ST7 FAMILY

PROGRAMMING MANUAL

INTRODUCTION

The ST7 family of HCMOS Microcontrollers has been designed and built around an industry standard 8-bit core and a library of peripheral blocks, which include ROM, EPROM, RAM, EEPROM, I/O, Serial Interfaces (SPI, SCI, I2C,...), 16-bit Timers, etc. These blocks may be assembled in various combinations in order to provide cost-effective solutions for application dedicated products.

The ST7 family forms part of the STMicroelectronics 8-bit MCU product line, and finds place in a wide variety of applications such as automotive systems, remote controls, video monitors, car radio and numerous other consumer, industrial, telecom, multimedia and automotive products.

ST7 ARCHITECTURE

The 8-bit ST7 Core is designed for high code efficiency. It contains 6 internal registers, 17 main addressing modes and 63 instructions. The 6 internal registers include 2 Index registers, an Accumulator, a 16-bit Program Counter, a Stack Pointer and a Condition Code register. The two Index registers X and Y enable Indexed Addressing modes with or without offset, along with read-modify-write type data manipulations. These registers simplify branching routines and data modifications.

The 16-bit Program Counter is able to address up to 64K of ROM/EPROM memory. The 6-bit Stack Pointer provides access to a 64-level Stack and an upgrade to an 8-bit Stack Pointer is foreseen in order to be able to manage a 256-level Stack. The Core also includes a Condition Code Register providing 5 Condition Flags that indicate the result of the last instruction executed.

The 17 main Addressing modes, including Indirect Relative and Indexed addressing, allow sophisticated branching routines or CASE-type functions. The Indexed Indirect Addressing mode, for instance, permits look-up tables to be located anywhere in the address space, thus enabling very flexible programming and compact C-based code.

The 63-instruction Instruction Set is 8-bit oriented with a 2-byte average instruction size. This Instruction Set offers, in addition to standard data movement and logic/arithmetic functions, byte multiplication, bit manipulation, data transfer between Stack and Accumulator (Push/Pop) with direct stack access, as well as data transfer using the X and Y registers.

Depending of the target device, different methods of Interrupt priority management may be selected: the number of Interrupt vectors can vary from 6 to 16, and the priority level may be managed by software on some versions. Some peripherals include Direct Memory Access (DMA) between serial interfaces and memory.

Power-saving may be managed under program control by placing the device in WAIT or HALT mode.

A high test coverage is achieved for ST7 family devices thanks to the use of an autotest method based on "Cyclic Redundancy Checking" (CRC). This approach is based on the analysis of a data flow comprising not only input and output, but also internal data, which affords a detailed inside view of the behaviour of the core and of the peripherals.

Rev. 1.1

March 1999

Table of Contents

ADDITIONAL BLOCKS

The additional blocks take the form of integrated hardware peripherals arranged around the central processor core. The following list details the features of some of the currently available blocks:

ROM — User ROM, in sizes up to 64K

EPROM — EPROM based devices, same sizes as ROM

RAM — Sizes up to several K byte

EEPROM — Sizes up to several K byte. Erase/programming operations do not require additional external power sources. Up to 32 bytes can be programmed or erased simultaneously.

TIMER — Different versions based on a 16-bit free running timer/counter are available. They can be coupled with either input captures, output compares or PWM facilities.

PWM — Software programmable duty cycle between 0% to 100% in up to 1024 steps. The outputs can be filtered to provide D/A conversion.

A/D CONVERTER — The Analog to Digital Converter uses a sample and hold technique. It has an 8-bit range.

I2C — Multi/master, single master, single slave modes, DMA or 1byte transfer, standard and fast I2C modes, 7 and 10-bit addressing.

SPI — The Serial peripheral Interface is a fully synchronous 4 wire interface ideal for Master and Slave applications such as driving devices with input shift register (LCD driver, external memory,...).

SCI — The Serial Communication Interface is a fast asynchronous interface which features both duplex transmission, NRZ format, programmable baud rates and standard error detection. The SCI can also emulate RS232 protocol.

WATCHDOG — It has the ability to induce a full reset of the MCU if its counter counts down to zero prior to being reset by the software. This feature is especially useful in noisy applications.

I/O PORTS — They are programmable from software to act in several input or output configurations on an individual line basis including high current and interrupt generation. The basic block has eight CMOS/TTL compatible lines.

LCD — Liquid Crystal Display drive with simple addressing in RAM and drive capability from 1 to 16 multiplexing rates.

Static DAC — True Digital to Analog Converter with up to 12-bit resolution.

DDC — Complete DDC interface for "plug and play" multimedia applications

SYNC PROC. — East/West deflection and synchronization processor for digital monitors.

RDS — Complete RDS decoder embedded in the device for radio applications.

New blocks are continuously being added to the peripheral block library to meet customers' specific needs with regard to the optimal integration level in high volume projects.

ST7 DEVELOPMENT SUPPORT

The ST7 family of MCUs is supported by a comprehensive range of development tools. This family presently comprises hardware tools (emulators, programmers), a software package (assembler-linker, debugger, archiver) and a C-compiler development tool.

The PC-compatible host system forms the platform for the assembler/linker and for the symbolic debugger; it also controls the emulator and enables object code to be downloaded through the RS232 serial link.

The real-time emulator is connected through the probe to the target application. It provides the proper electrical connections, thus allowing duplication of the MCU's functions in the target system. The user program can be executed in real time, in step-by-step mode, or by stepping over call modes.

Breakpoints can be included on instructions, on memory addresses, on address ranges, on the state of one of two output triggers, as well as in trap mode (automatic reset). A logical analyser mode can record four input signals as external events, using a 1K x 32-bit trace and six recording modes, with or without breakpoints. In addition, two output signals are available for synchronisation and for timing measurement.

Each member of the ST7 family has an exclusive probe, dedicated to the device and package. A remote programming tool is available to program EPROM and OTP devices independently (Epromer) or by batches of 10 parts (Gang programmer).

The ST7 assembler is used to translate the source code into relocatable machine code. It accepts a source file written in ST7 assembly language and transforms it into a linkable object file. The assembler recognizes the use of symbols, macros and conditional assembly directives. The ST7 linker/loader combines a number of object files into a single program, associating an absolute address to each section of code. It generates a binary file, containing the image of the ST7 EPROM or ROM memory content. The ST7 library archiver maintains libraries of software object files. Libraries may be used as entry for the linker/loader, together with object files. This allows the user to develop standard modules for repetitive use. The ST7 executable file formatter is responsible for generating an executable file. This file can be downloaded to the emulator or, via the debugger, to the Eprom or OTP device for evaluation and small volume production with the programmer or sent to STMicroelectronics for production of ROM parts.

A C-compiler development environment is also available. It includes an editor, a compiler, a linker, a debugger and a simulator which are Windows™ compatible, and take full advantage of the ST7 architecture to generate excellent quality code.

1 GLOSSARY

mnem	mnemonic
src	source
dst	destination
cy	duration of the instruction in CPU clock cycles (internal clock)
lgth	length of the instruction in byte(s)
op-code	instruction byte(s) implementation (1..4 bytes)
mem	memory location
byte	a byte
short	represent a short 8-bit addressing mode
long	represent a long 16-bit addressing mode
EA	Effective Address: The final computed data byte address
Page Zero	all data located at [00..FF] addressing space (single byte address)
(XX)	content of a memory location XX
XX	a byte value
MS	Most Significant Byte of a 16-bit value (MSB)
LS	Least Significant Byte of a 16-bit value (LSB) AAccumulator Register
X	X Index Register
Y	Y Index Register
reg	A, X or Y register
ndx	index register, either X or Y
PC	16-bit Program Counter Register
SP	16-bit Stack Pointer
S	Stack Pointer LSB
CC	Condition Code Register:

1	1	1	H	I	N	Z	C

For each instruction, we show how it affects the CC flags:

Nothing	Flag not affected Flag NameFlag affected 0Flag cleared 1Flag set

Example:

H	I	N	Z	C
	0	N	Z	1

See the Core Description for further details on the CC Register content

2 ST7 CORE DESCRIPTION

2.1 INTRODUCTION

The CPU has a full 8-bit architecture. Six internal registers allow efficient 8-bit data manipulations. The CPU is able to execute 63 basic instructions. It features 17 main addressing modes and can address 6 internal registers.

2.2 CPU Registers

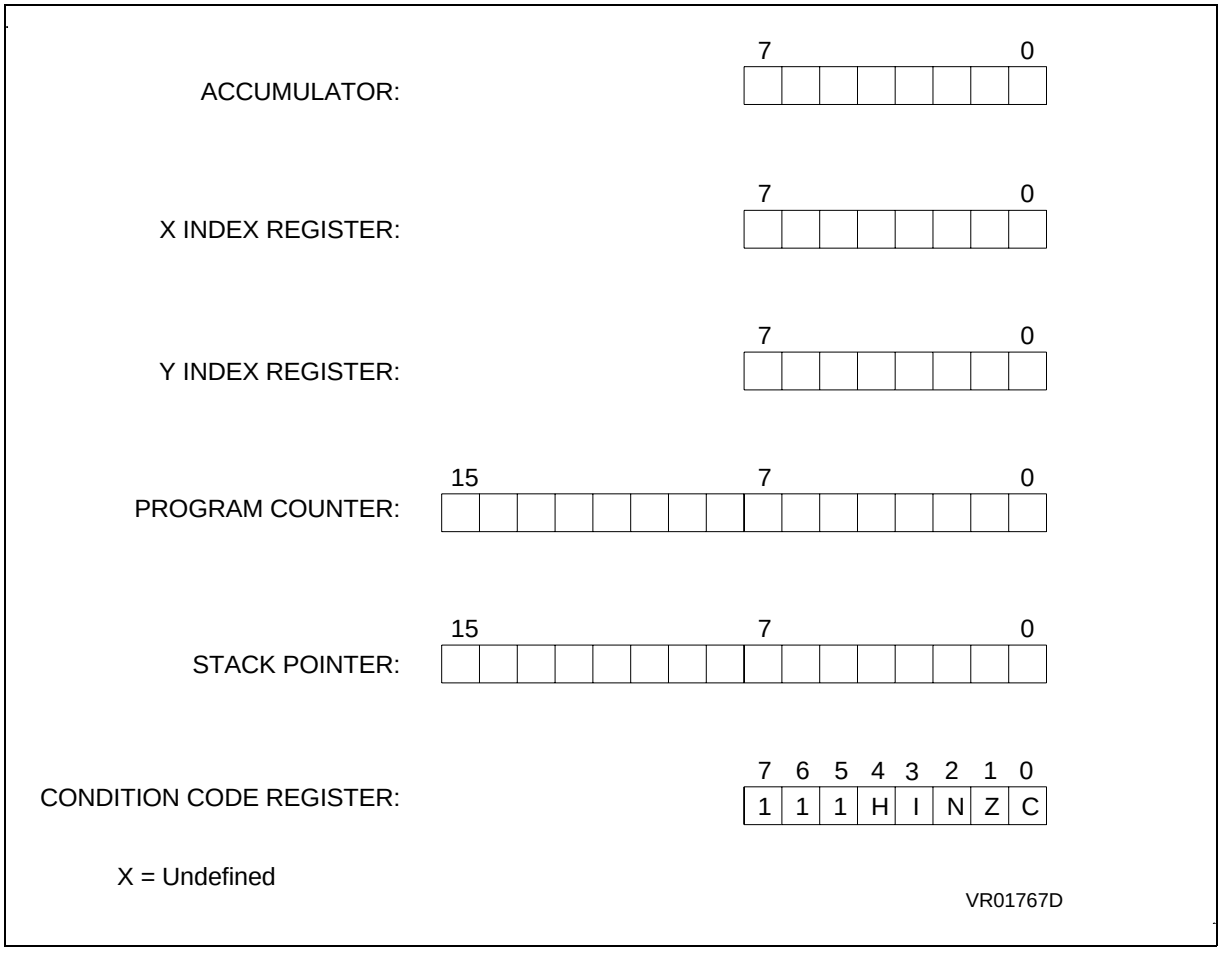
The 6 CPU registers are shown in the programming model in Figure 1.. Following an interrupt, the registers are pushed onto the stack in the order shown in Figure 2.. They are popped from stack in the reverse order. The Y register is not affected by these automatic procedures. The interrupt routine must therefore handle it, if needed, through the POP and PUSH instructions.

Accumulator (A). The accumulator is an 8-bit general purpose register used to hold operands and the results of the arithmetic and logic calculations as well as data manipulations.

Index Registers (X and Y). These 8-bit registers are used to create effective addresses or as temporary storage area for data manipulations. The cross assembler generates a PRECEDE instruction (PRE) to indicate that the following instruction refers to the Y register. The Y register is never automatically stacked. Interrupt routines must push or pop it by using the POP and PUSH instructions.

Program Counter (PC). The program counter is a 16-bit register used to store the address of the next instruction to be executed by the CPU. It is automatically refreshed after each processed instruction. As a result, the ST7 core can access up to 64 kb of memory.

Figure 1. Programming Mode



Stack Pointer (SP):

The stack pointer is a 16-bit register. The 6 least significant bits contain the address of the next free location of the stack. The 10 most significant bits are forced to a preset value. They are reserved for future extension of ST72 family.

The stack is used to save the CPU context on subroutines calls or interrupts. The user can also directly use it through the POP and PUSH instructions.

After an MCU reset, or after the Reset Stack Pointer instruction (RSP), the Stack Pointer is set to its upper value. It is then decremented after data has been pushed onto the stack and incremented after data is popped from the stack. When the lower limit is exceeded, the stack pointer wraps around to the stack upper limit. The previously stored information is then overwritten, and therefore lost.

A subroutine call occupies two locations and an interrupt five locations.

Condition Code Register (CC):

The Condition Code register is a 5-bit register which indicates the result of the instruction just executed as well as the state of the processor. These bits can be individually tested by a program and specified action taken as a result of their state. The following paragraphs describe each bit.

Half carry bit (H):

The H bit is set to 1 when a carry occurs between the bits 3 and 4 of the ALU during an ADD or ADC instruction. The H bit is useful in BCD arithmetic subroutines.

Interrupt mask (I):

When the I bit is set to 1, all interrupts are disabled. Clearing this bit enables them. Interrupts requested while I is set, are latched and can be processed when I is cleared (only one interrupt request per interrupt enable flag can be latched). This bit can be set/reset by software and is automatically set after reset or at the beginning of an interrupt routine.

Negative (N):

When set to 1, this bit indicates that the result of the last arithmetic, logical or data manipulation is negative (i.e. the most significant bit is a logic 1).

Zero (Z):

When set to 1, this bit indicates that the result of the last arithmetic, logical or data manipulation is zero.

Carry/Borrow (C):

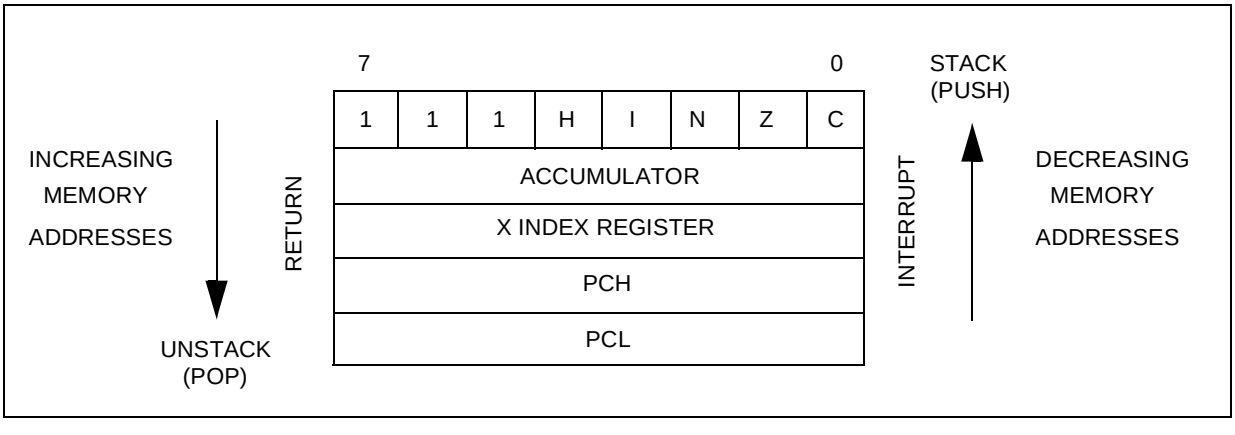
When set, C indicates that a carry or borrow out of the ALU occurred during the last arithmetic operation on the MSB operation result bit. This bit is also affected during bit test, branch, shift, rotate and load instructions. See ADD, ADC, SUB, SBC instructions. In bit test operations, C is the copy of the tested bit. See BTJF, BTJT instructions. In shift and rotates operations, the carry is updated. See RRC, RLC, SRL, SLL, SRA instructions

This bit can be set/reset by S/W

Example: Addition:$B5 + $94 = "C" + $49 = $149

	C	7						0
	0	1	0	1	1	0	1	0	1
+	0	1	0	0	1	0	1	0	0
=	1	0	1	0	0	1	0	0	1

Figure 2. Stacking Order



3 ST7 ADDRESSING MODES

The ST7 core features 17 different addressing modes which can be classified in 7 main groups:

Addressing Mode	Example
Inherent	nop
Immediate	ld A,#$55
Direct	ld A,$55
Indexed	ld A,($55,X)
Indirect	ld A,([$55],X)
Relative	jrne loop
Bit operation	bset byte,#5

The ST7 Instruction set is designed to minimize the number of required bytes per instruction: To do so, most of the addressing modes can be split in two sub-modes called long and short:

- The long addressing mode is the most powerful because it can reach any byte in the 64kb addressing space, but the instruction is bigger and slower than the short addressing mode.
- The short addressing mode is less powerful because it can generally only access the page zero (00..FF range), but the instruction size is more compact, and faster. All memory to memory instructions are only working with short addressing modes (CLR, CPL, NEG, BSET, BRES, BTJT, BTJF, INC, DEC, RLC, RRC, SLL, SRL, SRA, SWAP)

Both modes have pros and cons, but the programmer doesn't need to choose which one is the best: the ST7 Assembler will always choose the best one.

