```

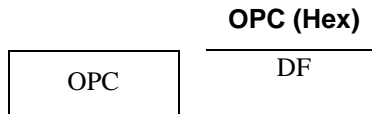
**Example:** Register D4H contains 5FH. Register 5FH contains 4CH. The C flag is set. The following statement leaves the value 49H in register 5FH. The D flag is set to 1, and the C, Z, S, V, and H flags are set to 0.

```
SBC @D4H, #02H  
Op Code: 37 D4 02
```

## SCF Set Carry Flag

### Instruction Format:

SRC



### Operation:

$C \leftarrow 1$

The C flag is set to 1, regardless of its previous value.

### Flags:

When the instruction is executed, the flags are set as follows:

- C     1.
- Z     The value set by the preceding instruction.
- S     The value set by the preceding instruction.
- V     The value set by the preceding instruction.
- D     The value set by the preceding instruction.
- H     The value set by the preceding instruction.

**Example:** The C flag is currently 0. The following statement sets the Carry flag to 1.

SCF

Op Code: DF

## SRA Shift Right Arithmetic

### Instruction Format:

SRA dst

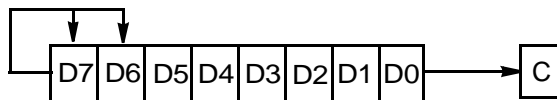
		OPC (Hex)	Address Mode dst
OPC	dst	D0	R
		D1	IR

### Operation:

```

C ← dst(0)
dst(0) ← dst(1)
dst(1) ← dst(2)
dst(2) ← dst(3)
dst(3) ← dst(4)
dst(4) ← dst(5)
dst(5) ← dst(6)
dst(6) ← dst(7)
dst(7) ← dst(7)
    
```

An arithmetic right shift by one bit position is performed on the destination operand. Bit 0 replaces the C flag. The value of Bit 7 (the sign bit) is unchanged. Bit 6 becomes the same as the value of bit 7. The result is a signed divide by two holding the half-bit remainder stored in the Carry (C) flag.



### Flags:

When the instruction is executed, the flags are set as follows:

- C: 1 if the value rotated from the least-significant bit (bit 0) position was 1.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

## SRA Shift Right Arithmetic

**Example:** The contents of working register R6 are 31H (00110001B). The following statement leaves the value 98H (00011000B) in working register R6. The C flag is set to 1, and the Z, V, and S flags are set to 0.

```
SRA R6  
Op Code: D0 E6
```

**Example:** Register C6 contains the value DFH. Register DFH contains the value B8H (10111000B). The following statement leaves the value DCH (11011100B) in Register DFH. The C, Z, and V flags are reset to 0, and the S flag is set to 1.

```
SRA @C6  
Op Code: D1 C6
```

## SRP Set Register Pointer

**Instruction Format:**

SRP src

**Operation:**

RP ← src

The specified value is loaded into the Register Pointer (RP) Control Register (FDH). Bits 7-4 determine the working register group. Bits 3-0 selects the Memory Page. Addressing non-existent working register groups and memory pages results in undefined behavior.

**Table 3-17. Register Pointers, Working Register Groups, and Actual Registers**

Register Pointer (FDH) Contents (Bin)	Working Register Group (Hex)	Actual Registers (Hex)
1111 0000	F	F0-FF
1110 0000	E	E0-EF
1101 0000	D	D0-DF
1100 0000	C	C0-CF
1011 0000	B	B0-BF
1010 0000	A	A0-AF
1001 0000	9	90-9F
1000 0000	8	80-8F
0111 0000	7	70-7F
0110 0000	6	60-6F
0101 0000	5	50-5F
0100 0000	4	40-4F
0011 0000	3	30-3F
0010 0000	2	20-2F
0001 0000	1	10-1F
0000 0000	0	00-0F

## SRP Set Register Pointer

### Flags:

When the instruction is executed, the flags are set as follows:

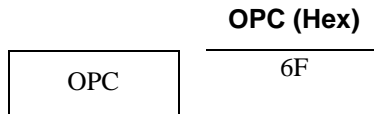
- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The following statement `SRP %70` assigns registers 070H through 07FH to be the current working register group, and, therefore, accessible as R0 through R15 in four bit addressing modes. The active memory page is set to page 0, and all eight-bit addressed register accesses are on page 0.

```
SRP %70  
Op Code: 31 F0
```

**STOP**  
**Stop****Instruction Format:**

STOP

**Operation:**

This instruction turns off the internal system clock (SCLK) and external crystal (XTAL) oscillator, and draws only standby current. The STOP mode is terminated by a RESET or Stop Mode Recovery (SMR) which causes the processor to restart the application program at address 0020H. The waken up source can be determined by reading the FLAGS register, specifically the SMR and WDT flags (see page 3–5 for more information).

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The following statements place the Z8 into STOP mode.

```
STOP
Op Codes: 6F
```

**NOTE:** Unlike the Z8, the Z8<sup>PLUS</sup> does not require a NOP before the STOP instruction.

## SUB Subtract

### Instruction Format:

SUB dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	22	r	r
			23	r	Ir
OPC	src	dst	24	R	R
			25	R	IR
OPC	dst	src	26	R	IM
			27	IR	IM

### Operation:

dst ← dst - src

The source operand is subtracted from the destination operand and the result is stored in the destination operand. The contents of the source operand are not changed. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: 0 if a value is carried from the most significant bit of the result; otherwise, 1, indicating a borrow.
- Z: 1 if the result is 0; otherwise, 0.
- V: 1 if arithmetic overflow occurred (if the operands have opposite sign and the sign of the result has the same as the source); reset otherwise.
- S: 1 if the result is negative; otherwise, 0.
- H: 0 if there is a carry from the most significant bit of the low-order four bits of the result; otherwise, 1, indicating a borrow.
- D: 1.



**SUB**  
**Subtract**

**Example:** Working register R3 contains 16H. Working register R11 contains 20H. The following statement leaves the value F6H in working register R3. The C, S, and D flags are set to 1, and the Z, V, and H flags are set to 0.