Table 1. ST7 Addressing Mode Overview:

Mode			Syntax	Destination	Ptr adr	Ptr size	Lgth
Inherent			nop				+ 0
Immediate			ld A,#$55				+ 1
Short	Direct		ld A,$10	00..FF			+ 1
Long	Direct		ld A,$1000	0000..FFFF			+ 2
No Offset	Direct	Indexed	ld A,(X)	00..FF			+ 0
Short	Direct	Indexed	ld A,($10,X)	00..1FE			+ 1
Long	Direct	Indexed	ld A,($1000,X)	0000..FFFF			+ 2
Short	Indirect		ld A,[$10]	00..FF	00..FF	byte	+ 2
Long	Indirect		ld A,[$10.w]	0000..FFFF	00..FF	word	+ 2
Short	Indirect	Indexed	ld A,([$10],X)	00..1FE	00..FF	byte	+ 2
Long	Indirect	Indexed	ld A,([$10.w],X)	0000..FFFF	00..FF	word	+ 2
Relative	Direct		jrne loop	PC+/-127			+ 1
Relative	Indirect		jrne [$10]	PC+/-127	00..FF	byte	+ 2
Bit	Direct		bset $10,#7	00..FF			+ 1
Bit	Indirect		bset [$10],#7	00..FF	00..FF	byte	+ 2
Bit	Direct	Relative	btjt $10,#7,skip	00..FF			+ 2
Bit	Indirect	Relative	btjt [$10],#7,skip	00..FF	00..FF	byte	+ 3

Inherent:

All related instructions are single byte ones. The op-code fully specify all required information for the CPU to process the operation. These instructions are single byte ones.

Example:

1000 98 rcf

1001 9D nop

Action: Do the operation:

Inherent Instruction	Function
NOP	No operation
TRAP	S/W Interrupt
WFI	Wait For Interrupt (Low Power Mode)
HALT	Halt Oscillator (Lowest Power Mode)
RET	Sub-routine Return
IRET	Interrupt Sub-routine Return
SIM	Set Interrupt Mask
RIM	Reset Interrupt Mask
SCF	Set Carry Flag
RCF	Reset Carry Flag
RSP	Reset Stack Pointer
LD	Load
CLR	Clear
PUSH/POP	Push/Pop to/from the stack
INC/DEC	Increment/Decrement
TNZ	Test Negative or Zero
CPL, NEG	1 or 2 Complement
MUL	Byte Multiplication
SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles

Immediate:

The required data byte to do the operation is following the op-code.

Immediate Instruction	Function
LD	Load
CP	Compare
BCP	Bit Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC	Arithmetic Operations

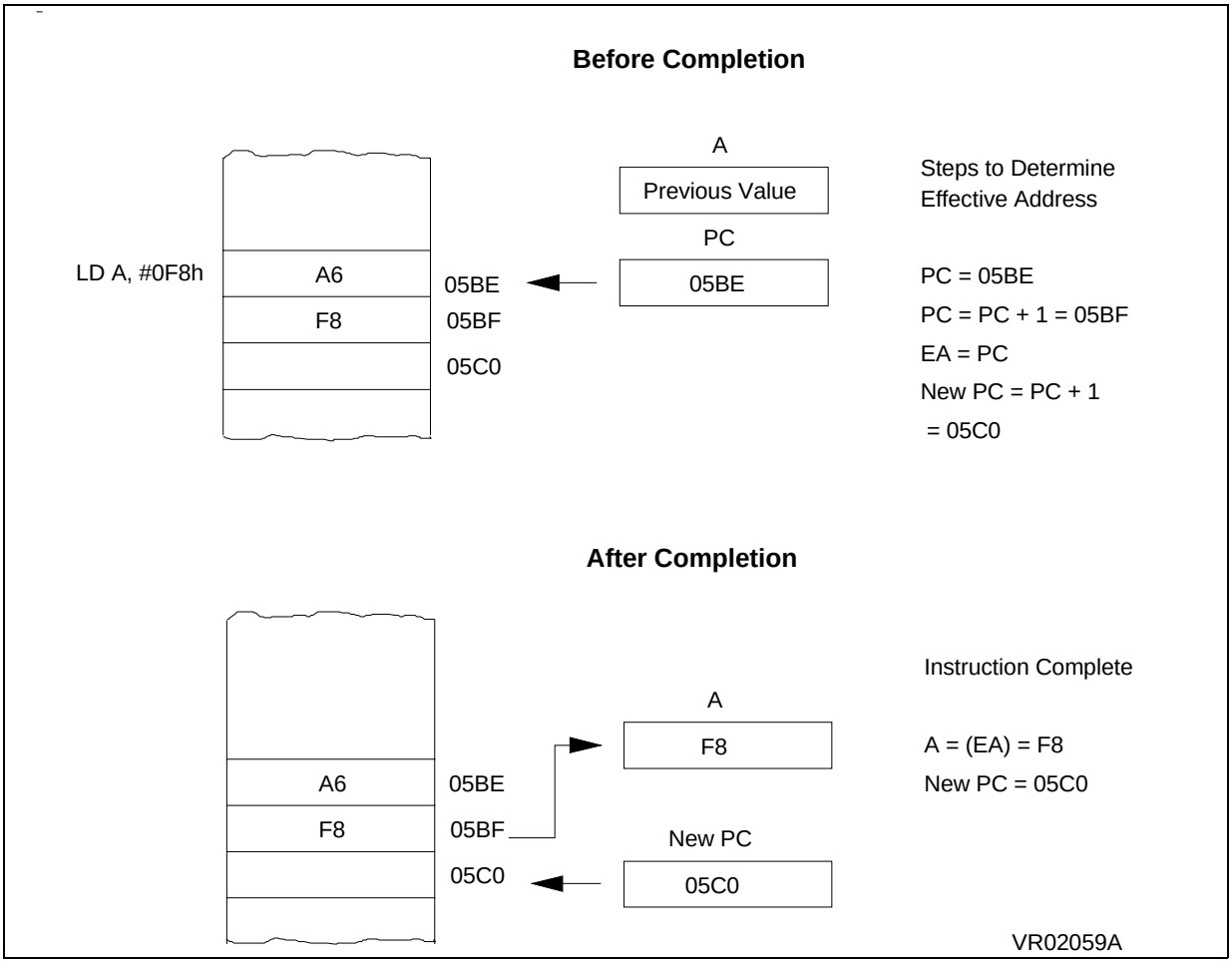
These are two byte instructions, one for the opcode and the other one for the immediate data byte.

Example:

```
1000    AEFF    ld    X,#$FF
1002    A355    cp    X,#$55
1004    A6F8    ld    A,#$F8
```

Action: X = $FF
Compare (X, $55)
A = $F8

Figure 3. Immediate Addressing Mode Example



Direct (short, long):

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Short	Direct		(ptr)	(ptr)	op + 1	Byte	00..FF
Long	Direct		(ptr)	(ptr.w)	op + 1..2	Word	0000..FFFF

The required data byte to do the operation is found by its memory address, which follows the op-code. The direct addressing mode is made of two sub-modes:

Available Long and Short Direct Instructions	Function
LD	Load
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC	Arithmetic Additions/Substractions operations
BCP	Bit Compare

Short Direct Instructions Only	Function
CLR	Clear
INC, DEC	Increment/Decrement
TNZ	Test Negative or Zero
CPL, NEG	1 or 2 Complement
BSET, BRES	Bit Operations
BTJT, BTJF	Bit Test and Jump Operations
SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles
CALL, JP	Call or Jump subroutine

Short direct:

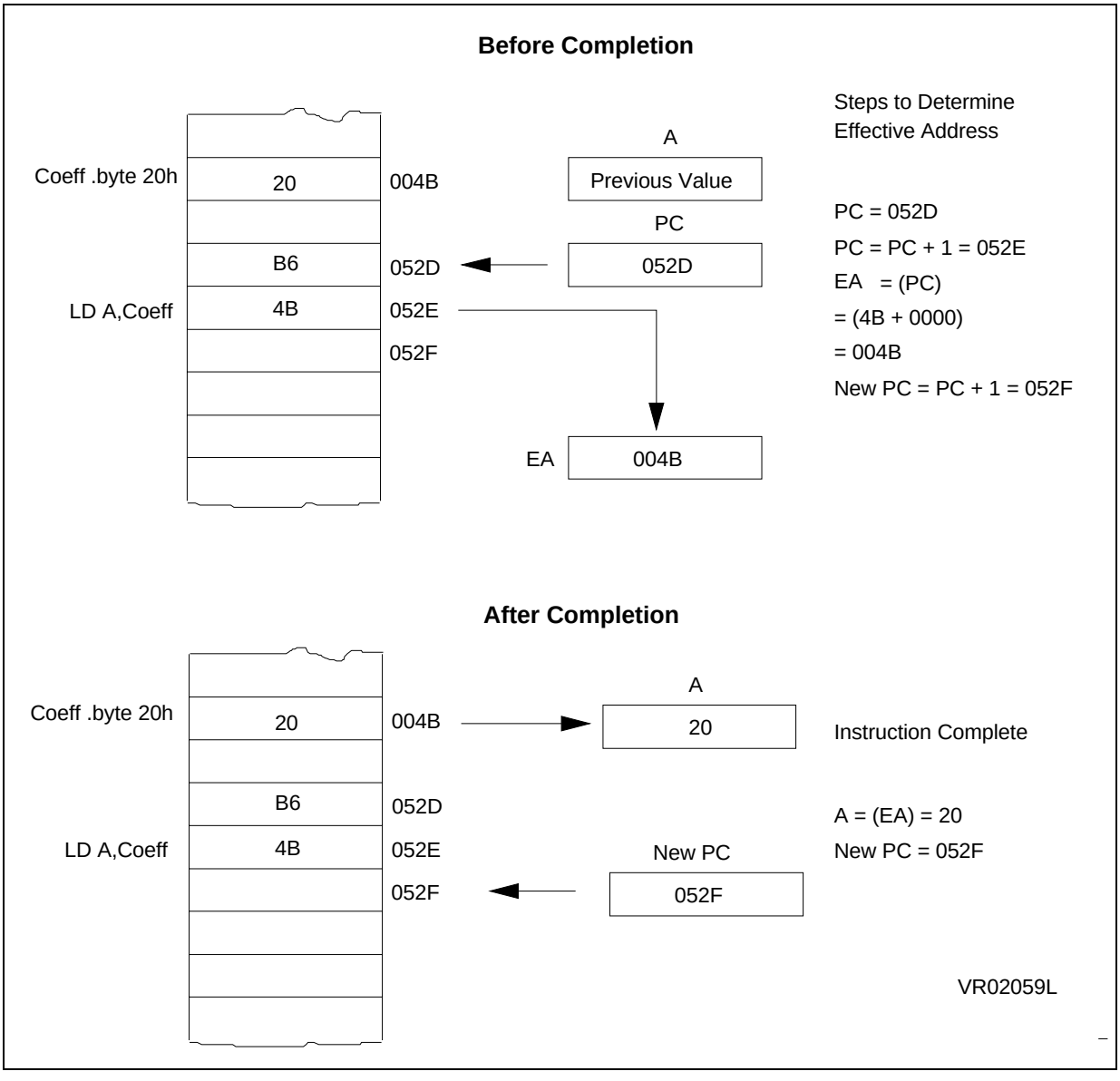
The address is a byte, thus require only one byte after the op-code, but only allow 00..FF addressing space.

Example:

004B	20	coeff	dc.b$20
052D	B64B		ld A,coeff

Action: A = (coeff) = ($4B) = $20

Figure 4. Short Direct Addressing Mode Example



Long direct:

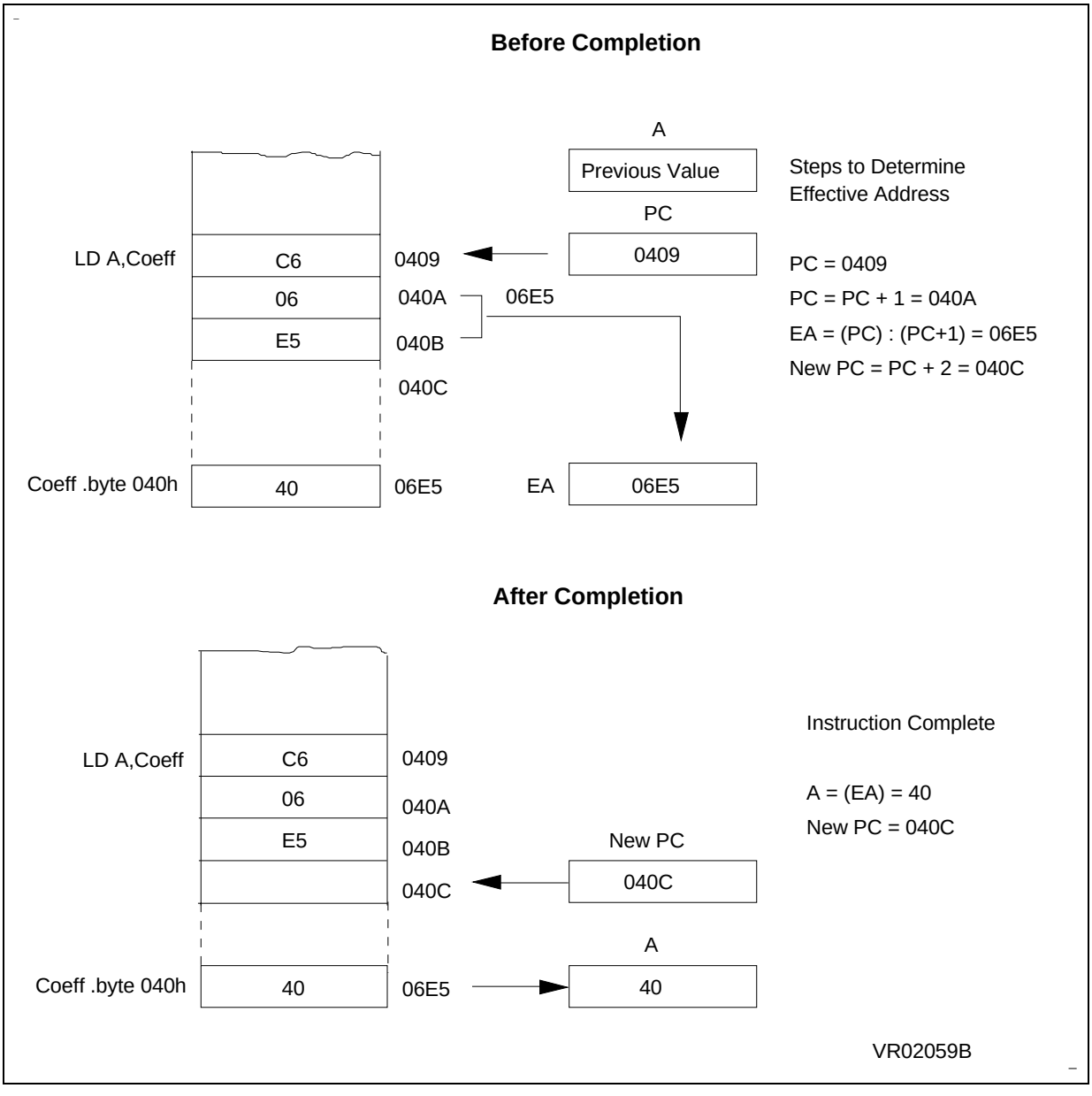
The address is a word, thus allowing 64 kb addressing space, but requires 2 bytes after the op-code.

Example:

```
0409        C606E5              ld   A,coeff
06E5        40          coeff   dc.b$ 40
```

Action: A = (coeff) = ($06E5) = $40

Figure 5. Long Direct Addressing Mode Example



Indexed (no offset, short, long)

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
No offset	Direct	Indexed	(ndx)	(ndx)	---	---	00..FF
Short	Direct	Indexed	(ptr,ndx)	(ptr + ndx)	op + 1	Byte	00..1FE
Long	Direct	Indexed	(ptr.w,ndx)	(ptr.w + ndx)	op + 1..2	Word	0000..FFFF

The required data byte to do the operation is found by its memory address, which is defined by the unsigned addition of an index register (X or Y) with an offset which follows the op-code.

The indexed addressing mode is made of three sub-modes:

No Offset, Long and Short Indexed Instructions	Function
LD	Load
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC	Arithmetic Additions/Substractions operations
BCP	Bit Compare

No Offset and Short Indexed Instructions Only	Function
CLR	Clear
INC, DEC	Increment/Decrement
TNZ	Test Negative or Zero
CPL, NEG	1 or 2 Complement
BSET, BRES	Bit Operations
BTJT, BTJF	Bit Test and Jump Operations
SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles
CALL, JP	Call or Jump subroutine

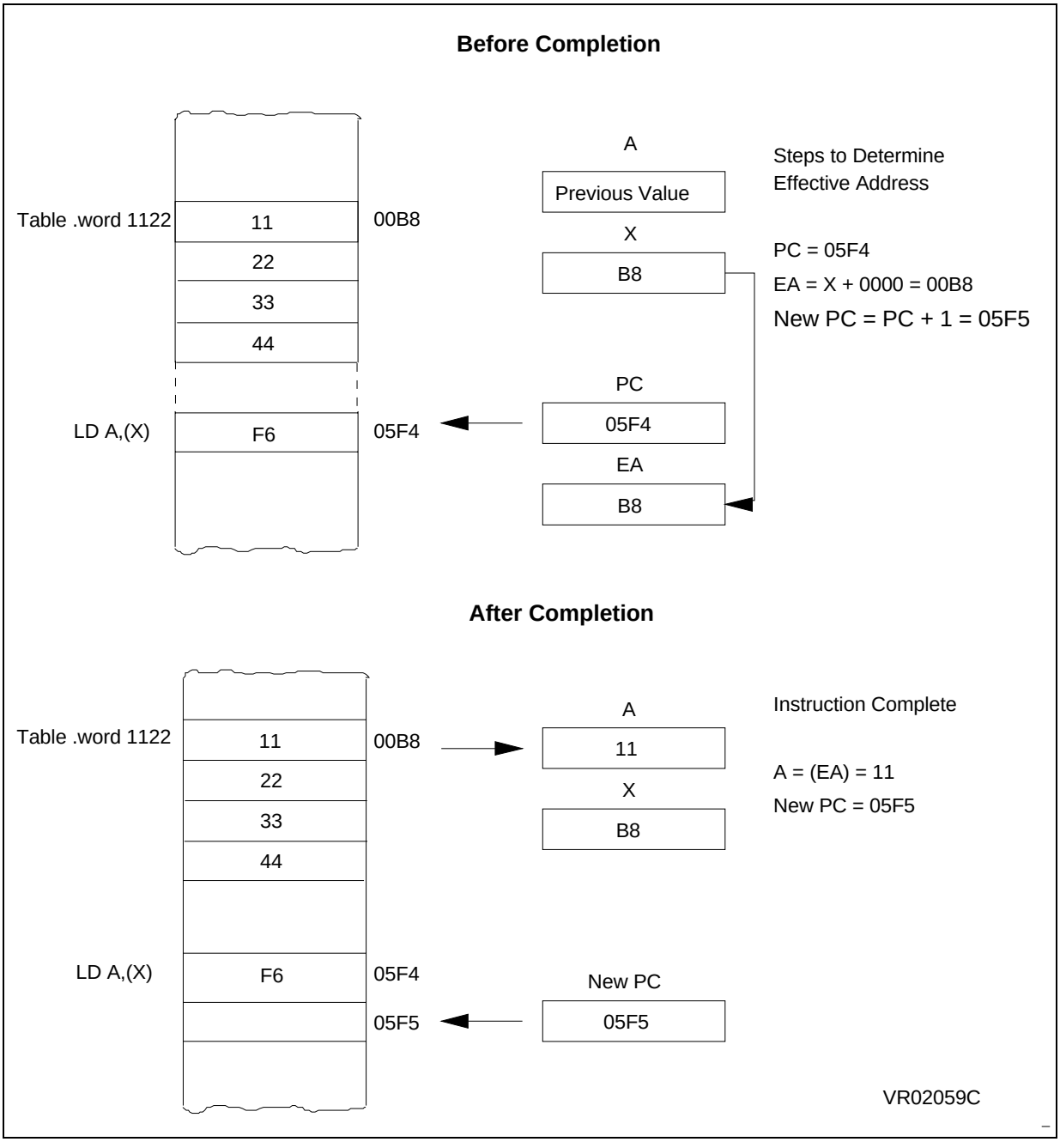
(no offset) Indexed:

There is no offset, (no extra byte after the op-code), but only allows 00..FF addressing space.

Example:

```
**00**B8    11223344    table   dc.w$1122, $3344
05F2        AEB8                ld   X,#table
05F4        F6                  ld   A,(X)
Action:     X = table
            A = (X) = (table) = ($B8) = $11
```

Figure 6. No offset Indexed Addressing Mode Example



Short Indexed:

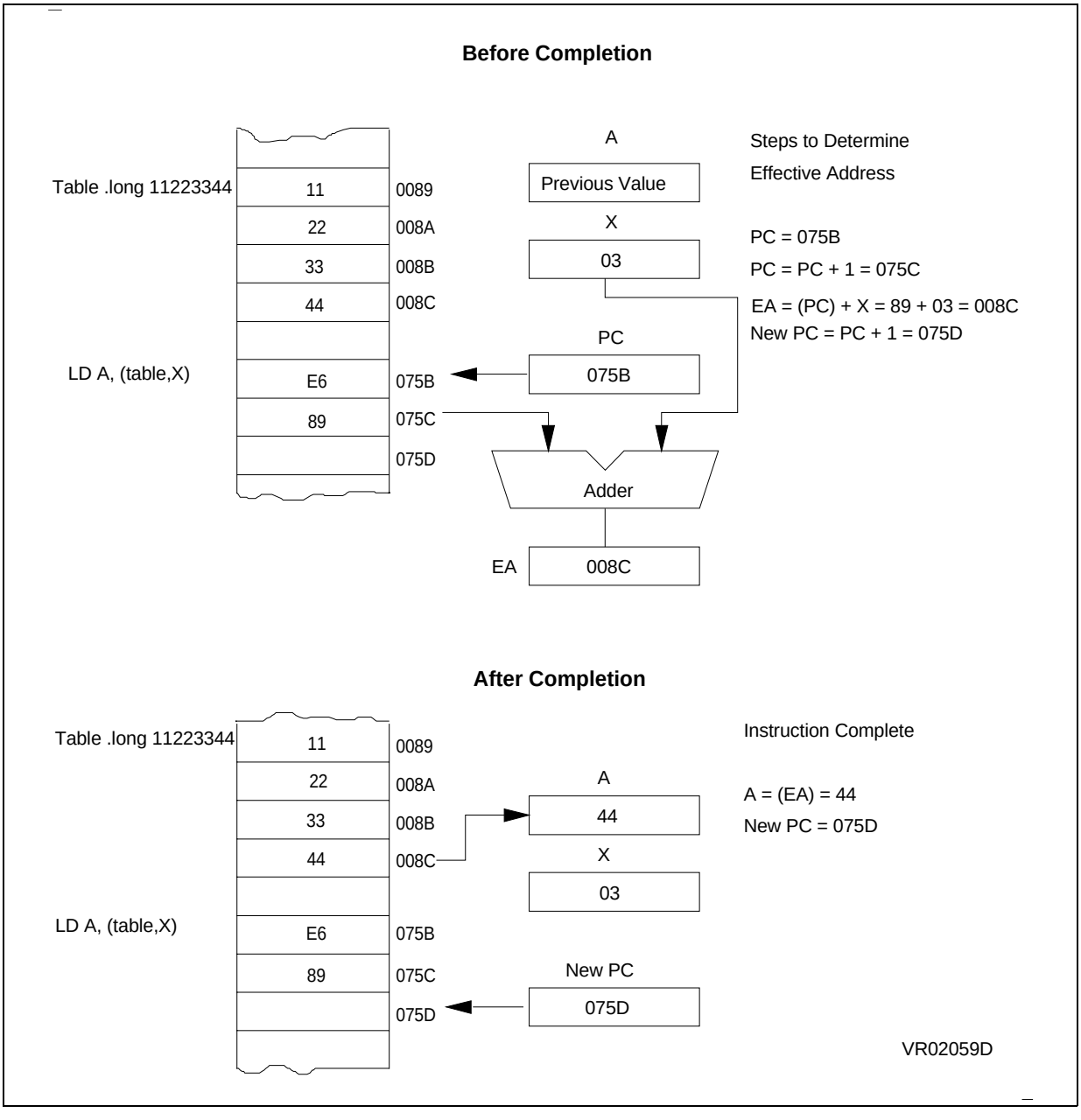
The offset is a byte, thus require only one byte after the op-code, but only allow 00..1FE addressing space.

Example:

```
0089        11223344      table   dc.l $11223344
0759        AE03                  ld   X,#3
075B        E689                  ld   A,(table,X)
```

Action: X = 3
A = (table, X) = ($89, X) = ($89, 3) = ($8C) = $44

Figure 7. Short Indexed - 8-bit offset - Addressing Mode Example



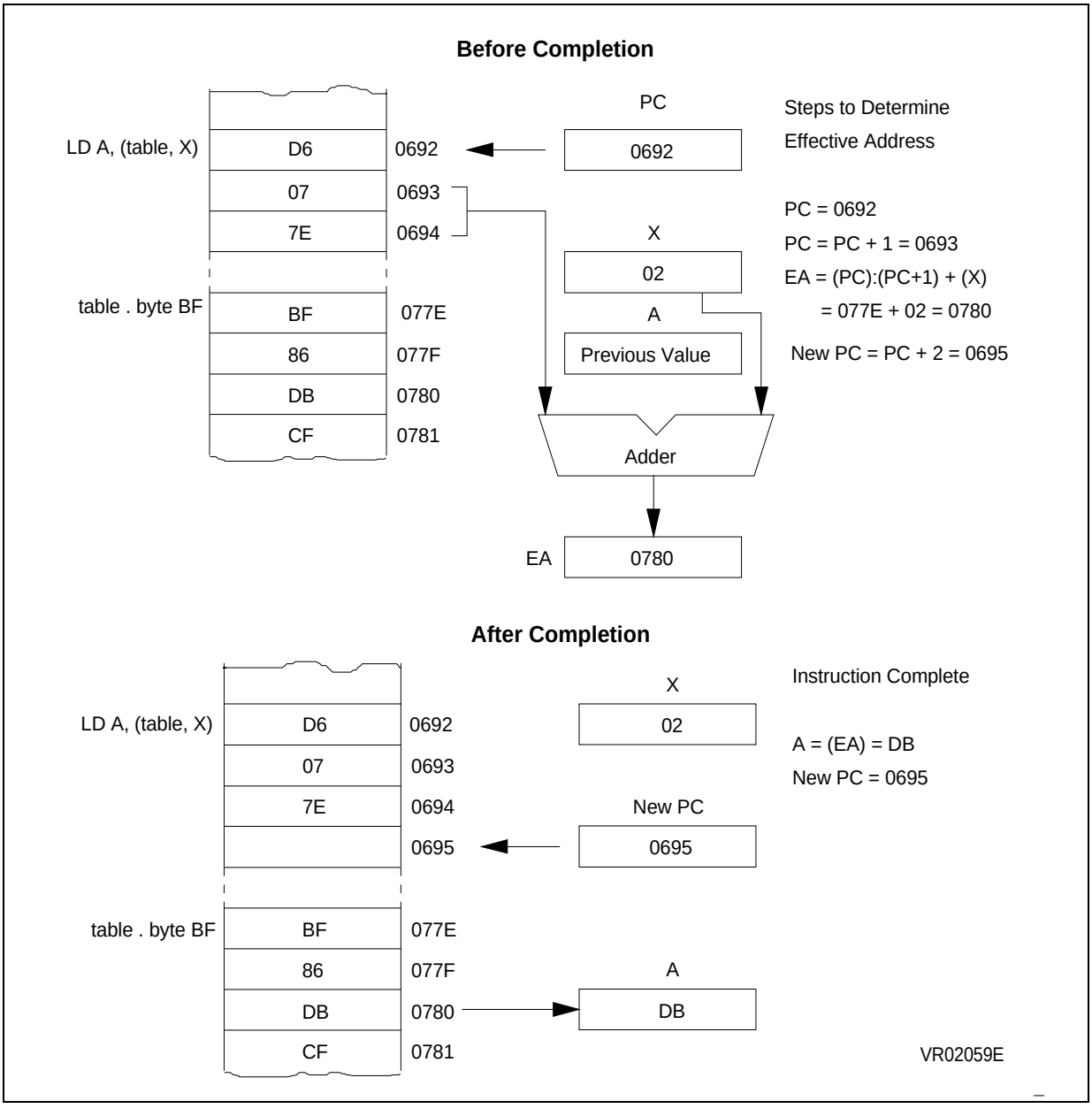
Long Indexed:

The offset is a word, thus allowing 64 kb addressing space, but requires 2 bytes after the op-code.

Example:

```
0690        AE02        ld      X,#2
0692        D6077E      ld      A,(table,X)

077E        BF          table   dc.b$BF
            86                  dc.b$86
            DBCF                dc.w$DBCF
Action:     X = 2
            A = (table, X) = ($077E, X) = ($077E, 2) = ($0780) = $DB
```

Figure 8. Long Indexed - 16-bit offset - Addressing Mode Example



Indirect (short, long):

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Short	Indirect		((ptr))	((ptr))	00..FF	Byte	00..FF
Long	Indirect		((ptr.w))	((ptr.w))	00..FF	Word	0000..FFFF

The required data byte to do the operation is found by its memory address, located in memory (pointer). The pointer address follows the op-code. The indirect addressing mode is made of two sub-modes:

Available Long and Short Indirect Instructions	Function
LD	Load
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC	Arithmetic Additions/Substractions operations
BCP	Bit Compare

Short Indirect Instructions Only	Function
CLR	Clear
INC, DEC	Increment/Decrement
TNZ	Test Negative or Zero
CPL, NEG	1 or 2 Complement
BSET, BRES	Bit Operations
BTJT, BTJF	Bit Test and Jump Operations
SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles
CALL, JP	Call or Jump subroutine

Short Indirect:

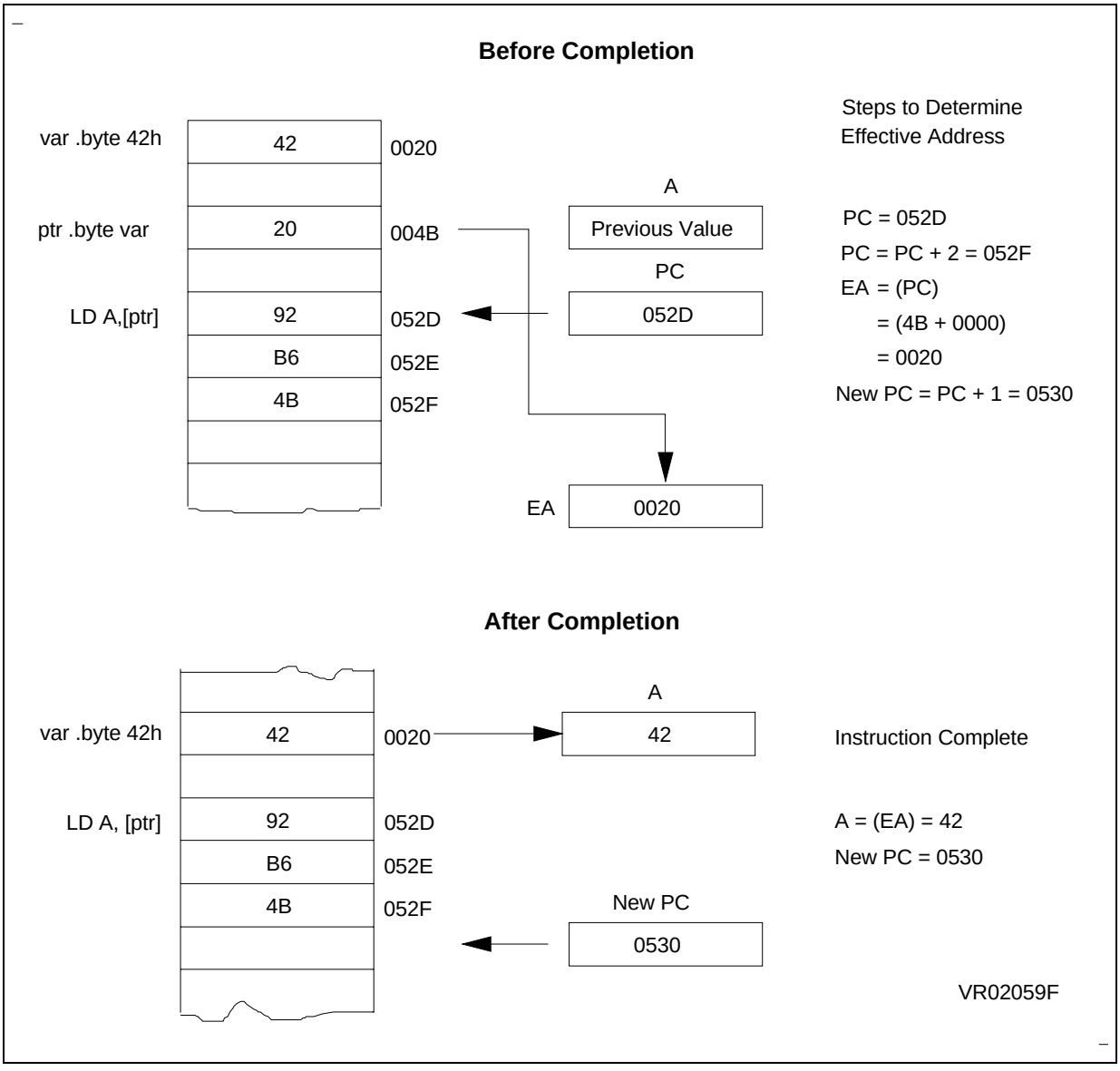
The pointer address is a byte, the pointer size is a byte, thus allowing 00..FF addressing space, and requires 1 byte after the op-code.