```
SUB R3, R11  
Op Code: 22 3B
```

**Example:** Working register R15 contains 16H. Working register R10 contains 20H. Register 20H contains 11H. The following statement leaves the value 05H in working register R15. The D flag is set to 1, and the C, Z, S, V, and H flags are set to 0.

```
SUB R16, @R10  
Op Code: 23 FA
```

**Example:** Register 34H contains 2EH. Register 12H contains 1BH. The following statement leaves the value 13H in register 34H. The D flag is set to 1, and the C, Z, S, V, and H flags are set to 0.

```
SUB 34H, 12H  
Op Code: 24 12 34
```

**Example:** Register 4BH contains 82H. Working register R3 contains 10H. Register 10H contains 01H. The following statement leaves the value 81H in register 4BH. The D and S flags are set to 1, and the C, Z, V, and H flags are set to 0.

```
SUB 4BH, @R3  
Op Code: 25 E3 4B
```

**Example:** Register 6CH contains 2AH. The following statement leaves the value 27H in register 6CH. The D flag is set to 1, and the C, Z, S, V, and H flags are set to 0.

```
SUB 6CH, #03H  
Op Code: 26 6C 03
```

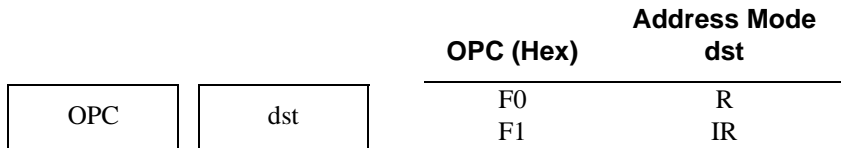
**Example:** Register D4H contains 5FH. Register 5FH contains 4CH. The following statement leaves the value 4AH in register 5FH. The D flag is set to 1, and the C, Z, S, V, and H flags are set to 0.

```
SUB @D4H, #02H  
Op Code: 17 D4 02
```

## SWAP Swap Nibbles

### Instruction Format:

SWAP dst



### Operation:

dst(7-4) ↔ dst(3-0)

The contents of the lower four bits and upper four bits of the destination operand are swapped.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Register BCH contains B3H (10110011B). The following statement leaves the value 3BH (00111011B) in register BCH. The Z and S flags are set to 0.

```
SWAP B3H
Op Code: F0 B3
```

**Example:** Working register R5 contains BCH and register BCH contains B3H (10110011B). The following statement leaves the value 3BH (00111011B) in register BCH. The Z and S flags are set to 0.

```
SWAP @R5H
Op Code: F1 E5
```

## TCM Test Complement Under Mask

### Instruction Format:

TCM dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	62	r	r
			63	r	Ir
OPC	src	dst	64	R	R
			65	R	IR
OPC	dst	src	66	R	IM
			67	IR	IM

### Operation:

(NOT dst) AND src

This instruction tests selected bits in the destination operand for a logical 1 value. The bits to be tested are specified by setting a 1 bit in the corresponding bit position in the source operand (the mask). The TCM instruction complements the destination operand, and then performs a logical AND operation using ANDs with the mask (source operand). The Zero (Z) flag can then be read to check the result. If the Z flag is set, then the tested bits were 1. When the TCM operation is complete, the destination and source operands still contain their previous values.

### Flags:

When the instruction is executed, the flags are set as follows::

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Working register R3 contains 45H (01000101B). Working register R7 contains the value 01H (00000001B) (bit 0 is being tested if it is 1). The following statement sets the Z flag indicating bit 0 in the destination operand is 1. The V and S flags are set to 0.

TCM R3, R7  
Op Code: 62 37

## TCM Test Complement Under Mask

**Example:** Working register R14 contains the value F3H (11110011B). Working register R5 contains CBH. Register CBH contains 88H (10001000B) (bit 7 and bit 3 are tested if they are 1). The following statement resets the Z flag to 0, because bit 3 in the destination operand is not a 1. The V and S flags are also set to 0.

```
TCM R14, @R5  
Op Code: 63 E5
```

**Example:** Register D4H contains the value 04H (000001000B). Working register R0 contains the value 80H (10000000B) (bit 7 is tested if it is 1). The following statement resets the Z flag to 0, because bit 7 in the destination operand is not a 1. The S flag is set to 1, and the V flag is set to 0.

```
TCM D4H, R0  
Op Code: 64 E0 D4
```

**Example:** Register DFH contains the value FFH (11111111B). Register 07H contains the value 1FH. Register 1FH contains the value BDH (10111101B) (bit 7, bit 5, bit 4, bit 3, bit 2, and bit 0 are tested if they are 1). The following statement sets the Z flag to 1 indicating the tested bits in the destination operand are 1. The S and V flags are set to 0.

```
TCM DFH, @07H  
Op Code: 65 07 DF
```

**Example:** Working register R13 contains the value F2H (11110010B). The following statement tests bit 1 of the destination operand for 1. The Z flag is set to 1 indicating bit 1 in the destination operand was 1. The S and V flags are set to 0.

```
TCM R13, #02H  
Op Code: 66 ED, 02
```

**Example:** Register 5DH contains A0H. Register A0H contains 0FH (00001111B). The statement tests bit 4 of the Register A0H for 1. The Z flag is reset to 0 indicating bit 1 in the destination operand was not 1. The S and V flags are set to 0.

```
TCM @5D, #10H  
Op Code: 67 5D 10
```

**TM**  
**Test Under Mask**

**Instruction Format:**

TM dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	72	r	r
			73	r	Ir
OPC	src	dst	74	R	R
			75	R	IR
OPC	dst	src	76	R	IM
			77	IR	IM

**Operation:** dst AND src

This instruction tests selected bits in the destination operand for a logical 0 value. The bits to be tested are specified by setting a 1 bit in the corresponding bit position in the source operand (the mask). The TM instruction ANDs the destination operand with the mask (the source operand). The Zero (Z) flag can then be read to check the result. If the Z flag is set, then the tested bits were 0. When the TM operation is complete, the destination and source operands still contain their previous values.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** Working register R3 contains 45H (01000101B). Working register R7 contains the value 02H (00000010B) (bit 1 is tested if it is 0). The following statement sets the Z flag to 1 indicating bit 1 in the destination operand is 0. The V and S flags are set to 0.

TM R3, R7  
Op Code: 72 37

## TM Test Under Mask

**Example:** Working register R14 contains the value F3H (11110011B). Working register R5 contains CBH. Register CBH contains 88H (10001000B) (bit 7 a bit 3 are tested if they are 0). The following statement resets the Z flag to 0, because bit 7 in the destination operand is not a 0. The S flag is set to 1, and the V flag is set to 0.

```
TM R14, @R5  
Op Code: 73 E5
```

**Example:** Register D4H contains the value 08H (00001000B). Working register R0 contains the value 04H (00000100B) (bit 2 is tested if it is 0). The statement sets the Z flag to 1, because bit 2 in the destination operand is a 0. The S and V flags are set to 0.

```
TM D4H, R0  
Op Code: 74 E0 D4
```

**Example:** Register DFH contains the value 00H (00000000B). Register 07H contains the value 1FH. Register 1FH contains the value BDH (10111101B) (bit 7, bit 5, bit 4, bit 3, bit 2, and bit 0 are tested if they are 0). The following statement sets the Z flag to 1, indicating the tested bits in the destination operand are 0. The S is set to 1, and the V flag is set to 0.

```
TM DFH, @07H  
Op Code: 75 07 DF
```

**Example:** Working register R13 contains the value F1H (11110001B). The following statement tests bit 1 of the destination operand for 0. The Z flag is set to 1, indicating bit 1 in the destination operand was 0. The S and V flags are set to 0.

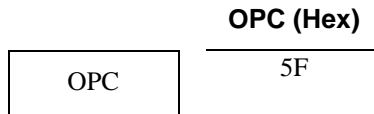
```
TM R13, #02H  
Op Code: 76 ED, 02
```

**Example:** Register 5DH contains A0H. Register A0H contains 0FH (00001111B). The following statement tests bit 4 of the register A0H for 0. The Z flag is set to 1, indicating bit 4 in the destination operand was 0. The S and V flags are set to 0.

```
TM @5D, #10H  
Op Code: 77 5D 10
```

**WDT**  
**Watch-Dog Timer****Instruction Format:**

WDT

**Operation:**

The Watch-Dog Timer (WDT) is a retriggerable one-shot timer that resets the device if it reaches its terminal count. Each execution of the WDT instruction refreshes the timer and prevents the WDT from timing out.

**Flags:**

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: The value set by the preceding instruction.
- S: The value set by the preceding instruction.
- V: The value set by the preceding instruction.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.

**Example:** The WDT is enabled. The following statement refreshes the Watch-Dog Timer.

```
WDT
Op Code : 5F
```

## XOR Logical Exclusive OR

### Instruction Format:

XOR dst, src

			OPC (Hex)	Address Mode	
				dst	src
OPC	dst	src	B2	r	r
			B3	r	Ir
OPC	src	dst	B4	R	R
			B5	R	IR
OPC	dst	src	B6	R	IM
			B7	IR	IM

### Operation:

dst ← dst XOR src

The source operand performs a logical EXCLUSIVE ORed operation, which stores a 1 in the destination operand whenever the corresponding bits in the two operands are different. The destination operand is set to 1; otherwise, a 0 is stored. The contents of the source operand are not changed.

### Flags:

When the instruction is executed, the flags are set as follows:

- C: The value set by the preceding instruction.
- Z: 1 if the result is 0; otherwise, 0.
- S: 1 if bit 7 of the result is 1; otherwise, 0.
- V: 0.
- D: The value set by the preceding instruction.
- H: The value set by the preceding instruction.



## XOR Logical Exclusive OR

**Example:** Working register R1 contains 38H (00111000B). Working register R14 contains 8DH (10001101B). The following statement leaves the value B5H (10110101B) in working register R1. The Z, and V flags are set to 0, and the S flag is set to 1.

```
XOR R1, R14  
Op Code: B2 1E
```

**Example:** Working register R4 contains F9H (11111001B). Working register R13 contains 7BH. Register 7B contains 6AH (01101010B). The following statement leaves the value 93H (10010011B) in working register R4. The S flag is set to 1, and the Z and V flags are set to 0.

```
XOR R4, @R13  
Op Code: B3 4D
```

**Example:** Register 3AH contains the value F5H (11110101B). Register 42H contains the value 0AH (00001010B). The following statement leaves the value FFH (11111111B) in register 3AH. The S flag is set to 1, and the C and V flags are set to 0.

```
XOR 3AH, 42H  
Op Code: B4 42 3A
```

**Example:** Working register R5 contains F0H (11110000B). Register 45H contains 3AH. Register 3A contains 7F (01111111B). The statement leaves the value 8FH (10001111B) in working register R5. The S flag is set to 1, and the C and V flags are set to 0.

```
XOR R5, @45H  
Op Code: B5 45 E5
```

**Example:** Register 7AH contains the value F7H (11110111B). The following statement leaves the value 07H (00000111B) in register 7AH. The Z, V, and S flags are set to 0.

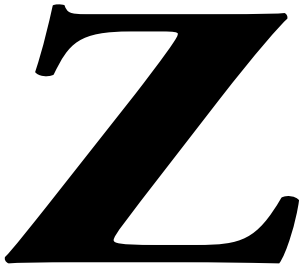
```
XOR 7AH, #F0H  
Op Code: B6 7A F0
```

**Example:** Working register R3 contains the value 3EH. Register 3EH contains the value 6CH (01101100B). The following statement leaves the value 69H (01101001B) in register 3EH. The Z, V, and S flags are set to 0.