Example:

0020	42	var	dc.b$42
004B	**20**	ptr	dc.bvar
052D	92B64B		ld A,[ptr]

Action: A = [ptr] = ((ptr)) = (($4B)) = ($20) = $42

Figure 9. Short Indirect Addressing Mode Example



Long Indirect:

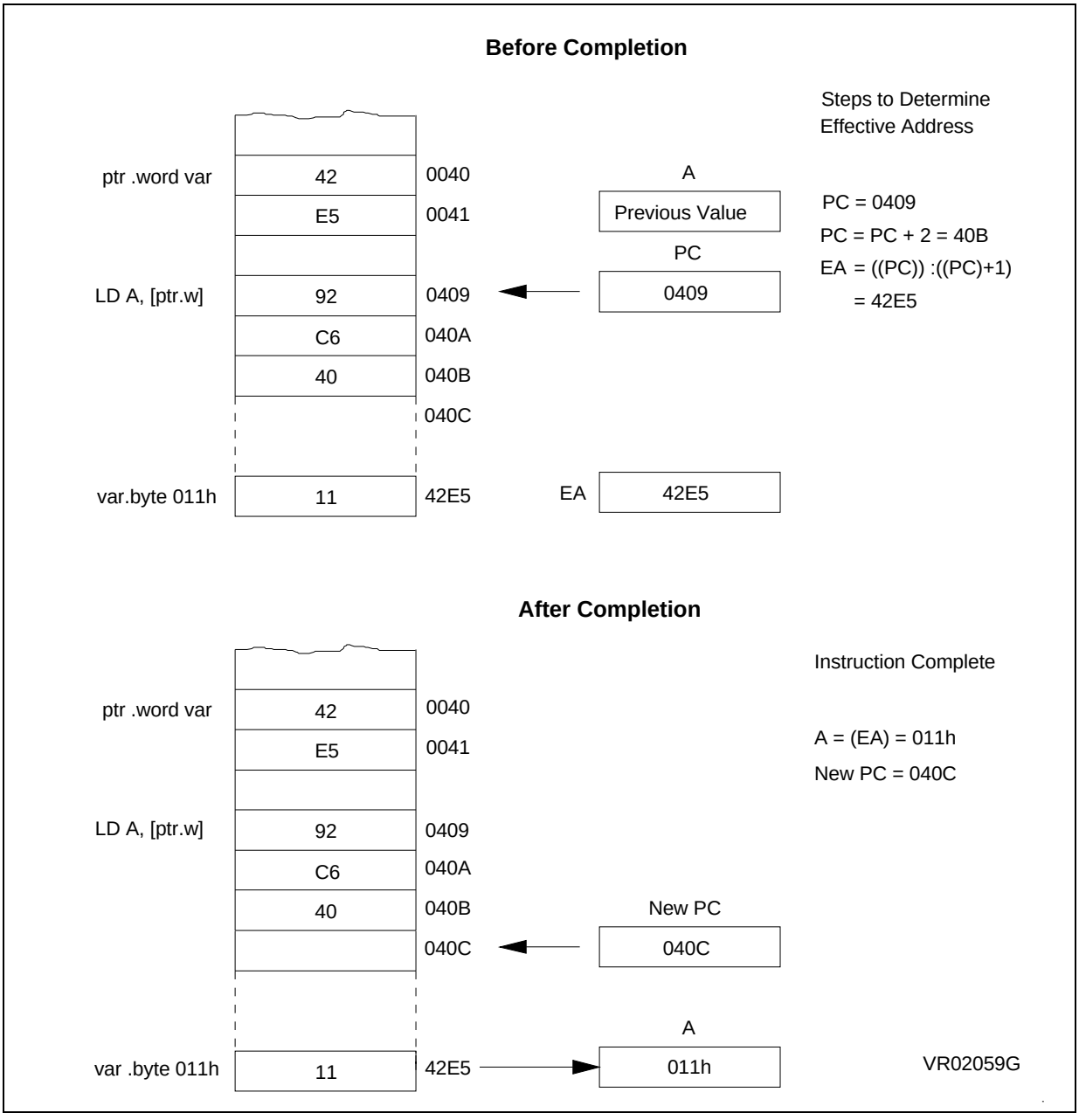
The pointer address is a byte, the pointer size is a word, thus allowing 64 kb addressing space, and requires 1 byte after the op-code.

Example:

```
00 40    42E5    ptr    dc.w var
0409     92C640         ld   A,[ptr.w]
42E5     11      var    dc.b $11
```

Action: A = [ptr.w] = ((ptr.w)) = (($40.w)) = ($**42E5**) = $11

Figure 10. Long Indirect Addressing Mode Example



Indirect indexed (short, long):

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Short	Indirect	Indexed	([ptr],ndx)	((ptr) + ndx)	00..FF	Byte	00..1FE
Long	Indirect	Indexed	([ptr.w],ndx)	((ptr.w) + ndx)	00..FF	Word	0000..FFFF

This is a combination of indirect and short indexed addressing mode. The required data byte to do the operation is found by its memory address, which is defined by the unsigned addition of an index register value (X or Y) with a pointer value located in memory. The pointer address follows the op-code.

The indirect indexed addressing mode is made of two sub-modes:

Long and Short Indirect Indexed Instructions	Function
LD	Load
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC	Arithmetic Additions/Substractions operations
BCP	Bit Compare

Short Indirect Indexed Instructions Only	Function
CLR	Clear
INC, DEC	Increment/Decrement
TNZ	Test Negative or Zero
CPL, NEG	1 or 2 Complement
BSET, BRES	Bit Operations
BTJT, BTJF	Bit Test and Jump Operations
SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles
CALL, JP	Call or Jump subroutine

Short indirect indexed:

The pointer address is a byte, the pointer size is a byte, thus allowing 00..1FE addressing space, and requires 1 byte after the op-code.

Example:

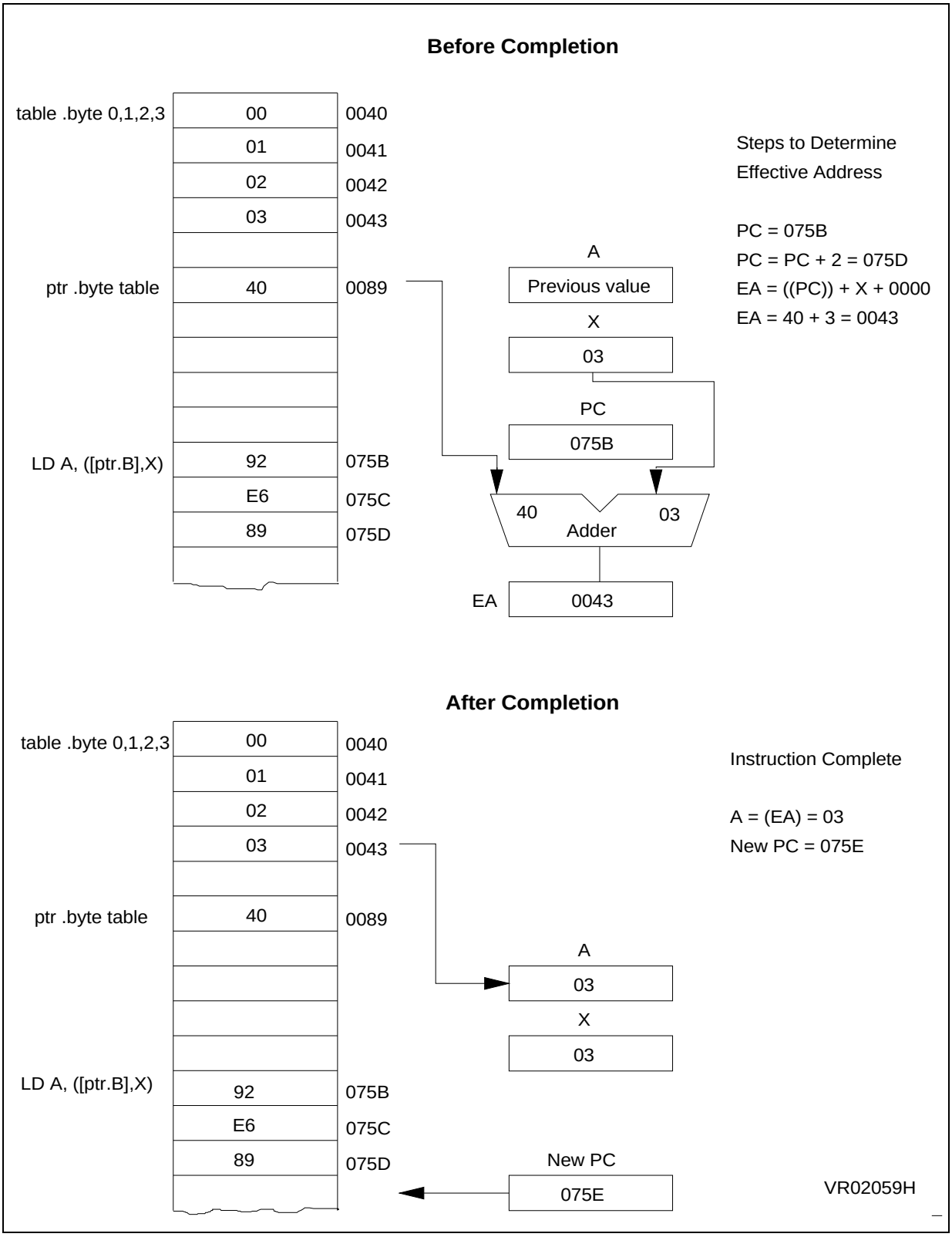
```
0040    00          table   dc.b0,1,2,3
0041    01
0042    02
0043    03

0089    40          ptr     dc.btable

0759    AE03                ld   X,#3
075B    92E689              ld   A,([ptr],X)
```

Action:

X = 3

A = ([ptr],X) = ((ptr) , X) = (($89), 3) = ($40, 3) = ($43) = 3

Figure 11. Short Indirect Indexed Addressing Mode Example



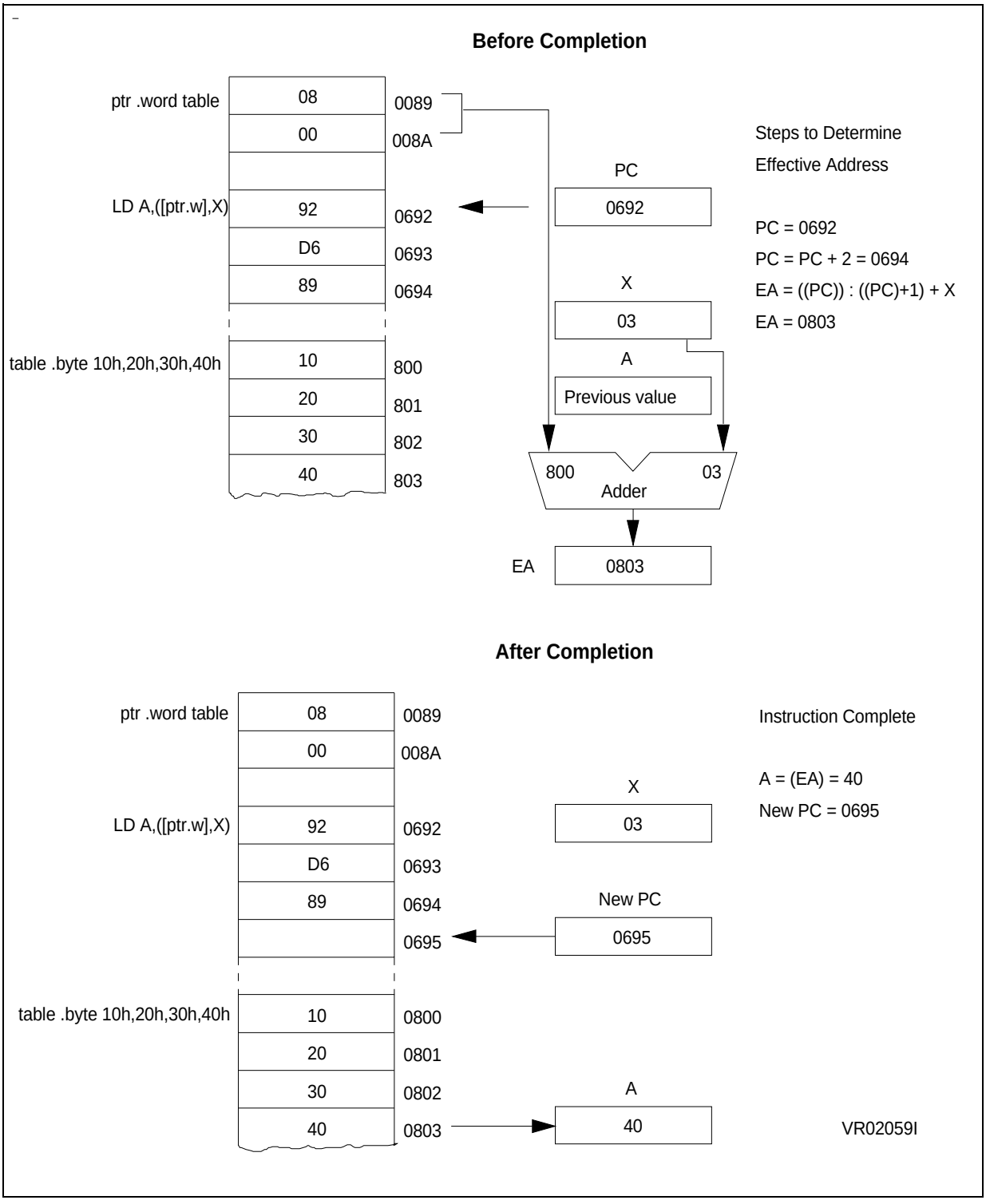
Long indirect indexed:

The pointer address is a byte, the pointer size is a word, thus allowing 64 kb addressing space, and requires 1 byte after the op-code.

Example:

```
0089    0800        ptr     dc.wtable

0800    10203040    table   dc.b$10,$20,$30,$40

0690    AE03                ld   X,#3
0692    92D689              ld   A,([ptr.w],X)
                            X = 3
                            A = ([ptr.w],X) = ((ptr.w), X) = (($89.w), 3)
                            = ($0800,3) = ($0803) = $40
```

Figure 12. Long Indirect Indexed Addressing Mode Example



Relative mode (direct, indirect):

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Direct	Relative		oft	PC = PC + oft	op + 1	---	PC +/- 127
Indirect	Relative		[oft]	PC = PC + (oft)	00..FF	Byte	PC +/- 127

This addressing mode is used to modify the PC register value, by adding an 8-bit signed offset to it. The relative addressing mode is made of two sub-modes:

Available Relative Direct/Indirect Instructions	Function
JRxx	Conditional Jump
CALLR	Call Relative

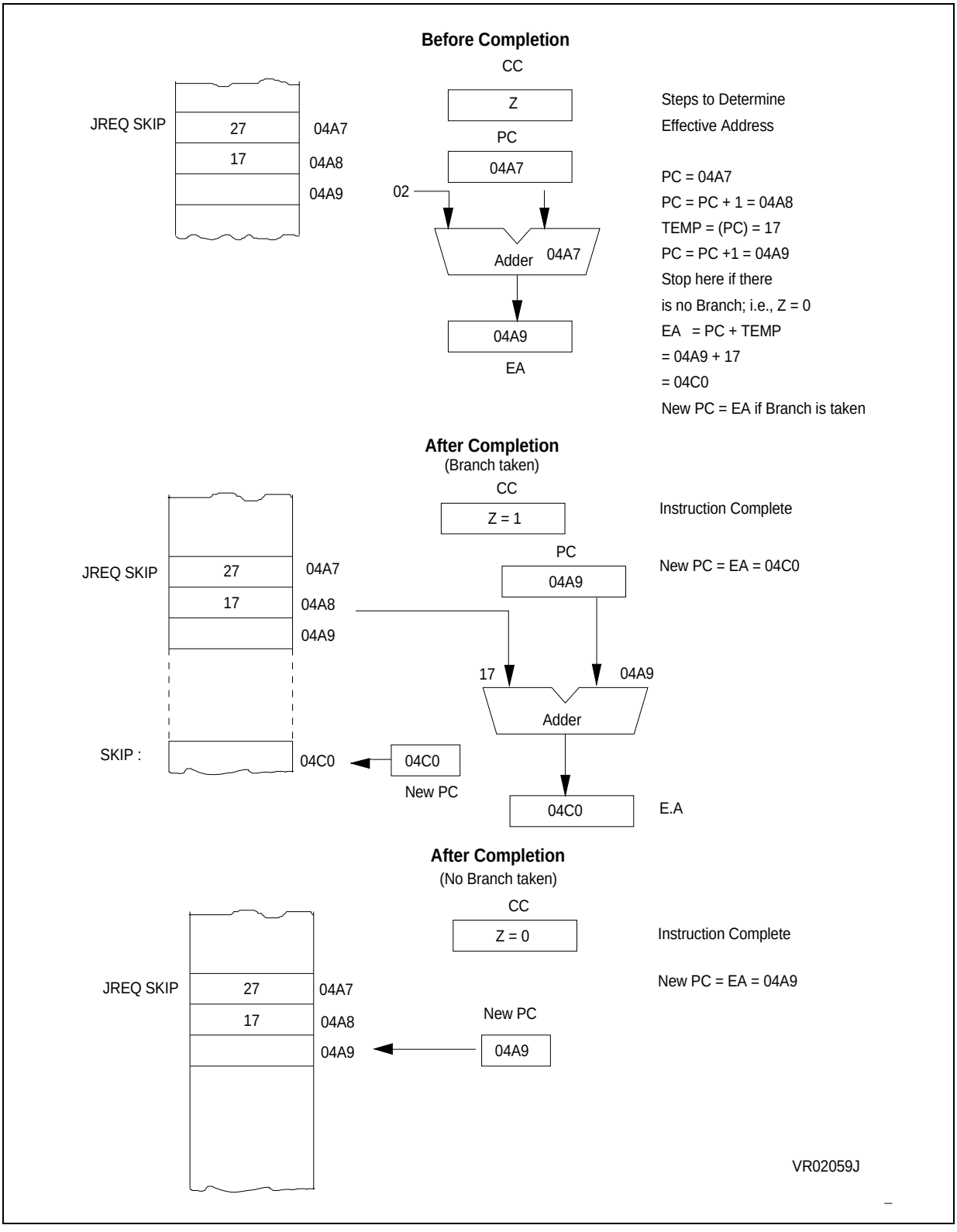
Relative (Direct):

The offset is following the op-code.