```
XOR @R3, #05H  
Op Code: B7 E3 05
```

---

---



## CHAPTER 4 INTERRUPTS

### INTRODUCTION

The Z8<sup>PLUS</sup> core allows 15 different interrupts from a variety of sources:

- external inputs
- on-chip peripherals
- software

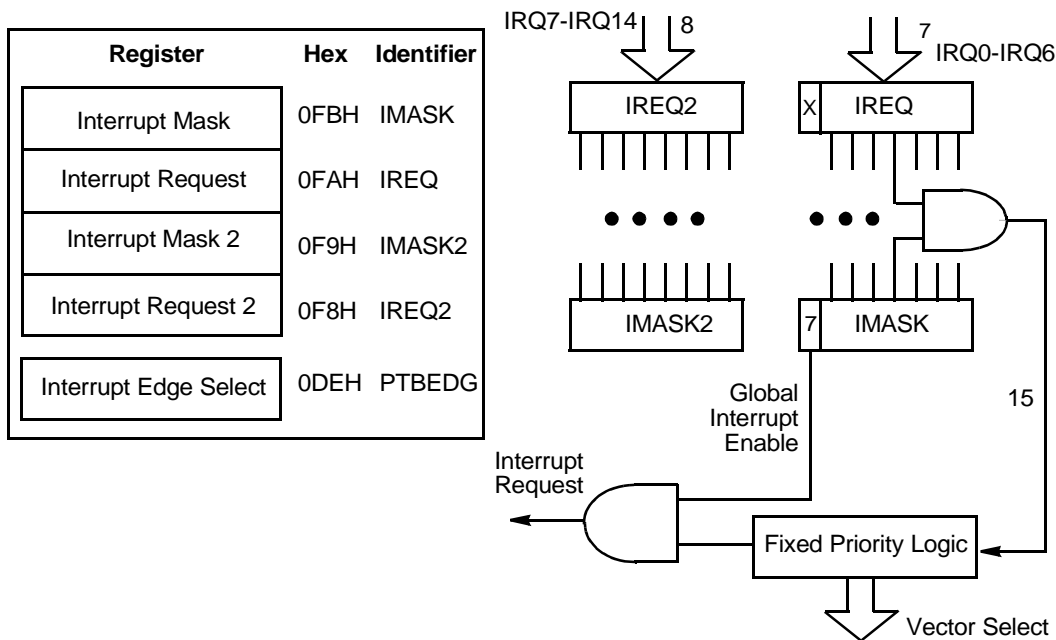
Interrupts can be masked by using the Interrupt Mask Register. All interrupts can be globally disabled by setting the master Interrupt Enable, bit 7 in the Interrupt Mask Register, to 0, with a Disable Interrupt (DI) instruction. Interrupts are globally enabled by setting bit 7 to 1 with an Enable Interrupt (EI) instruction.

There are four interrupt control registers: the Interrupt Request Registers (IREQ and IREQ2) and the Interrupt Mask registers (IMASK and IMASK2). Figure 4-1 shows addresses and identifiers for the interrupt control registers. Figure 4-2 is a block diagram showing the Interrupt Mask and Interrupt Priority logic.

Register	HEX	Identifier
Interrupt Mask	0FBH	IMASK
Interrupt Request	0FAH	IREQ
Interrupt Mask 2	0F9H	IMASK2
Interrupt Request 2	0F8H	IREQ2

Figure 4-1. Interrupt Control Register Addresses and Identifiers

The Z8<sup>PLUS</sup> MCU family supports both vectored and polled interrupt handling. Details on vectored and polled interrupts can be found later in this chapter.



**Figure 4-2. Interrupt Block Diagram**

**NOTE:** See the selected Z8<sup>PLUS</sup> MCU's product specification for the exact interrupt sources supported.

## INTERRUPT SOURCES

Table 4-1 presents the interrupt types, sources, and vectors available in the Z8E001. Other processors from the Z8<sup>PLUS</sup> family may define the interrupts differently.

**Table 4-1. Z8E001 Interrupt Types, Sources, and Vectors**

Name	Sources	Vector Location	Comments	Fixed Priority
IREQ <sub>0</sub>	Timer0 Time-out	2,3	Internal	1 (Highest)
IREQ <sub>1</sub>	PB4 High-to-Low Transition	4,5	External (PB4), Edge Triggered	2
IREQ <sub>2</sub>	Timer1 Time-out	6,7	Internal	3
IREQ <sub>3</sub>	PB2 High-to-Low Transition	8,9	External (PB2), Edge Triggered	4
IREQ <sub>4</sub>	PB4 Low-to-High Transition	A,B	External (PB4), Edge Triggered	5
IREQ <sub>5</sub>	Timer2 Time-out	C,D	Internal	6 (Lowest)
IREQ <sub>6</sub> - IREQ <sub>15</sub>	Reserved		Reserved for future expansion	

### External Interrupt Sources

External sources can be generated by a transition on the corresponding Port pin. The interrupt may detect a rising edge, a falling edge, or both.

#### NOTES:

1. The interrupt sources and trigger conditions are device dependent. See the device product specification to determine available sources (internal and external), triggering edge options, and exact programming details.
2. Although interrupts are edge triggered, minimum interrupt request Low and High times must be observed for proper operation. See the device product specification for exact timing requirements on external interrupt requests ( $T_{WIL}$ ,  $T_{WIH}$ ).

## Internal Interrupt Sources

Internal interrupt sources and trigger conditions are device dependent. On-chip peripherals may set interrupt under various conditions. Some peripherals always set their corresponding IREQ bit while others must be specifically configured to do so.

See the device product specification to determine available sources, triggering edge options, and exact programming details. For more details on the interrupt sources, refer to the chapters describing the timers, comparators, I/O ports, and other peripherals.

## INTERRUPT REQUEST (IREQ) REGISTER LOGIC AND TIMING

The Z8<sup>PLUS</sup> core responds to interrupts as it retires each instruction. If an unmasked interrupt is detected as an instruction is being retired, the Z8<sup>PLUS</sup> core does not execute an instruction during the next instruction cycle. The Z8<sup>PLUS</sup> MCU instead selects the highest priority outstanding interrupt to be serviced. The program counter and flags register are pushed to the stack during the next instruction cycle. The appropriate IREQ bit is cleared, the master enable is cleared and the MCU fetches the interrupt vector from program memory. It then jumps to the user's interrupt routine during the following cycle (See Figure 4-3).

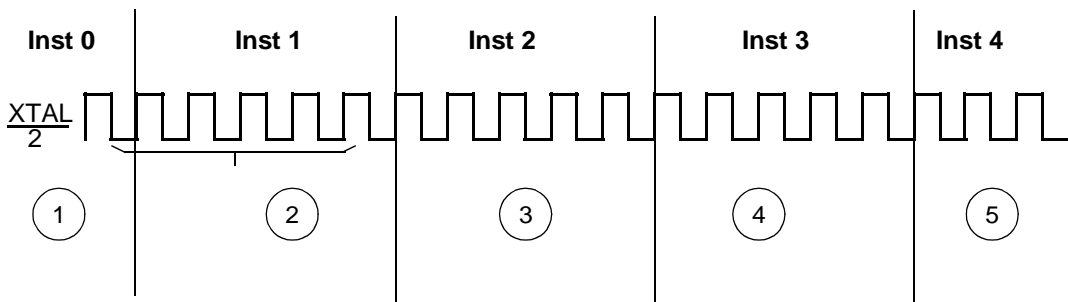


Figure 4-3. Interrupt Service Sequence

### NOTES:

1. There are no outstanding, unmasked interrupts.
2. Interrupt source sets an IREQ bit during this interval. This bit is highest priority, has an unmasked IREQ, and is bit-sampled.
3. PC and flags are pushed, IREQ bit cleared, IMASK (7) cleared, and vector fetched.
4. JUMP to interrupt vector.
5. This portion is the first instruction of user's interrupt service routine.

## Interrupt Mask Register (IMASK) Initialization

The IMASK register individually or globally enables or disables the interrupts (see Figure 4-4). When bits 0 through bit 6 are set to 1, the corresponding interrupt requests are enabled. The IMASK2 register, bits 0 through 7, enable and disable IRQ7 through IRQ14, respectively. Bit 7 is the master enable bit and must be set before any of the individual interrupt requests can be recognized. Resetting bit 7 disables all the interrupt requests. Bit 7 is set and reset by the EI and DI instructions. It is automatically set to 0 during an interrupt service routine and set to 1 following the execution of an Interrupt Return (IRET) instruction. The IMASK registers are reset to 00H, disabling all interrupts.

### NOTE:

1. It is not good programming practice to directly assign a value to the master enable bit. A value change should always be accomplished by issuing the EI and DI instructions.
2. Care should be taken not to set or clear IMASK bits while the master enable is set.

**Figure 4-4. Interrupt Mask Register**

**Interrupt Mask Register—IMASK (FBH)**

Bit	7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
R = Read W = Write X = Indeterminate U = Undefined/Undetermined								

Bit Position	R/W	Value	Description
7		0	Disables Interrupts
		1	Enables Interrupts
6		0	Disables IRQ5
		1	Enables IRQ5
5		0	Disables IRQ5
		1	Enables IRQ5
4		0	Disables IRQ4
		1	Enables IRQ4
3		0	Disables IRQ3
		1	Enables IRQ3
2		0	Disables IRQ2
		1	Enables IRQ2
1		0	Disables IRQ1
		1	Enables IRQ1
0		0	Disables IRQ0
		1	Enables IRQ0



**Figure 4-5. Interrupt Mask 2 Register****Interrupt Mask 2 Register–IMASK2 (F9H)**

Bit	7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
R = Read W = Write X = Indeterminate U = Undefined/Undetermined								

Bit Position	R/W	Value	Description
7	R/W	0 1	Disables IRQ14 Enables IRQ14
6	R/W	0 1	Disables IRQ13 Enables IRQ13
5	R/W	0 1	Disables IRQ12 Enables IRQ12
4	R/W	0 1	Disables IRQ11 Enables IRQ11
3	R/W	0 1	Disables IRQ10 Enables IRQ10
2	R/W	0 1	Disables IRQ9 Enables IRQ9
1	R/W	0 1	Disables IRQ8 Enables IRQ8
0	R/W	0 1	Disables IRQ7 Enables IRQ7

## Interrupt Request (IREQ) Register Initialization

IREQ (see Figure 4-6) is a register that stores the interrupt requests for both vectored and polled interrupts. When an interrupt is issued, the corresponding bit position in the register is set to 1. Bit 0 to bit 5 are assigned to interrupt requests IREQ0 to IREQ5, respectively.

Whenever RESET is executed, the IREQ register is set to 00H.

**Figure 4-6. Interrupt Request Register.**

### Interrupt Request Register–IREQ (FAH)

Bit	7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
R = Read W = Write X = Indeterminate U = Undefined/Undetermined								

Bit Position	R/W	Value	Description
7	R/W	0	Reserved, must be 0
6	R/W	0 1	IRQ6 reset IRQ6 set
5	R/W	0 1	IRQ5 reset IRQ5 set
4	R/W	0 1	IRQ4 reset IRQ4 set
3	R/W	0 1	IRQ3 reset IRQ3 set
2	R/W	0 1	IRQ2 reset IRQ2 set
1	R/W	0 1	IRQ1 reset IRQ1 set
0	R/W	0 1	IRQ0 reset IRQ0 set

Figure 4-7. Interrupt Request Register 2

## Interrupt Request Register 2–IREQ2 (F8H)

Bit	7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
R = Read W = Write X = Indeterminate U = Undefined/Undetermined								

Bit Position	R/W	Value	Description
7	R/W	0	IRQ14 reset
		1	IRQ14 set
6	R/W	0	IRQ13 reset
		1	IRQ13 set
5	R/W	0	IRQ12 reset
		1	IRQ12 set
4	R/W	0	IRQ11 reset
		1	IRQ11 set
3	R/W	0	IRQ10 reset
		1	IRQ10 set
2	R/W	0	IRQ9 reset
		1	IRQ9 set
1	R/W	0	IRQ8 reset
		1	IRQ8 set
0	R/W	0	IRQ7 reset
		1	IRQ7 set

---

## IREQ SOFTWARE INTERRUPT GENERATION

IREQ can be used to generate software interrupts by specifying IREQ as the destination of any instruction referencing the Z8<sup>PLUS</sup> Standard Register File. These software interrupts (SWI) are controlled in the same manner as hardware generated requests. In other words, the IMASK controls the enabling of each SWI.

To generate a SWI, the request bit in IREQ is set by the following statement:

```
OR IREQ, #NUMBER
```

The immediate data variable, NUMBER, has a 1 in the bit position corresponding to the required level of SWI. For example, an SWI must be issued when an IREQ5 occurs. Bit 5 of NUMBER must have a value of 1.

```
OR IREQ, #00100000B
```

If the interrupt system is globally enabled, IREQ5 is enabled, and there are no higher priority requests pending, control is transferred to the service routine pointed to by the IREQ5 vector.

**NOTE:** Note that software may modify the IREQ register at any time. Care should be taken when using any instruction that modifies the IREQ register while interrupt sources are active. The software writeback always takes precedence over the hardware. If a software writeback takes place on the same cycle as an interrupt source tries to set an IREQ bit, the new interrupt is lost.

---

## VECTORED PROCESSING

Each Z8<sup>PLUS</sup> interrupt level has its own vector. When an interrupt occurs, control passes to the service routine pointed to by the interrupt's vector location in program memory. The sequence of events for vectored interrupts is as follows:

- PUSH the PC Low Byte on the Stack
- PUSH the PC High Byte on the Stack
- PUSH the FLAGS on the Stack
- Disable Global Interrupts (bit 7 of IMASK)
- Fetch the High Byte of the Vector
- Fetch the Low Byte of the Vector
- Branch to the Service Routine specified by Vector

Figure 4-8 and Figure 4-9 show vectored interrupt operation.

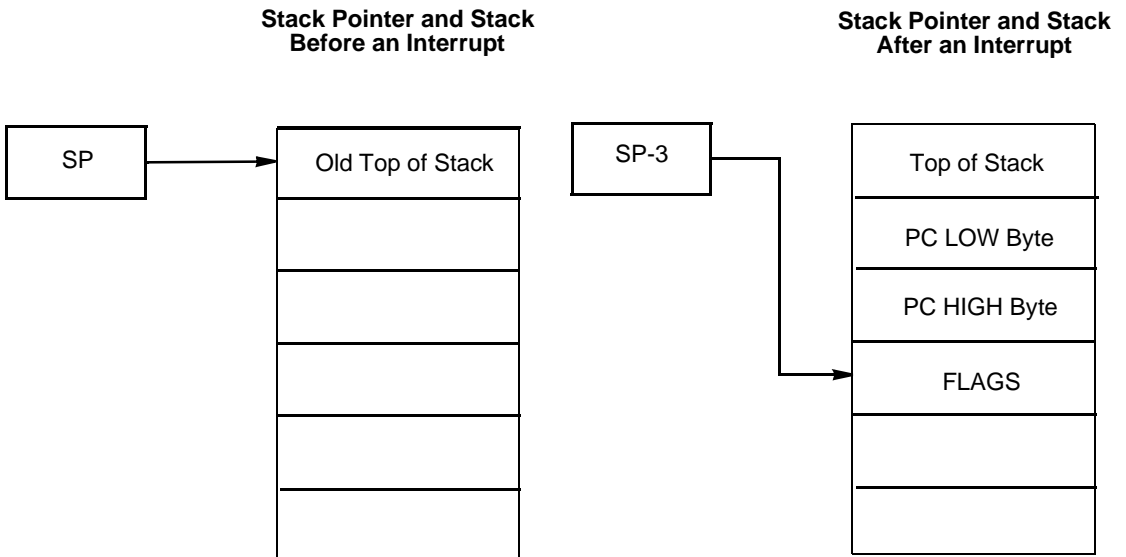


Figure 4-8. Stacks Before and After Interrupt

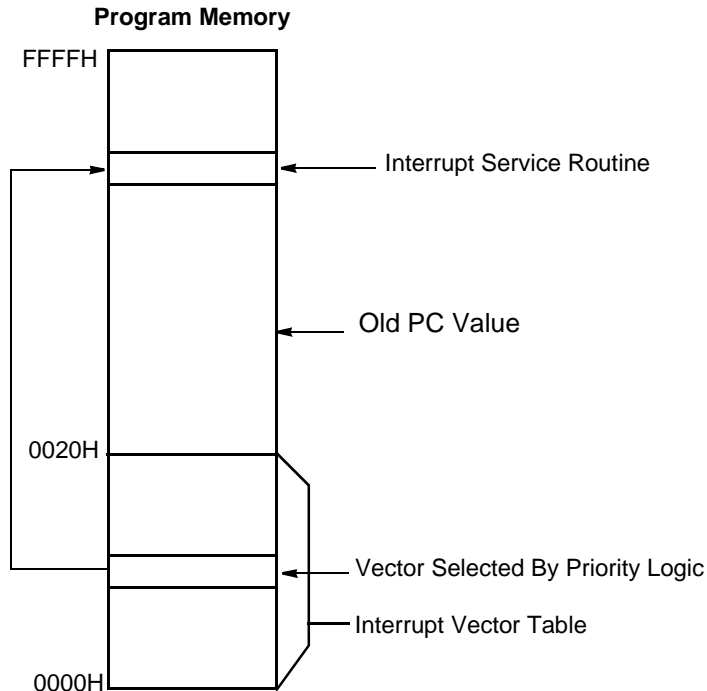


Figure 4-9. Interrupt Vector Table Location

## Nesting of Vectored Interrupts

Nesting vectored interrupts allows higher priority requests to interrupt a lower priority request. To initiate vectored interrupt nesting, perform the following steps during the interrupt service routine:

- PUSH the old IMASK on the stack.
- Load IMASK with a new mask to disable lower priority interrupts.
- Execute an EI instruction.
- Proceed with interrupt processing.
- Execute a DI instruction after processing is complete.
- Restore the IMASK to its original value by POPing the previous mask from the stack.
- Execute IRET.

Depending on the application, some simplification of the above procedure may be possible.

## POLLED PROCESSING

Polled interrupt processing is supported by masking off the IREQ to be polled. This process is accomplished by setting the corresponding bits in the IMASK to 0.

To initiate polled processing, check the appropriate bits in the IREQ using the Test Under Mask (TM) instruction. If the bit is set to 1, call or branch to the service routine. The service routine services the request, resets its Request Bit in the IREQ, and branches or returns back to the main program. An example of a polling routine is as follows:

```
TM IREQ,#MASKA;Test for request

JR Z, NEXT;If no request go to NEXT

CALL SERVICE;If request is there,then
;service it

NEXT:
.
.
.

SERVICE:;Process Request
.
.
.

AND IREQ, #MASKB ;Clear Request Bit

RET;Return to next
```

In this example, if IREQ2 is being polled, MASKA is 00000100B and MASKB is 11111011B.

---

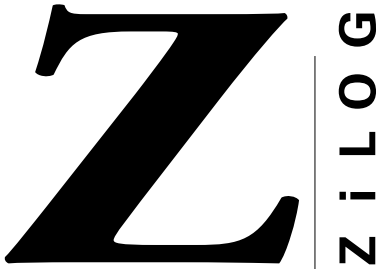
## RESET CONDITIONS

The IMASK and IREQ registers initialize to 00H on RESET.

---

---





*Totally Logical*

## APPENDIX A

### ACCESSING THE ZBBS/INTERNET

---

#### **BULLETIN BOARD INFORMATION**

The ZiLOG Bulletin Board Service (ZBBS) currently provides basic information on ZiLOG products and includes a ROM CODE upload area. In addition, the ZBBS provides valuable information on items of interest, such as ZiLOG specialty software and documentation.

#### **How to Access the ZBBS**

The ZBBS can be reached by dialing 1-408-558-8890. The ZBBS supports speeds up to 28.8K Baud with connections 8-N-1 (8 bits, No parity, 1 stop bit). We recommend that you use an ANSI/BBS terminal emulation setup.

To preview information or download files, follow the on-screen instructions.

The latest production released version of the Z8 GUI software can be downloaded from this site.

---

#### **ZiLOG ON THE INTERNET**

ZiLOG has a Home Page on the Internet. The Home Page address is:

<http://www.zilog.com>

The ZiLOG Home Page includes valuable information about hardware and software development tools. The latest production released version of the Z8 GUI software can be downloaded from this site.

---

---



Z8<sup>PLUS</sup> USER'S MANUAL

*Totally Logical*

**PROBLEM/SUGGESTION REPORT FORM**

If you experience any problems while operating this product, or if you note any inaccuracies while reading the User's Manual, please copy this form, fill it out, then mail or fax it to ZiLOG (see "Return Information"). We also welcome your suggestions!

**Customer Information**

Name	_____	Country	_____
Company	_____	Telephone	_____
Address	_____	Fax Number	_____
City/State/ZIP	_____	E-Mail Address	_____

**Product Information**

Serial # or Board Fab #/Rev. #  
Software Version  
Manual Number  
Host Computer Description/Type

**Return Information**

ZiLOG, Inc.  
System Test/Customer Support  
910 E. Hamilton Ave., Suite 110, MS 4-3  
Campbell, CA 95008  
Fax Number: (408) 558-8536  
Email: tools@zilog.com

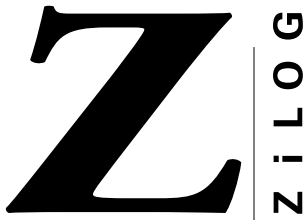
**Problem Description or Suggestion**

Provide a complete description of the problem or your suggestion. If you are reporting a specific problem, include all steps leading up to the occurrence of the problem. Attach additional pages as necessary.

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

---

---



*Totally Logical*

## INDEX

- A**
- add (ADD) 3-23
  - add with carry (ADC) 3-20
  - addressing
    - 12-bit 2-1
    - 16-bit 2-1
    - 4-bit address 2-1
    - 8-bit address 2-1
    - direct 2-7
    - immediate data 2-9
    - indexed 2-5
    - indirect register 2-3
    - modes 2-1
    - register 2-2
    - relative 2-8
  - arithmetic instructions
    - add (ADD) 3-2, 3-23
    - add with carry (ADC) 3-2, 3-20
    - compare (CP) 3-2, 3-32
    - decimal adjust (DA) 3-2, 3-34
    - decrement (DEC) 3-2, 3-37
    - decrement word (DECW) 3-2, 3-38
    - increment (INC) 3-2, 3-44
    - increment word (INCW) 3-2, 3-46
    - subtract (SUB) 3-2, 3-82
    - subtract with carry (SBC) 3-2, 3-74
  - assembly language syntax 3-12
- B**
- binary encoding 3-10
  - bit manipulation instructions
    - bit clear (AND) 3-3
    - bit complement (XOR) 3-3
    - bit set (OR) 3-3
    - test complement under mask (TCM) 3-3, 3-85
    - test under mask (TM) 3-3, 3-87
  - block diagram, interrupt 4-2
- C**
- call procedure (CALL) 3-27
  - carry flag (C) 3-5
  - clear (CLR) 3-30
  - compare (CP) 3-32
  - complement (COM) 3-31
  - complement carry flag (CCF) 3-29
  - condition codes (cc) 3-7, 3-8
  - control
    - core registers 1-3
    - peripheral registers 1-10
    - registers 1-10
  - CPU control instructions 3-4
    - complement carry flag (CCF) 3-4, 3-29
    - disable interrupts (DI) 3-4, 3-39
    - enable interrupts (EI) 3-4
    - halt (HALT) 3-4, 3-43
    - no operation (NOP) 3-4, 3-59
    - reset carry flag (RCF) 3-4, 3-64
    - set carry flag (SCF) 3-4
    - set carry flag(SCF) 3-76
    - set register pointer (SRP) 3-4, 3-79