Example:

```
04A7    2717            jreq skip
04A9    9D              nop
04AA    9D              nop

04C0    20FE    skip    jra  *              ; Infinite loop

Action: if (Z == 1)     then PC = PC + $17 = $04A9 + $17    = $04C0
                        else PC = PC       = $04A9
```

Figure 13. Relative Direct Indexed Addressing Mode Example



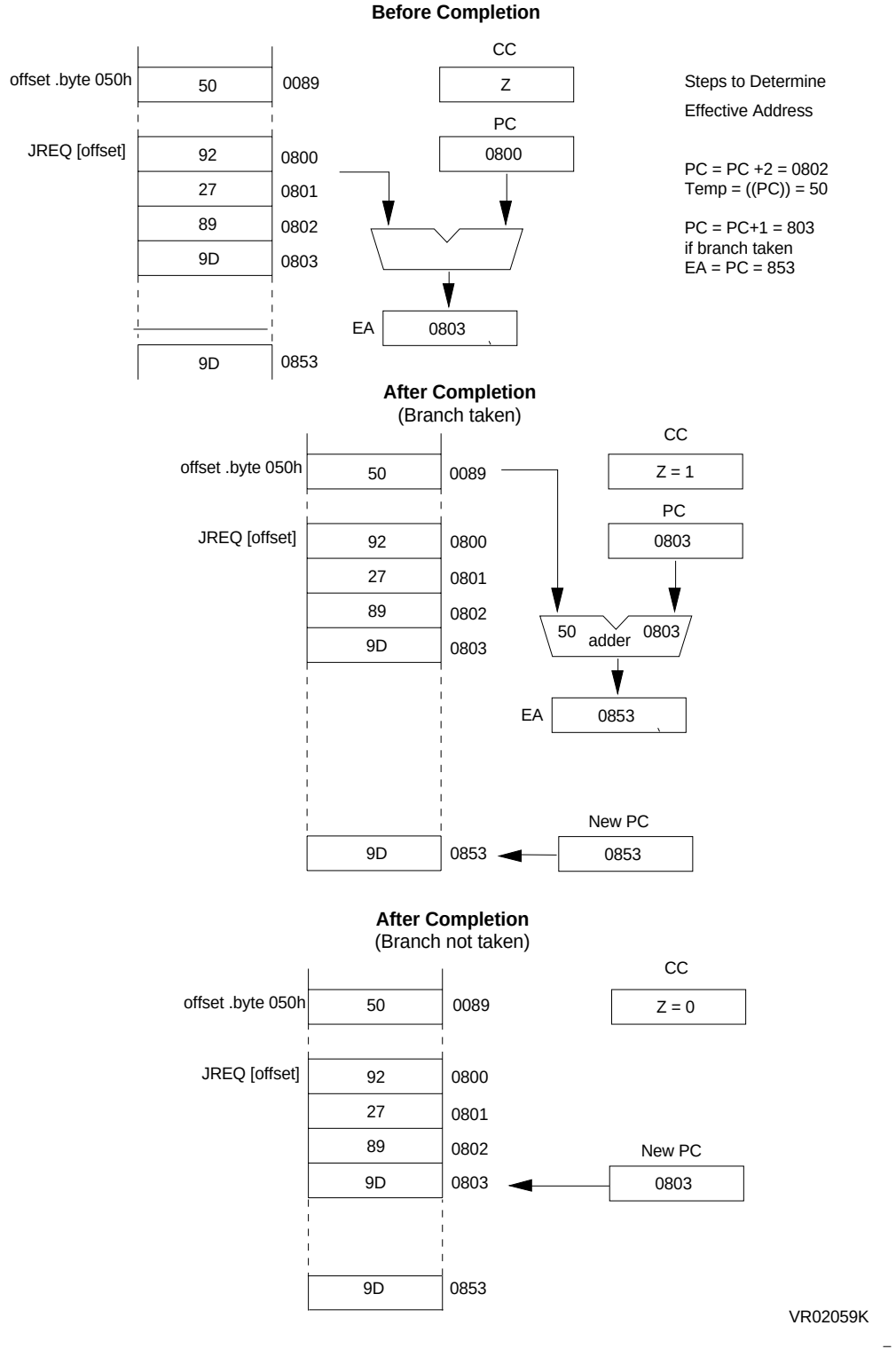
Relative Indirect:

The offset is defined in memory, which address follows the op-code.

Example:

```
0089    50          offset  dc.b$50
0800    922789              jreg[offset]
0803    9D                  nop
0853    9D                  nop
```

Relative Indirect Indexed Addressing Mode Example


Before Completion
offset .byte 050h
50
0089
JREQ [offset]
92
0800
27
0801
89
0802
9D
0803
9D
0853
CC
Z
PC
0800
EA
0803
Steps to Determine
Effective Address
PC = PC +2 = 0802
Temp = ((PC)) = 50
PC = PC+1 = 803
if branch taken
EA = PC = 853
After Completion
(Branch taken)
offset .byte 050h
50
0089
JREQ [offset]
92
0800
27
0801
89
0802
9D
0803
9D
0853
CC
Z = 1
PC
0803
50
adder
0803
EA
0853
New PC
0853
After Completion
(Branch not taken)
offset .byte 050h
50
0089
JREQ [offset]
92
0800
27
0801
89
0802
9D
0803
9D
0853
CC
Z = 0
New PC
0803
VR02059K

4 ST7 INSTRUCTION SET

4.1 INTRODUCTION

This chapter describes all the ST7 instructions. They are 63 and are described in alphabetical order. However, they can be classified in 13 main groups as follows:

Load and Transfer	LD	CLR						
Stack operation	PUSH	POP	RSP					
Increment/Decrement	INC	DEC						
Compare and Tests	CP	TNZ	BCP					
Logical operations	AND	OR	XOR	CPL	NEG			
Bit Operation	BSET	BRES						
Conditional Bit Test and Branch	BTJT	BTJF						
Arithmetic operations	ADC	ADD	SUB	SBC	MUL			
Shift and Rotates	SLL	SRL	SRA	RLC	RRC	SWAP	SLA	
Unconditional Jump or Call	JRA	JRT	JRF	JP	CALL	CALLR	NOP	RET
Conditional Branch	JRxx							
Interruption management	TRAP	WFI	HALT	IRET				
Code Condition Flag modification	SIM	RIM	SCF	RCF				

Using a pre-byte

The instructions are described with one to four bytes.

In order to extend the number of available opcodes for an 8-bit CPU (256 op-codes), three different prebyte opcodes are defined. These prebytes modify the meaning of the instruction they precede.

The whole instruction becomes:

PC-2 End of previous instruction

PC-1 Prebyte

PC Op-code

PC+1 Additional word (0 to 2) according to the number of bytes required to compute the effective address

These prebytes enable instruction in Y as well as indirect addressing modes to be implemented. They precede the opcode of the instruction in X or the instruction using direct addressing mode. The prebytes are:

PDY 90 Replace an X based instruction using immediate, direct, indexed or inherent addressing mode by a Y one.

PIX 92 Replace an instruction using direct, direct bit, or direct relative addressing mode to an instruction using the corresponding indirect addressing mode.
It also changes an instruction using X indexed addressing mode to an instruction using indirect X indexed addressing mode.

PIY 91 Replace an instruction using indirect X indexed addressing mode by a Y one.

4.2 INSTRUCTION SET SUMMARY

Mnemo	Description	Function/Example	Dst	Src	H	I	N	Z	C
ADC	Add with Carry	A = A + Mem + C	A	Mem	H		N	Z	C
ADD	Addition	A = A + Mem	A	Mem	H		N	Z	C
AND	Logical And	A = A . Mem	A	Mem			N	Z	
BCP	Bit compare A, Memory	tst (A . Mem)	A	Mem			N	Z	
BRES	Bit Reset	bres Byte, #3	Mem						
BSET	Bit Set	bset Byte, #3	Mem						
BTJF	Jump if bit is false (0)	btjf Byte, #3, Jmp1	Mem						C
BTJT	Jump if bit is true (1)	btjt Byte, #3, Jmp1	Mem						C
CALL	Call subroutine								
CALLR	Call subroutine relative								
CLR	Clear		reg, Mem				0	1	
CP	Arithmetic Compare	tst(Reg - Mem)	reg	Mem			N	Z	C
CPL	One Complement	A = FFH-A	reg, Mem				N	Z	1
DEC	Decrement	dec Y	reg, Mem				N	Z	
HALT	Halt					0			
IRET	Interrupt routine return	Pop CC, A, X, PC			H	I	N	Z	C
INC	Increment	inc X	reg, Mem				N	Z	
JP	Absolute Jump	jp [TBL.w]							
JRA	Jump relative always								
JRT	Jump relative								
JRF	Never jump	jrf *							
JRIH	Jump if Port INT pin = 1	(no Port Interrupts)							
JRIL	Jum if Port INT pin = 0	(Port interrupt)							
JRH	Jump if H = 1	H = 1 ?							
JRNH	Jump if H = 0	H = 0 ?							
JRM	Jump if I = 1	I = 1 ?							
JRNM	Jump if I = 0	I = 0 ?							
JRMI	Jump if N = 1 (minus)	N = 1 ?							
JRPL	Jump if N = 0 (plus)	N = 0 ?							
JREQ	Jump if Z = 1 (equal)	Z = 1 ?							
JRNE	Jump if Z = 0 (not equal)	Z = 0 ?							
JRC	Jump if C = 1	C = 1 ?							
JRNC	Jump if C = 0	C = 0 ?							
JRULT	Jump if C = 1	Unsigned <							
JRUGE	Jump if C = 0	Jmp if unsigned >=							
JRUGT	Jump if (C + Z = 0)	Unsigned >							
JRULE	Jump if (C + Z = 1)	Unsigned <=							

Mnemo	Description	Function/Example	Dst	Src	H	I	N	Z	C
LD	Load	dst <= src	reg, Mem	Mem, reg			N	Z	
MUL	Multiply	X,A = X * A	A, X, Y	X, Y, A	0				0
NEG	Negate (2's compl)	neg $10	reg, Mem				N	Z	C
NOP	No Operation								
OR	OR operation	A = A + Mem	A	Mem			N	Z	
POP	Pop from the Stack	pop reg pop CC	reg CC	Mem Mem	H	I	N	Z	C
PUSH	Push onto the Stack	push Y	Mem	reg, CC					
RCF	Reset carry flag	C = 0							0
RET	Subroutine Return								
RIM	Enable Interrupts	I = 0				0			
RLC	Rotate left true C	C <= A <= C	reg, Mem				N	Z	C
RRC	Rotate right true C	C => A => C	reg, Mem				N	Z	C
RSP	Reset Stack Pointer	S = Max allowed							
SBC	Subtract with Carry	A = A - Mem - C	A	Mem			N	Z	C
SCF	Set carry flag	C = 1							1
SIM	Disable Interrupts	I = 1				1			
SLA	Shift left Arithmetic	C <= A <= 0	reg, Mem				N	Z	C
SLL	Shift left Logic	C <= A <= 0	reg, Mem				N	Z	C
SRL	Shift right Logic	0 => A => C	reg, Mem				0	Z	C
SRA	Shift right Arithmetic	A7 => A => C	reg, Mem				N	Z	C
SUB	Substraction	A = A - Mem	A	Mem			N	Z	C
SWAP	SWAP nibbles	A7-A4 <=> A3-A0	reg, Mem				N	Z	
TNZ	Test for Neg & Zero	tnz lbl1					N	Z	
TRAP	S/W trap	S/W interrupt				1			
WFI	Wait for Interrupt					0			
XOR	Exclusive OR	A = A XOR Mem	A	M			N	Z	

ADC

Addition with Carry

ADC

Syntax adc dst,src e.g.: adc A,#$15

Operation dst <= dst + src + C

Description The source byte, along with the carry flag, is added to the destination byte and the result is stored in the destination byte. The source is a memory byte, and the destination is the A register.

Instruction Overview:

mnem	dst	src
ADC	A	Mem

Condition Flags

H	I	N	Z	C
H		N	Z	C

Detailed Description:

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A9	XX	
A	short	3	2		B9	XX	
A	long	4	3		C9	MS	LS
A	(X)	3	1		F9		
A	(short,X)	4	2		E9	XX	
A	(long,X)	5	3		D9	MS	LS
A	(Y)	4	2	90	F9		
A	(short,Y)	5	3	90	E9	XX	
A	(long,Y)	6	4	90	D9	MS	LS
A	[short]	5	3	92	B9	XX	
A	[long.w]	6	3	92	C9	XX	
A	([short],X)	6	3	92	E9	XX	
A	([long.w],X)	7	3	92	D9	XX	
A	([short],Y)	6	3	91	E9	XX	
A	([long.w],Y)	7	3	91	D9	XX	

See Also: ADD,SUB,SBC,MUL

ADD

Addition

ADD

Syntax add dst,src e.g.: add A,#%11001010

Operation dst <= dst + src

Description The source byte is added to the destination byte and the result is stored in the destination byte. The source is a memory byte, and the destination is the A register.

Instruction Overview

mnem	dst	src
ADD	A	Mem

Condition Flags

H	I	N	Z	C
H		N	Z	C

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		AB	XX	
A	short	3	2		BB	XX	
A	long	4	3		CB	MS	LS
A	(X)	3	1		FB		
A	(short,X)	4	2		EB	XX	
A	(long,X)	5	3		DB	MS	LS
A	(Y)	4	2	90	FB		
A	(short,Y)	5	3	90	EB	XX	
A	(long,Y)	6	4	90	DB	MS	LS
A	[short]	5	3	92	BB	XX	
A	[long.w]	6	3	92	CB	XX	
A	([short],X)	6	3	92	EB	XX	
A	([long.w],X)	7	3	92	DB	XX	
A	([short],Y)	6	3	91	EB	XX	
A	([long.w],Y)	7	3	91	DB	XX	

See Also:ADC, SUB, SBC, MUL

AND

Logical

AND

Syntax and dst,src e.g.: and A,#%00110101

Operation dst <= dst AND src

Description The source byte, is ANDed with the destination byte and the result is stored in the destination byte. The source is a memory byte, and the destination is the A register.

Truth Table:

AND	0	1
0	0	0
1	0	1

Instruction Overview

mnem	dst	src
AND	A	Mem

Condition Flags

H	I	N	Z	C
		N	Z	

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A4	XX	
A	short	3	2		B4	XX	
A	long	4	3		C4	MS	LS
A	(X)	3	1		F4		
A	(short,X)	4	2		E4	XX	
A	(long,X)	5	3		D4	MS	LS
A	(Y)	4	2	90	F4		
A	(short,Y)	5	3	90	E4	XX	
A	(long,Y)	6	4	90	D4	MS	LS
A	[short]	5	3	92	B4	XX	
A	[long.w]	6	3	92	C4	XX	
A	([short],X)	6	3	92	E4	XX	
A	([long.w],X)	7	3	92	D4	XX	
A	([short],Y)	6	3	91	E4	XX	
A	([long.w],Y)	7	3	91	D4	XX	

See Also: OR, XOR, CPL, NEG

BCP

Logical Bit Compare

BCP

Syntax bcp src,dst e.g.: bcp A,#%10100101

Operation {N, Z} <= src AND dst

Description The source byte, is ANDed to the destination byte. The result is lost but condition flags N and Z are updated accordingly. The source is a memory byte, and the destination is A register. This instruction can be used to perform bit tests on A.

Instruction Overview

mnem	dst	src
BCP	A	Mem

Condition Flags

H	I	N	Z	C
		N	Z	

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A5	XX	
A	short	3	2		B5	XX	
A	long	4	3		C5	MS	LS
A	(X)	3	1		F5		
A	(short,X)	4	2		E5	XX	
A	(long,X)	5	3		D5	MS	LS
A	(Y)	4	2	90	F5		
A	(short,Y)	5	3	90	E5	XX	
A	(long,Y)	6	4	90	D5	MS	LS
A	[short]	5	3	92	B5	XX	
A	[long.w]	6	3	92	C5	XX	
A	([short],X)	6	3	92	E5	XX	
A	([long.w],X)	7	3	92	D5	XX	
A	([short],Y)	6	3	91	E5	XX	
A	([long.w],Y)	7	3	91	D5	XX	

See Also: CP, TNZ

BRES

Bit Reset

BRES

Syntax bres dst,#pos pos = [0..7] e.g.: bres PADR,#6

Operation dst <= dst AND (2**pos)

Description Read the destination byte, reset the corresponding bit (bit position), and write the result in destination byte. The destination is a memory byte. The bit position is a constant. This instruction is fast, compact, and does not affect any register. Very useful for boolean variable manipulation.

Instruction Overview

mnem	dst	bit position
BRES	Mem	#pos

Condition Flags

H	I	N	Z	C

Detailed Description

dst	pos = 0..7
short	n = 11+2.pos
[short]	n = 11+2.pos

cy	lgth
5	2
7	3

Op-Code(s)			
	1n	XX	
92	1n	XX	

See Also: BSET

BSET — Bit Set — BSET

Syntax bset dst,#pos pos = [0..7] e.g.: bset PADR,#0

Operation dst <= dst OR (2**pos)

Description Read the destination byte, set the corresponding bit (bit position), and write the result in destination byte. The destination is a memory byte. The bit position is a constant. This instruction is fast, compact, and does not affect any register. Very useful for boolean variable manipulation.

Instruction Overview

mnem	dst	bit position
BSET	Mem	#pos

Condition Flags

H	I	N	Z	C

Detailed Description

dst	pos = 0..7
short	n = 10+2.pos
[short]	n = 10+2.pos

cy	lgth
5	2
7	3

Op-Code(s)			
	1n	XX	
92	1n	XX	

See Also: BRES

BTJF

Bit Test and Jump if False

BTJF

Syntax

btjf dst,#pos,rel pos = [0..7], rel is relative jump label
e.g.: btjf PADR,#3,skip

Operation

PC = PC + 3
PC = PC + rel IF (dst AND (2**pos)) = 0

Description

Read the destination byte, test the corresponding bit (bit position), and jump to 'rel' label if the bit is false (0), else continue the program to the next instruction. The tested bit is saved in the C flag. The destination is a memory byte. The bit position is a constant. The jump label points to an memory location around the instruction (relative jump). This instruction is used for boolean variable manipulation, H/W register flag tests, or I/O polling method. This instruction is fast, compact, and does not affect any register. Very useful for boolean variable manipulation.

Instruction Overview

mnem	dst	bit position	jump label
BTJF	Mem	#pos	rel

Condition Flags

H	I	N	Z	C
				C

Detailed Description

dst	pos = 0..7
short	n = 01+2.pos
[short]	n = 01+2.pos

cy	lgth
5	3
7	4

Op-Code(s)			
	0n	XX	XX
92	0n	XX	XX

See also: BTJT

BTJT — Bit Test and Jump if True — BTJT

Syntax

	btjt	dst,#pos,rel	pos = [0..7], rel is relative jump label
	e.g.:	btjt	PADR,#7,skip

Operation

PC = PC + 3
PC = PC + rel IF (dst AND (2**pos)) <> 0

Description

Read the destination byte, test the corresponding bit (bit position), and jump to 'rel' label if the bit is true (1), else continue the program to the next instruction. The tested bit is saved in the C flag. The destination is a memory byte. The bit position is a constant. The jump label points to an memory location around the instruction (relative jump). This instruction is used for boolean variable manipulation, H/W register flag tests, or I/O polling method.

Instruction Overview

mnem	dst	bit position	jump label
BTJT	Mem	#pos	rel

Condition Flags

H	I	N	Z	C
				C

Detailed Description

dst	pos = 0..7
short	n = 00+2.pos
[short]	n = 00+2.pos

cy	lgth
5	3
7	4

Op-Code(s)			
	0n	XX	XX
92	0n	XX	XX

See Also: BTJF

CALL

CALL Subroutine (Absolute)

CALL

Syntax CALL dst e.g.: call divide32_16

Operation
PC = PC+lgth
(SP--) = LSB (PC)
(SP--) = MSB (PC)
PC = dst

Description The current PC register value is pushed onto the stack, then PC is loaded with the destination address. This instruction should be used versus CALLR when developing a program.

Instruction Overview

mnem	dst
CALL	Mem

Condition Flags

H	I	N	Z	C

Detailed Description

dst	cy	lgth	Op-Code(s)			
short	5	2		BD	XX	
long	6	3		CD	MS	LS
(X)	5	1		FD		
(short,X)	6	2		ED	XX	
(long,X)	7	3		DD	MS	LS
(Y)	6	2	90	FD		
(short,Y)	7	3	90	ED	XX	
(long,Y)	8	4	90	DD	MS	LS
[short]	7	3	92	BD	XX	
[long.w]	8	3	92	CD	XX	
([short],X)	8	3	92	ED	XX	
([long.w],X)	9	3	92	DD	XX	
([short],Y)	8	3	91	ED	XX	
([long.w],Y)	9	3	91	DD	XX	

See Also: CALLR, RET

CALLR

CALL Subroutine Relative

CALLR

Syntax CALLR dst e.g.: callr chk_pol

Operation

PC = PC+lgth
(SP--) = LSB(PC)
(SP--) = MSB(PC)
PC = PC + dst

Description The current PC register value is pushed onto the stack, then PC is loaded with the relative destination addresss.This instruction is used, once a program is debugged, to shrink the overall program size.

Instruction Overview

mnem	dst
CALLR	Mem

Condition Flags

H	I	N	Z	C

Detailed Description

dst
short
[short]

cy	lgth
6	2
8	3

Op-Code(s)			
	AD	XX	
92	AD	XX	

See Also: CALL, RET

CLR — CLEAR — CLR

Syntax clr dst e.g.: clr X

Operation dst <= 00

Description The destination byte is forced to 00 value. The destination is either a memory byte location, or a register. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
CLR	Mem
CLR	Reg

Condition Flags

H	I	N	Z	C
		0	1	
		0	1	

Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		4F		
X	3	1		5F		
Y	4	2	90	5F		
short	5	2		3F	XX	
(X)	5	1		7F		
(short,X)	6	2		6F	XX	
(Y)	6	2	90	7F		
(short,Y)	7	3	90	6F	XX	
[short]	7	3	92	3F	XX	
([short],X)	8	3	92	6F	XX	
([short],Y)	8	3	91	6F	XX	

See Also: LD

CP

Compare

CP

Syntax cp dst,src e.g.: cp A,(tbl,X)

Operation {N, Z, C} = Test (dst - src)

Description The source byte is subtracted from the destination byte and the result is lost. However, N, Z, C are updated according to the result. The destination is a register, and the source is a memory byte. This instruction generally is placed just before a conditional jump instruction.

Instruction Overview

mnem	dst	src
CP	Reg	Mem

Condition Flags

H	I	N	Z	C
		N	Z	C

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A1	XX	
A	short	3	2		B1	XX	
A	long	4	3		C1	MS	LS
A	(X)	3	1		F1		
A	(short,X)	4	2		E1	XX	
A	(long,X)	5	3		D1	MS	LS
A	(Y)	4	2	90	F1		
A	(short,Y)	5	3	90	E1	XX	
A	(long,Y)	6	4	90	D1	MS	LS
A	[short]	5	3	92	B1	XX	
A	[long.w]	6	3	92	C1	XX	
A	([short],X)	6	3	92	E1	XX	
A	([long.w],X)	7	3	92	D1	XX	
A	([short],Y)	6	3	91	E1	XX	
A	([long.w],Y)	7	3	91	D1	XX	

(Continued on next page)

CP Detailed Description (Cont'd)

dst	src	cy	lgth	Op-Code(s)			
X	#byte	2	2		A3	XX	
X	short	3	2		B3	XX	
X	long	4	3		C3	MS	LS
X	(X)	3	1		F3		
X	(short,X)	4	2		E3	XX	
X	(long,X)	5	3		D3	MS	LS
X	[short]	5	3	92	B3	XX	
X	[long.w]	6	3	92	C3	XX	
X	([short],X)	6	3	92	E3	XX	
X	([long.w],X)	7	3	92	D3	XX	

dst	src	cy	lgth	Op-Code(s)			
Y	#byte	3	3	90	A3	XX	
Y	short	4	3	90	B3	XX	
Y	long	5	4	90	C3	MS	LS
Y	(Y)	4	2	90	F3		
Y	(short,Y)	5	3	90	E3	XX	
Y	(long,Y)	6	4	90	D3	MS	LS
Y	[short]	5	3	91	B3	XX	
Y	[long.w]	6	3	91	C3	XX	
Y	([short],Y)	6	3	91	E3	XX	
Y	([long.w],Y)	7	3	91	D3	XX	

See Also: TNZ, BCP

CPL — Logical 1-Complement — CPL

Syntax cpl dst e.g.: cpl (X)

Operation dst <= dst XOR FF, or FF - dst

Description The destination byte is read, then each bit is toggled (inverted) and the result is written at the destination byte. The destination is either a memory byte or a register. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
CPL	Mem
CPL	Reg

Condition Flags

H	I	N	Z	C
		N	Z	1
		N	Z	1

Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		43		
X	3	1		53		
Y	4	2	90	53		
short	5	2		33	XX	
(X)	5	1		73		
(short,X)	6	2		63	XX	
(Y)	6	2	90	73		
(short,Y)	7	3	90	63	XX	
[short]	7	3	92	33	XX	
([short],X)	8	3	92	63	XX	
([short],Y)	8	3	91	63	XX	

See Also: NEG, XOR, AND, OR

DEC — Decrement — DEC

Syntax dec dst e.g.: dec Y

Operation dst <= dst - 1

Description The destination byte is read, then decremented by one, and the result is written at the destination byte. The destination is either a memory byte or a register. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
DEC	Mem
DEC	Reg

Condition Flags

H	I	N	Z	C
		N	Z	
		N	Z	

Detailed description

dst	cy	lgth	Op-Code(s)			
A	3	1		4A		
X	3	1		5A		
Y	4	2	90	5A		
short	5	2		3A	XX	
(X)	5	1		7A		
(short,X)	6	2		6A	XX	
(Y)	6	2	90	7A		
(short,Y)	7	3	90	6A	XX	
[short]	7	3	92	3A	XX	
([short],X)	8	3	92	6A	XX	
([short],Y)	8	3	91	6A	XX	

See Also: INC

HALT

HALT Oscillator (CPU + Peripherals)

HALT

Syntax HALT

Operation I = 0, The Oscillator is stopped till an interrupt occur.

Description The interrupt mask is reset, allowing interrupts to be fetched. Then the Oscillator is stopped thus stopping the CPU and all internal peripherals, reducing the microcontroller to its lowest possible power consumption. The micro will continue the program upon an external interrupt, by restarting the oscillator (with 4096 clock cycles delay), and then, fetching the corresponding external interrupt, which is generally either an I/O interrupt or an external Reset.

Instruction Overview

mnem
HALT

Condition Flags

H	I	N	Z	C
	0			

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	8E		

See Also: WFI

INC

Increment

INC

Syntax inc dst e.g.: inc counter

Operation dst <= dst + 1

Description The destination byte is read, then incremented by one, and the result is written at the destination byte. The destination is either a memory byte or a register.This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
INC	Mem
INC	Reg

Condition Flags

H	I	N	Z	C
		N	Z	
		N	Z	

Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		4C		
X	3	1		5C		
Y	4	2	90	5C		
short	5	2		3C	XX	
(X)	5	1		7C		
(short,X)	6	2		6C	XX	
(Y)	6	2	90	7C		
(short,Y)	7	3	90	6C	XX	
[short]	7	3	92	3C	XX	
([short],X)	8	3	92	6C	XX	
([short],Y)	8	3	91	6C	XX	

See Also: DEC

IRET

Interrupt Return

IRET

Syntax IRET

Operation

CC	= (++SP)
A	= (++SP)
X	= (++SP)
MSB (PC)	= (++SP)
LSB (PC)	= (++SP)

Description Placed at the end of an interrupt routine, return to the original program context before the interrupt occurred. All registers which have been saved/pushed onto the stack (Y excepted) are restored/popped.

Instruction Overview

mnem
IRET

Condition Flags

H	I	N	Z	C
H	I	N	Z	C

X: Condition Flags set or reset according to the first byte pulled from the stack

Detailed Description

cy	lgth
9	1

Op-Code(s)			
	80		

See Also: Interrupts, TRAP

JP

Jump (absolute)

JP

Syntax jp dst e.g.: jp test

Operation PC <= dst

Description The unconditional jump simply replaces the content of PC by dst. Control then passes to the statement addressed by the program counter. This instruction should be used instead of JRA during S/W development.

Instruction Overview

mnem	dst
JP	Mem

Condition Flags

H	I	N	Z	C

Detailed Description

dst	cy	lgth	Op-Code(s)			
short	2	2		BC	XX	
long	3	3		CC	MS	LS
(X)	2	1		FC		
(short,X)	3	2		EC	XX	
(long,X)	4	3		DC	MS	LS
(Y)	3	2	90	FC		
(short,Y)	4	3	90	EC	XX	
(long,Y)	5	4	90	DC	MS	LS
[short]	4	3	92	BC	XX	
[long.w]	5	3	92	CC	XX	
([short],X)	5	3	92	EC	XX	
([long.w],X)	6	3	92	DC	XX	
([short],Y)	5	3	91	EC	XX	
([long.w],Y)	6	3	91	DC	XX	

See Also: JRA

JRA

Jump Relative Always

JRA

Syntax jra dst e.g.: jra loop

Operation PC <= PC + dst

Description Unconditional relative jump. PC is updated by the signed addition of PC and dst. Control then passes to the statement addressed by the program counter. This instruction may be used, once the S/W debugged to fasten and shrink a program.

Instruction Overview

mnem	dst
JRA	Mem

Condition Flags

H	I	N	Z	C

Detailed Description

dst
rel
[rel]

cy	lgth
3	2
5	3

Op-Code(s)			
	20	XX	
92	20	XX	

See Also: JP

JRxx — Conditional Jump Relative Instruction — JRxx

Syntax jrxx dst e.g.: jrxx loop

Operation PC <= PC + dst if Condition is True

Description Conditional relative jump. PC is updated by the signed addition of PC and dst if the condition is true. Control then passes to the statement addressed by the program counter. Else, the program continues normally.

Instruction Overview

mnem	dst
JRxx	Mem

Condition Flags

H	I	N	Z	C

Instruction List

mnem	meaning	sym	Condition	Op-Code (OC)
JRC	Carry		C = 1	25
JREQ	Equal	=	Z = 1	27
JRF	False		False	21
JRH	Half-Carry		H = 1	29
JRIH	Interrupt Line is High			2F
JRIL	Interrupt Line is Low			2E
JRM	Interrupt Mask		I = 1	2D
JRMI	Minus	< 0	N = 1	2B
JRNC	Not Carry		C = 0	24
JRNE	Not Equal	<> 0	Z = 0	26
JRNH	Not Half-Carry		H = 0	28
JRNM	Not Interrupt Mask		I = 0	2C
JRPL	Plus	>= 0	N = 0	2A
JRT	True		True	20
JRUGE	Unsigned Greater or Equal	>=	C = 0	24
JRUGT	Unsigned Greater Then	>	(C or Z) = 0	22
JRULE	Unsigned Lower or Equal	<=	(C or Z) = 1	23
JRULT	Unsigned Lower Than	<	C = 1	25

Detailed Description

dst	cy	lgth
rel	3	2
[rel]	5	3

Op-Code(s)			
	OC	XX	
92	OC	XX	

LD — Load — LD

Syntax ld dst,src e.g.: ld A,#$15

Operation dst <= src

Description Load the destination byte with the source byte.

Instruction Overview

mnem	dst	src
LD	reg	mem
LD	mem	reg
LD	reg	reg
LD	S	reg
LD	reg	S

Condition Flags

H	I	N	Z	C
		N	Z	
		N	Z	

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A6	XX	
A	short	3	2		B6	XX	
A	long	4	3		C6	MS	LS
A	(X)	3	1		F6		
A	(short,X)	4	2		E6	XX	
A	(long,X)	5	3		D6	MS	LS
A	(Y)	4	2	90	F6		
A	(short,Y)	5	3	90	E6	XX	
A	(long,Y)	6	4	90	D6	MS	LS
A	[short]	5	3	92	B6	XX	
A	[long.w]	6	3	92	C6	XX	
A	([short],X)	6	3	92	E6	XX	
A	([long.w],X)	7	3	92	D6	XX	
A	([short],Y)	6	3	91	E6	XX	
A	([long.w],Y)	7	3	91	D6	XX	

LD Detailed Description (Cont'd)

dst	src	cy	lgth	Op-Code(s)			
short	A	4	2		B7	XX	
long	A	5	3		C7	MS	LS
(X)	A	4	1		F7		
(short,X)	A	5	2		E7	XX	
(long,X)	A	6	3		D7	MS	LS
(Y)	A	5	2	90	F7		
(short,Y)	A	6	3	90	E7	XX	
(long,Y)	A	7	4	90	D7	MS	LS
[short]	A	6	3	92	B7	XX	
[long.w]	A	7	3	92	C7	XX	
([short],X)	A	7	3	92	E7	XX	
([long.w],X)	A	8	3	92	D7	XX	
([short],Y)	A	7	3	91	E7	XX	
([long.w],Y)	A	8	3	91	D7	XX	

dst	src	cy	lgth	Op-Code(s)			
X	#byte	2	2		AE	XX	
X	short	3	2		BE	XX	
X	long	4	3		CE	MS	LS
X	(X)	3	1		FE		
X	(short,X)	4	2		EE	XX	
X	(long,X)	5	3		DE	MS	LS
X	[short]	5	3	92	BE	XX	
X	[long.w]	6	3	92	CE	XX	
X	([short],X)	6	3	92	EE	XX	
X	([long.w],X)	7	3	92	DE	XX	

dst	src	cy	lgth	Op-Code(s)			
short	X	4	2		BF	XX	
long	X	5	3		CF	MS	LS
(X)	X	4	1		FF		
(short,X)	X	5	2		EF	XX	
(long,X)	X	6	3		DF	MS	LS
[short]	X	6	3	92	BF	XX	
[long.w]	X	7	3	92	CF	XX	
([short],X)	X	7	3	92	EF	XX	
([long.w],X)	X	8	3	92	DF	XX	

LD Detailed Description (Cont'd)

dst	src	cy	lgth	Op-Code(s)			
Y	#byte	3	3	90	AE	XX	
Y	short	4	3	90	BE	XX	
Y	long	5	4	90	CE	MS	LS
Y	(Y)	4	2	90	FE		
Y	(short,Y)	5	3	90	EE	XX	
Y	(long,Y)	6	4	90	DE	MS	LS
Y	[short]	5	3	91	BE	XX	
Y	[long.w]	6	3	91	CE	XX	
Y	([short],Y)	6	3	91	EE	XX	
Y	([long.w],Y)	7	3	91	DE	XX	

dst	src	cy	lgth	Op-Code(s)			
short	Y	5	3	90	BF	XX	
long	Y	6	4	90	CF	MS	LS
(Y)	Y	5	2	90	FF		
(short,Y)	Y	6	3	90	EF	XX	
(long,Y)	Y	7	4	90	DF	MS	LS
[short]	Y	6	3	91	BF	XX	
[long.w]	Y	7	3	91	CF	XX	
([short],Y)	Y	7	3	91	EF	XX	
([long.w],Y)	Y	8	3	91	DF	XX	

dst	src	cy	lgth	Op-Code(s)			
X	A	2	1		97		
A	X	2	1		9F		
Y	A	3	2	90	97		
A	Y	3	2	90	9F		
Y	X	3	2	90	93		
X	Y	2	1		93		
A	S	2	1		9E		
S	A	2	1		95		
X	S	2	1		96		
S	X	2	1		94		
Y	S	3	2	90	96		
S	Y	3	2	90	94		

See Also: CLR

MUL — Multiply (unsigned) — MUL

Syntax mul dst,src e.g.: mul X,A

Operation dst:src <= dst x src

Description The source byte, is multiplied (unsigned), with the destination byte. The 16 bit MSB word result is saved in dst location, and the LSB one in src location.