    - stop (STOP) 3-4, 3-81
    - watch-dog timer (WDT) 3-4, 3-89

## D

- decimal adjust
  - DA instruction 3-2, 3-34
  - flag 3-6
- decrement
  - and jump if non-zero (DJNZ) 3-40
  - DEC instruction 3-37
  - word (DECW) 3-38
- definitions
  - flag 3-7
  - flag settings 3-8
- destination operand (dst) 3-1
- direct addressing mode (DA) 2-7
- disable interrupts (DI) 3-39

## E

- enable interrupts (EI) 3-42
- encoding notation and binary 3-10
- external interrupt sources 4-3

## F

- flag
  - carry (C) 3-5
  - decimal adjust 3-6
  - definitions 3-7
  - half-carry 3-6
  - overflow 3-6
  - processor 3-5
  - register 3-5
  - settings definitions 3-8
  - sign 3-6
  - stop mode recovery 3-7
  - watch-dog timer 3-7
  - zero 3-5

## G

- general purpose registers 1-5

## H

- half-carry flag (H) 3-6
- halt (HALT) 3-43
- high nibble 1-6

## I

- immediate data addressing (IM) 2-9
- increment (INC) 3-44
- increment word (INCW) 3-46
- indexed addressing (X) 2-5
- indirect register addressing (IR) 2-3
- instructions
  - arithmetic 3-2
  - bit manipulation 3-3
  - block transfer 3-3
  - CPU control 3-4
  - load 3-2
  - logical 3-2
  - program control 3-3
  - rotate and shift 3-4
  - summary 3-12
- internal interrupt sources 4-4
- interrupt
  - block diagram 4-2
  - control registers 4-1
  - external sources 4-3
  - internal sources 4-4
  - mask register initialization 4-5
  - mask registers (IMASK) 4-1
  - polled 4-2
  - request register (IREQ) 4-1
  - request register initialization 4-7
  - request register logic and timing 4-4
  - return (IRET) 3-47
  - sources 4-3
  - vectored 4-2

## J

- jump (JP) 3-48
- jump relative (JR) 3-50

- L**
  - load 3-51
    - constant (LDC) 3-55
    - constant auto increment (LDCI) 3-3, 3-57
  - load instructions
    - clear (CLR) 3-2, 3-30
    - load (LD) 3-2, 3-51
    - load constant (LDC) 3-2, 3-55
    - pop (POP) 3-2, 3-62
    - push (PUSH) 3-2, 3-63
  - logical
    - AND (AND) 3-25
    - exclusive OR (XOR) 3-90
  - logical instructions 3-2
    - complement 3-2
    - complement (COM) 3-31
    - logical AND (AND) 3-2
    - logical and (AND) 3-25
    - logical exclusive OR (XOR) 3-2, 3-90
    - logical OR (OR) 3-2, 3-60
  - lower nibble values 3-17
- M**
  - memory
    - map 1-12
    - program 1-11
- N**
  - nibble
    - high 1-6
    - lower values 3-17
  - no operation (NOP) 3-59
  - notation and binary encoding 3-10
  - notational shorthand 3-10
- O**
  - opcode map 3-18
  - operand
    - destination 3-1
    - dst 3-1
    - source 3-1
    - src 3-1
  - overflow flag 3-6
- P**
  - peripheral registers 1-10
  - polled interrupt 4-2
  - pop (POP) 3-62
  - processor flags 3-5
  - program control instructions
    - IRET instruction 3-47
  - program control instructions
    - call procedure (CALL) 3-3, 3-27
    - decrement and jump if non-zero (DJNZ) 3-40
    - decrement and jump non-zero (DJNZ) 3-3
    - interrupt return (IRET) 3-3
    - jump (JP) 3-3, 3-48
    - jump relative (JR) 3-3, 3-50
    - return (RET) 3-3, 3-65
  - program memory 1-11
  - program memory map 1-12
- R**
  - register
    - addressing (R) 1-5, 2-2
    - control 1-10
    - control and peripheral 1-10
    - core control 1-3
    - file organization 1-4
    - file space 1-1
    - flag 3-5
    - general purpose 1-5
    - peripheral 1-10
    - pointer 1-6
    - stack pointer 1-13
    - working groups 1-6
  - relative addressing (RA) 2-8
  - reset carry flag (RCF) 3-64
  - return (RET) 3-65
  - rotate and shift instructions 3-4
    - rotate left (RL) 3-4, 3-66

rotate left through carry (RLC) 3-4, 3-68  
rotate right (RR) 3-4, 3-70  
rotate right through carry (RRC) 3-4, 3-72  
shift right arithmetic (SRA) 3-4, 3-77  
swap nibbles (SWAP) 3-4, 3-84

## **S**

set  
    set carry flag (SCF) 3-76  
    set register pointer (SRP) 3-79  
shift right arithmetic (SRA) 3-77  
shorthand, notational 3-10  
sign flag (S) 3-6  
source operand (src) 3-1  
stack pointer register (SP) 1-13  
stop (STOP) 3-81  
stop mode recovery flag (SMR) 3-7  
subtract (SUB) 3-82  
subtract with carry (SBC) 3-74  
swap nibbles (SWAP) 3-84  
syntax assembly language 3-12

## **T**

test complement under mask (TCM) 3-85  
test under mask (TM) 3-87  
timer, watch-dog (WDT) 1-8, 3-89

## **V**

vectored interrupt 4-2

## **W**

watch-dog timer 1-8  
watch-dog timer (WDT) 3-89  
watch-dog timer flag (WDT) 3-7  
working register groups 1-6

## **Z**

zero flag (Z) 3-5