Instruction Overview

mnem	dst	src
MUL	X:A	X, A
MUL	Y:A	Y, A

Condition Flags

H	I	N	Z	C
0				0
0				0

Detailed Description

dst	src
X	A
Y	A

cy	lgth
11	1
12	2

Op-Code(s)			
	42		
90	42		

See Also: ADD, ADC, SUB, SBC

NEG

Negate (Logical 2-Complement)

NEG

Syntax neg dst e.g.: neg (X)

Operation dst <= (dst XOR FF) + 1, or 00 - dst

Description The destination byte is read, then each bit is toggled (inverted), and the result is incremented before it is written at the destination byte. The destination is either a memory byte or a register. The Carry is cleared if the result is zero. This instruction is used to negate signed values. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
NEG	Mem
NEG	Reg

Condition Flags

H	I	N	Z	C
		N	Z	C
		N	Z	C

Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		40		
X	3	1		50		
Y	4	2	90	50		
short	5	2		30	XX	
(X)	5	1		70		
(short,X)	6	2		60	XX	
(Y)	6	2	90	70		
(short,Y)	7	3	90	60	XX	
[short]	7	3	92	30	XX	
([short],X)	8	3	92	60	XX	
([short],Y)	8	3	91	60	XX	

See Also: CPL, AND, OR, XOR

NOP

No operation

NOP

Syntax nop

Operation

Description Does nothing. This instruction can be used either to disable an instruction, or to build a waiting delay.

Instruction Overview

mnem
NOP

Condition Flag

H	I	N	Z	C

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	9D		

See Also: JRF

OR — Logical OR — OR

Syntax or dst,src e.g.: or A,#%00110101

Operation dst <= dst OR src

Description The source byte, is ORed with the destination byte and the result is stored in the destination byte. The source is a memory byte, and the destination is the Accumulator register.

Truth Table

OR	0	1
0	0	1
1	1	1

Instruction Overview

mnem	dst	src
OR	A	Mem

Condition Flags

H	I	N	Z	C
		N	Z	

Detailed Description

dst	src
A	#byte
A	short
A	long
A	(X)
A	(short,X)
A	(long,X)
A	(Y)
A	(short,Y)
A	(long,Y)
A	[short]
A	[long.w]
A	([short],X)
A	([long.w],X)
A	([short],Y)
A	([long.w],Y)

cy	lgth
2	2
3	2
4	3
3	1
4	2
5	3
4	2
5	3
6	4
5	3
6	3
6	3
7	3
6	3
7	3

Op-Code(s)			
	AA	XX	
	BA	XX	
	CA	MS	LS
	FA		
	EA	XX	
	DA	MS	LS
90	FA		
90	EA	XX	
90	DA	MS	LS
92	BA	XX	
92	CA	XX	
92	EA	XX	
92	DA	XX	
91	EA	XX	
91	DA	XX	

See Also: AND, XOR, CPL, NEG

POP

Pop from Stack

POP

Syntax: pop dst e.g.: pop CC

Operation: dst <= (++SP)

Description: Restore from the stack a data byte which will be placed in dst location. The stack pointer is incremented by one. Use to restore a register value.

Instruction Overview

mnem	dst
POP	A
POP	X
POP	Y
POP	CC

Condition Flag

H	I	N	Z	C
H	I	N	Z	C

X: Load Condition Flag from the stack

Detailed Description

dst
A
X
Y
CC

cy	lgth
4	1
4	1
5	2
4	1

Op-Code(s)			
	84		
	85		
90	85		
	86		

See Also: PUSH, RSP

PUSH

Push into the Stack

PUSH

Syntax push src e.g.: push A

Operation (SP--) <= dst

Description Save into the stack the dst byte location. The stack pointer is decremented by one. Used to save a register value.

Instruction Overview

mnem	dst
PUSH	A
PUSH	X
PUSH	Y
PUSH	CC

Condition Flag

H	I	N	Z	C

Detailed Description

dst
A
X
Y
CC

cy	lgth
3	1
3	1
4	2
3	1

Op-Code(s)			
	88		
	89		
90	89		
	8A		

See Also: POP, RSP

RCF

Reset Carry Flag

RCF

Syntax rcf

Operation C = 0

Description Clear the carry flag of the CC register. May be used as a boolean used controlled flags.

Instruction Overview

mnem
RCF

Condition Flags

H	I	N	Z	C
				0

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	98		

See Also: SCF

RET — Return from subroutine — RET

Syntax ret

Operation
MSB (PC) = (++SP)
LSB (PC) = (++SP)

Description Restore the PC from the stack. The stack pointer is incremented twice.This instruction is the last one of a subroutine.

Instruction Overview

mnem
RET

Condition Flags

H	I	N	Z	C

Detailed Description

cy	lgth
6	1

Op-Code(s)			
	81		

See Also: CALL, CALLR

RIM

Reset Interrupt Mask/Enable Interrupt

RIM

Syntax rim

Operation I = 0

Description Clear the Interrupt mask of the CC register, which enable interrupts. This instruction is generally put in the main program, after the reset routine, once all desired interrupts have been properly configured. This instruction is not needed before both WFI and HALT instructions.

Instruction Overview

mnem
RIM

Condition Flags

H	I	N	Z	C
	0			

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	9A		

See Also: SIM

RLC

Rotate Left Logical through Carry

RLC

Syntax rlc dst e.g.: rlc (X)

Operation

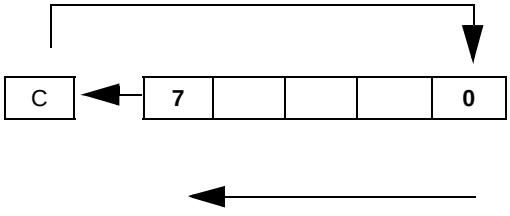
Description The destination is either a memory byte or a register. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

RLC	Mem
RLC	Reg
RLC	Mem

Condition Flag

H	I	N	Z	C
		N	Z	bit 7
		N	Z	bit 7



Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		49		
X	3	1		59		
Y	4	2	90	59		
short	5	2		39	XX	
(X)	5	1		79		
(short,X)	6	2		69	XX	
(Y)	6	2	90	79		
(short,Y)	7	3	90	69	XX	
[short]	7	3	92	39	XX	
([short],X)	8	3	92	69	XX	
([short],Y)	8	3	91	69	XX	

See Also: RRC, SLL, SRL, SRA, ADC, SWAP, SLA

RRC

Rotate Right Logical through Carry

RRC

Syntax rrc dst e.g.: rrc (X)

Operation

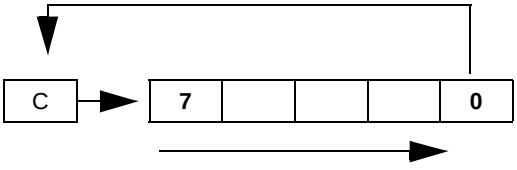
Description The destination is either a memory byte location or a register. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
RRC	Mem
RRC	Reg

Condition Flag

H	I	N	Z	C
		N	Z	bit 0
		N	Z	bit 0



Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		46		
X	3	1		56		
Y	4	2	90	56		
short	5	2		36	XX	
(X)	5	1		76		
(short,X)	6	2		66	XX	
(Y)	6	2	90	76		
(short,Y)	7	3	90	66	XX	
[short]	7	3	92	36	XX	
([short],X)	8	3	92	66	XX	
([short],Y)	8	3	91	66	XX	

See Also: RLC, SRL, SLL, SRA, SWAP, ADC, SLA

RSP

Reset Stack Pointer

RSP

Syntax rsp

Operation SP = Reset Value

Description Reset the stack pointer to its reset initial value.This instruction may be put as first executed instruction in the reset routine.

Trick It may be used to test current stack size used with an ST7 independent program.

Instruction Overview

mnem
RSP

Condition Flags

H	I	N	Z	C

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	9C		

See Also: PUSH, POP

SBC

Substraction with Carry

SBC

Syntax sbc dst,src e.g.: sbc A,#$15

Operation dst <= dst - src - C

Description The source byte, along with the carry flag, is subtracted from the destination byte and the result is stored in the destination byte. The source is a memory byte, and the destination is the A register.

Instruction Overview

mnem	dst	src
SBC	A	Mem

Condition Flags

H	I	N	Z	C
		N	Z	C

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A2	XX	
A	short	3	2		B2	XX	
A	long	4	3		C2	MS	LS
A	(X)	3	1		F2		
A	(short,X)	4	2		E2	XX	
A	(long,X)	5	3		D2	MS	LS
A	(Y)	4	2	90	F2		
A	(short,Y)	5	3	90	E2	XX	
A	(long,Y)	6	4	90	D2	MS	LS
A	[short]	5	3	92	B2	XX	
A	[long.w]	6	3	92	C2	XX	
A	([short],X)	6	3	92	E2	XX	
A	([long.w],X)	7	3	92	D2	XX	
A	([short],Y)	6	3	91	E2	XX	
A	([long.w],Y)	7	3	91	D2	XX	

See Also: ADD,SUB,SBC, MUL

SCF — Set Carry Flag — SCF

Syntax scf

Operation C = 1

Description Set the carry flag of the CC register. It may be used as user controlled flag.

Instruction Overview

mnem
SCF

Condition Flags

H	I	N	Z	C
				1

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	99		

See Also: RCF

SIM — Set Interrupt Mask/Disable Interrupt — SIM

Syntax: sim

Operation: I = 1

Description: Set the Interrupt mask of the CC register, which disables interrupts. This instruction is useless at the beginning of an interrupt/reset routine

Instruction Overview

mnem
SIM

Condition Flags

H	I	N	Z	C
	1			

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	9B		

See Also: RIM

SLA

Shift Left Arithmetic

SLA

Syntax sla dst e.g.: sla (X)

Operation

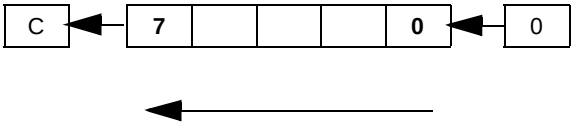
Description The destination is either a memory byte or a register.This instruction is equivalent to SLL one.

Instruction Overview

mnem	dst
SLA	Mem
SLA	Reg

Condition Flags

H	I	N	Z	C
		N	Z	bit 7
		N	Z	bit 7



Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		48		
X	3	1		58		
Y	4	2	90	58		
short	5	2		38	XX	
(X)	5	1		78		
(short,X)	6	2		68	XX	
(Y)	6	2	90	78		
(short,Y)	7	3	90	68	XX	
[short]	7	3	92	38	XX	
([short],X)	8	3	92	68	XX	
([short],Y)	8	3	91	68	XX	

See Also: SRL, SRA, RRC, RLC, SWAP, SLL

SLL — Shift Left Logical — SLL

Syntax sll dst e.g.: sll (X)

Operation

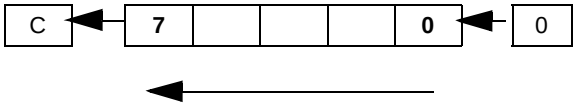
Description The destination is either a memory byte or a register.It double the affected value. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
SLL	Mem
SLL	Reg

Condition Flags

H	I	N	Z	C
		N	Z	bit 7
		N	Z	bit 7



Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		48		
X	3	1		58		
Y	4	2	90	58		
short	5	2		38	XX	
(X)	5	1		78		
(short,X)	6	2		68	XX	
(Y)	6	2	90	78		
(short,Y)	7	3	90	68	XX	
[short]	7	3	92	38	XX	
([short],X)	8	3	92	68	XX	
([short],Y)	8	3	91	68	XX	

See Also: SLA, SRA, SRL, RRC, RLC, SWAP

SRA

Shift Right Arithmetic

SRA

Syntax sra dst e.g.: sra (X)

Operation

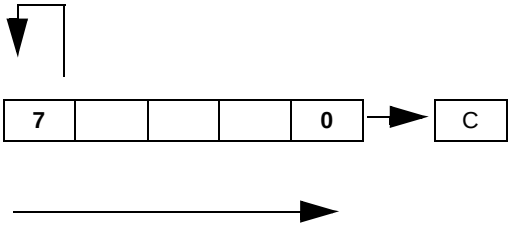
Description The destination is either a memory byte or a register.It perform an signed division by 2: The sign bit 7 is not modified.This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
SRA	Mem
SRA	Reg

Condition Flags

H	I	N	Z	C
		N	Z	bit 0
		N	Z	bit 0



Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		47		
X	3	1		57		
Y	4	2	90	57		
short	5	2		37	XX	
(X)	5	1		77		
(short,X)	6	2		67	XX	
(Y)	6	2	90	77		
(short,Y)	7	3	90	67	XX	
[short]	7	3	92	37	XX	
([short],X)	8	3	92	67	XX	
([short],Y)	8	3	91	67	XX	

See Also: SRL, SLL, RRC, RLC, SWAP

SRL — Shift Right Logical — SRL

Syntax srl dst e.g.: srl (X)

Operation

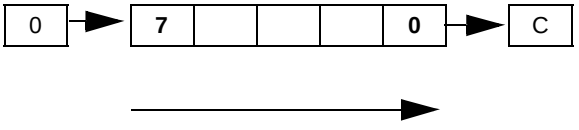
Description The destination is either a memory byte or a register.It perform an unsigned division by 2.This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
SRL	Mem
SRL	Reg

Condition Flags

H	I	N	Z	C
		0	Z	bit 0
		0	Z	bit 0



Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		44		
X	3	1		54		
Y	4	2	90	54		
short	5	2		34	XX	
(X)	5	1		74		
(short,X)	6	2		64	XX	
(Y)	6	2	90	74		
(short,Y)	7	3	90	64	XX	
[short]	7	3	92	34	XX	
([short],X)	8	3	92	64	XX	
([short],Y)	8	3	91	64	XX	

See Also: RLC, RRC, SRL, SRA, SWAP, SLL

SUB — Substraction — SUB

Syntax sub dst,src e.g.: sub A,#%11001010

Operation dst <= dst - src

Description The source byte is subtracted from the destination byte and the result is stored in the destination byte. The source is a memory byte, and the destination is the A register.

Instruction Overview

mnem	dst	src
SUB	A	Mem

Condition Flags

H	I	N	Z	C
		N	Z	C

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A0	XX	
A	short	3	2		B0	XX	
A	long	4	3		C0	MS	LS
A	(X)	3	1		F0		
A	(short,X)	4	2		E0	XX	
A	(long,X)	5	3		D0	MS	LS
A	(Y)	4	2	90	F0		
A	(short,Y)	5	3	90	E0	XX	
A	(long,Y)	6	4	90	D0	MS	LS
A	[short]	5	3	92	B0	XX	
A	[long.w]	6	3	92	C0	XX	
A	([short],X)	6	3	92	E0	XX	
A	([long.w],X)	7	3	92	D0	XX	
A	([short],Y)	6	3	91	E0	XX	
A	([long.w],Y)	7	3	91	D0	XX	

See Also: ADD, ADC, SBC, MUL

SWAP

Swap nibbles

SWAP

Syntax swap dst e.g.: swap counter

Operation

Description The destination byte upper and low nibbles are swapped over. The destination is either a memory byte or a register.This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
SWAP	Mem
SWAP	Reg

Condition Flags

H	I	N	Z	C
		N	Z	
		N	Z	

Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		4E		
X	3	1		5E		
Y	4	2	90	5E		
short	5	2		3E	XX	
(X)	5	1		7E		
(short,X)	6	2		6E	XX	
(Y)	6	2	90	7E		
(short,Y)	7	3	90	6E	XX	
[short]	7	3	92	3E	XX	
([short],X)	8	3	92	6E	XX	
([short],Y)	8	3	91	6E	XX	

See Also: RRC, RLC, SLL, SRL, SRA

TNZ

Test for Negative or Zero

TNZ

Syntax tnz dst e.g.: tnz A

Operation {N, Z} = Test(dst)

Description The destination byte is tested and both N and Z flags are updated accordingly.This instruction is compact, and does not affect any register when used with RAM variables.

Instruction Overview

mnem	dst
TNZ	Mem
TNZ	Reg

Condition Flags

H	I	N	Z	C
		N	Z	
		N	Z	

Detailed Description

dst	cy	lgth	Op-Code(s)			
A	3	1		4D		
X	3	1		5D		
Y	4	2	90	5D		
short	4	2		3D	XX	
(X)	4	1		7D		
(short,X)	5	2		6D	XX	
(Y)	5	2	90	7D		
(short,Y)	6	3	90	6D	XX	
[short]	6	3	92	3D	XX	
([short],X)	7	3	92	6D	XX	
([short],Y)	7	3	91	6D	XX	

See Also: CP, BCP

TRAP

Software Interrupt

TRAP

Syntax TRAP

Operation

PC = PC + 1
(SP--) = LSB (PC)
(SP--) = MSB (PC)
(SP--) = X
(SP--) = A
(SP--) = CC
PC = Vector Contents

Description When processed, this instruction force the trap interrupt to occur and to be processed. It cannot be masked by I flag.

Instruction Overview

mnem
TRAP

Condition Flags

H	I	N	Z	C
	1			

Detailed Description

cy	lgth
10	1

Op-Code(s)			
	83		

See Also: IRET

WFI

Wait for Interrupt (CPU Stopped, Low Power Mode)

WFI

Syntax WFI

Operation I = 0, The CPU Clock is stopped till an interrupt occur. Internal Peripheral are still running.

Description The interrupt flag is cleared, allowing interrupts to be fetched. Then the CPU clock is stopped, reducing the microcontroller to a lower power consumption. The micro will continue the program upon an internal or external interrupt.

Instruction Overview

mnem
WFI

Condition Flags

H	I	N	Z	C
	0			

Detailed Description

cy	lgth
2	1

Op-Code(s)			
	8F		

See Also: HALT

XOR

Logical Exclusive OR

XOR

Syntax xor dst,src e.g.: xor A,#%00110101

Operation dst <= dst XOR src

Description The source byte, is XORed with the destination byte and the result is stored in the destination byte. The source is a memory byte, and the destination is the A register.

Truth Table

XOR	0	1
0	0	1
1	1	0

Instruction Overview

mnem	dst	src
XOR	A	Mem

Condition Flags

H	I	N	Z	C
		N	Z	

Detailed Description

dst	src	cy	lgth	Op-Code(s)			
A	#byte	2	2		A8	XX	
A	short	3	2		B8	XX	
A	long	4	3		C8	MS	LS
A	(X)	3	1		F8		
A	(short,X)	4	2		E8	XX	
A	(long,X)	5	3		D8	MS	LS
A	(Y)	4	2	90	F8		
A	(short,Y)	5	3	90	E8	XX	
A	(long,Y)	6	4	90	D8	MS	LS
A	[short]	5	3	92	B8	XX	
A	[long.w]	6	3	92	C8	XX	
A	([short],X)	6	3	92	E8	XX	
A	([long.w],X)	7	3	92	D8	XX	
A	([short],Y)	6	3	91	E8	XX	
A	([long.w],Y)	7	3	91	D8	XX	

See Also: AND, OR, CPL, NEG

5 SOFTWARE Library

In order to simplify and hasten the development of any ST7 application, many useful standard routines are shown in this chapter. They are general purpose ones, since they do not interact with any H/W cell. These routines are split in 8 main groups:

Table of Contents:

5.1 TIPS GENERAL TRICKS

Trick 1:How to increment A up to XX?

```
            clr     A ; A=00h.
loop1       cp      A,#18    ; 18 is our example, you can take #XX.
            jreq    exit1        ; when A=#18, exit.
            adc     A,#0         ; The advantage to use adc and not add (add A,#1) is
            jrnc     loop1       ; that when A=#18, C=0 and A keeps the good value.
                     ; The instruction jra is also possible.
```

exit 1

Trick 2: How to decrement A down to XX?

```
            ld      A,#$FF       ; A is put at FF to be greater than #XX
loop2       cp      A,#$A5       ; A5 is here our example, you can take any value #XX.
            jreq    exit2        ; When A=#$A5, exit.
            adc     A,#$FF
            jrc     loop2        ; The instruction jra is also possible.
```

exit 2

Trick 3: How to convert A, hexa value between $00 (0) and $63 (99) in decimal?

```
            clr     dec_nbr      ; dec_nbr is the variable used to store the decimal number
temp        sub     A,#10        ; each nibbles represent 2 decimal digits 0..9,0..9
            jrc     unit
            inc     dec_nbr
            jra     temp
unit        add     A,#10        ; We add 10 because we substracted it one more time.
            swap    dec_nbr      ; We put the number of tens in the MSB part.
            OR      A,dec_nbr    ; A contains the rest, we add it with tens.
```

Trick 4: How to deduce a parity bit of X content value? (returned in C)

```
            ld      Y,#8         ; Number of bits to shift.
            clr     A
loop        srl     X            ; Unsigned division of X by 2
            adc     A,#0         ; A is equal to the number of 1 in X.
            dec     Y
            jrne     loop        ; Continue until Y=0.
            srl     A            ;If Carry=1, X not even; if Carry=0, X even.
```

5.2 DBSET/DBRES, Dynamic Bit Set/Reset

Inputs: reg XThe byte address to manipulate
reg AThe bit position

Action: Set or Reset the bit number A (0..7) at byte address X

Output: No register modified

Variable definition:

```
            WORDS
            segment 'rom'

bittbl      dc.b    $01,$02,$04,$08,$10,$20,$40,$80
```

Program Listing:

```
            WORDS
            segment 'rom'

; Dynamic Bit set
.dbset      push    CC          ; Push CC into the stack to save its value.
            push    A           ; Push A into the stack to save its value.
            and     A,#$07      ; To have a bit number between 0 and 7.
            ld      Y,A
            ld      A,(bittbl,Y)  ; Point on the corresponding mask.
            or      A,(X)
            ld      (X),A       ; Put the result at X address.
            pop     A           ; Restore A from the stack.
            pop     CC          ; Restore CC from the stack.
            ret
; Dynamic bit reset
.dbres      push    CC
            push    A
            and     A,#$07
            ld      Y,A
            ld      A,(bittbl,Y)
            cpl     A
            and     A,(X)
            ld      (X),A
            pop     A
            pop     CC
            ret
```

5.3 JMPCALLTBL, Implementation of jump/call vector tables

Inputs: X BYTEThe selected function (1)
ptr WORDVector table address

(1)X = 00..7FJump Function[X]
X = 80..FFCall Function[X]

Action: Implement a function array (smallest and fastest way)

Variable definition:

```
      WORDS
      segment 'rom'
.ptr        DC.W   fn0,fn1,fn2,null,fn4
.fn0        inc    X
            ret
.fn1        srl    X
            ret
.fn2        sll    X
            ret
.fn4        dec    X
            ret
.null       ret
```

Program Listing

JPCALLFNX

```
            sll    X
            jrnc   jump          ; If no overflow by shifting left X, jump.
            call   jump
            nop
            ret
jump

            ld     A,({ptr+1},X)  ; Load of the address of the function
            push   A              ; to execute in A.
            ld     A,(ptr,X)
            push   A
            ret
```

5.4 Unsigned Word Multiplication

```
;                       Multiplication  A * B
;
; DATE :                21/11/96
; REVISION :            V01.00
;
; SOFTWARE DESCRIPTION :    This routine multiplies two 16 bit numbers
;                           A and B, the result is saved into four 8 bits
;                           registers (16x16= 32 bits)
;                           A and B >= 0.
;
; INPUT PARAMETERS :        OPERAND_A registers contain the number A.
;                           OPERAND_B registers contain the number B.
;
;
;
; OUTPUT PARAMETERS :       res registers contain the result.
;
;BYTE :                   63 bytes
;
;EXAMPLE :                   ;***** program *****
;                           ld A,#$F3
;                           ld operand_a,A
;                           ld A,#$D3
;                           ld {operand_a+1},A
;                           ld A,#$FC
;                           ld operand_b,A
;                           ld A,#$C3
;                           ld {operand_b+1},A
;                           CALL multiw
;                           - do...
;                           - do...
;                       ;***** subroutine *****
;                           . multiw
;                           END
;
```

```
.multiw
        push A              ; save Accumulator in stack
        push X              ; save X register  in stack

        ld X,operand_b      ; \
        ld A,operand_a      ;  | Multiplies MSB operand
        mul X,A ; /
        ld res,X ;and store in the 2 MSB result registers
        ld {res+1},A

        ld X,{operand_a+1}  ; \
        ld A,{operand_b+1}  ;  | Multiplies LSB operand
        mul X,A             ; /
        ld {res+2},X        ; and store in the 2 LSB result registers
        ld {res+3},A
        ld X,operand_a      ; \
        ld A,{operand_b+1}  ;  | Multiplies cross operands
        mul X,A             ; /
        add A,{res+2}       ; Add to previous result
        ld {res+2},A
        ld A,X
        adc A,{res+1}
        ld {res+1},A
        ld A,res
        adc A,#0
        ld res,A

        ld X,operand_b      ; \
        ld A,{operand_a+1}  ;  | Multiplies cross operands
        mul X,A             ; /
        add A,{res+2}       ; Add to previous result
        ld {res+2},A
        ld A,X
        adc A,{res+1}
        ld {res+1},A
        ld A,res
        adc A,#0
        ld res,A
        pop X               ; restore context before the CALL
        pop A               ; restore context before the CALL
        ret     ; and go back to main program
```

5.5 Unsigned Long Word by Word Division

```
;                   Long by Word division A/B
; DATE :                22/11/96
; REVISION :              V01.00
; ; SOFTWARE DESCRIPTION :    This routine divides one 32 bits number A by
;                   a 16-bit number B. The result is saved in two
;                   registers.
;                   A and B >= 0.
;
; INPUT PARAMETERS :       DIVIDEND registers contain the DIVIDEND (32 b).
;                   DIVISOR registers contain the DIVISOR (16 b).
;
; INTERNAL PARAMETERS :     TEMPQUOT registers contain the QUOTIENT
;                   temporary value (32 b).
;
; OUTPUT PARAMETERS :       QUOTIENT registers contain the result (16 b).
;                   As the result is not stored on 32 bits, this
;                   division is not valid in the general case.
;
;  BYTE :                94 bytes
;
; EXAMPLE :                ;***** program *****
;                       ld A,#$0E
;                       ld dividend,A
;                       ld A,#$DC
;                       ld {dividend+1},A
;                       ld A,#$BA
;                       ld {dividend+2},A
;                       ld A,#$98
;                       ld {dividend+3},A
;                       ld A,#$AB
;                       ld divisor,A
;                       ld A,#$CD
;                       ld {divisor+1},A
;                       CALL div_lxw
;                       - do...
;                       - do...
;                   ;***** subroutine *****
;                       . div_lxw
;                       END
```

```
.div_lxw
        push A          ; save Accumulator in stack
        push X          ; save X register  in stack

        ld X,#32        ; Initialization process
        ld A,#0         ; We use the load instruction
        ld quotient,A   ; which is faster than the
        ld {quotient+1},A     ; clear instruction for
        ld tempquot,A         ; multiple short datas.
        ld {tempquot+1},A     ; For a smaller code size
        ld {tempquot+2},A     ; you'd better use the clear
        ld {tempquot+3},A     ; instruction

.execute
        sla {dividend+3}  ;Shift left dividend with 32 leading Zeros
        rlc {dividend+2}
        rlc {dividend+1}
        rlc dividend
        rlc {tempquot+3}
        rlc {tempquot+2}
        rlc {tempquot+1}
        rlc tempquot
        sla {quotient+1}      ; The result cannot be greater than 16 bits
        rlc quotient          ; so we can shift left the quotient

        ld A,tempquot         ; Test is left dividend is greater or equal
        or A,{tempquot+1}     ; to the divisor
        jrne dividendlsgreater

        ld A,{tempquot+2}
        cp A,divisor
        jrugt dividendlsgreater
        jrult nosubstract

        ld A,{tempquot+3}
        cp A,{divisor+1}
        jrult nosubstract
.dividendlsgreater      ; Subtract divisor from left dividend
        ld A,{tempquot+3}
        sub A,{divisor+1}
        ld {tempquot+3},A
```

```
        ld A,{tempquot+2}
        sbc A,divisor
        ld {tempquot+2},A

        ld A,{tempquot+1}
        sbc A,#0
        ld {tempquot+1},A

        ld A,tempquot
        sbc A,#0
        ld tempquot,A

        inc {quotient+1}      ; The result cannot be greater than 16 bits
        jrne nosubstract      ; so we can increment the quotient
        inc quotient

.nosubtract
        dec X ; Decrement loop counter
        jrne execute          ; if X = 0 then exit else continue

        pop X                 ; restore context before the CALL
        pop A                 ; restore context before the CALL
        ret                   ; and go back to main program
```

5.6 Min./Max. Check

```
;                    CHECK MIN / MAX
;
; DATE :                22/11/96
; REVISION :              V01.00
;
; SOFTWARE DESCRIPTION : This routine tests if a 16 bit numbers  value
;                    is within a predefined range.
;
;                      MIN =< DATA =< MAX
;
; INPUT PARAMETERS :      DATA registers contain the number to test.
;                    MIN registers contain the minimum value.
;                    MAX registers contain the maximum value.
;[
; OUTPUT PARAMETERS :     The C flag is updated according to the result.
;                    C=1 means that the test has failed.
;
;  BYTE :                32 bytes
;
; EXAMPLE :              ;***** program *****
;           ld A,#$25
;           ld data,A
;           ld A,#$00
;           ld {data+1},A
;           ld A,#$00
;           ld min,A
;            ld A,#$C3
;           ld {min+1},A
;           ld A,#$CC
;           ld max,A
;            ld A,#$05
;           ld {max+1},A
;           CALL check_min_max
;           - do...
;           - do...
;           ;***** subroutine *****
;           .check_min_max
;           END
```

```
.check_min_max
        push A ; save Accumulator in stack
        push X ; save X register  in stack
        ld X,data ; get DATA MSB in X
        ld A,{data+1}          ; get DATA LSB in A

        cp X,max               ; Compare MSB with MAX
        jrugt out_of_range     ; if greater than exit
        jrne comp_min          ; else if equals compare LSB
        cp A,{max+1}
        jrugt out_of_range     ; LSB greater than exit
comp_min
        cp X,min               ; same thing with the LSB and the min value
        jrult out_of_range
        jrne in_range
        cp A,{min+1}
        jrult out_of_range
in_range
        rcf                ; Value in range so reset C flag
        jra exit           ; the value is within the two values
out_of_range
        scf                ; Value out of range so set C flag
exit
        pop X              ; restore context before the CALL
        pop A              ; restore context before the CALL
        ret                ; and go back to main program
```

5.7 Range Check

```
;                   CHECK RANGE for a WORD
; DATE :                22/11/96
; REVISION :              V01.00
; SOFTWARE DESCRIPTION :    This routine tests if a 16 bit numbers  value
;                   is within a predefined range
;                     MEDIAN - DELTA =< DATA  =< MEDIAN + DELTA
; INPUT PARAMETERS :      DATA registers contain the number to test.
;                   MEDIAN registers contain the median value.
;                   DELTA registers contain the delta value to add
;                   and subtract to the MEDIAN value.
; OUTPUT PARAMETERS :     The C flag is updated according to the result.
;                   C=1 means that the test has failed.
; NOTES:                This routine uses three previous sub routines.
;             check_min_max
;                   addw
;                   subw
;  BYTE :                66 bytes
;
; EXAMPLE : ;***** program *****
;           ld A,#$25
;           ld data,A
;           ld A,#$00
;           ld {data+1},A
;           ld A,#$00
;           ld delta,A
;           ld A,#$23
;           ld {delta+1},A
;           ld A,#$CC
;           ld median,A
;           ld A,#$05
;           ld {median+1},A
;           CALL check_range
;           - do...
;           - do...
;           ;***** subroutine *****
;           .addw
;           .subw
;           .check_min_max
;           .check_range
;           END
```

```
.check_range

            push A              ; save Accumulator in stack
            push X              ; save X register  in stack
            ld A,{median+1}       ; get median' LSB
            add A,{delta+1}       ; add delta' LSB
            ld {res_add+1},A      ; store LSB
            ld A,median           ; get median' MSB
            adc A,delta           ; add delta' MSB with LSB's carry
            ld res_add,A          ; store MSB


            jrnc no_ovfmax        ; test if an overflow occured
            ld A,#$FF           ; if yes then the MAX value is set to FFFFh
            ld max,A            ; (saturation)
            ld {max+1},A
no_ovfmax
            ld A,res_add          ; else there is no overflow, then
            ld max,A            ; the computed value is the MAX value to keep.
            ld A,{res_add+1}
            ld {max+1},A

            ld A,{median+1}       ; get median' LSB
            sub A,{delta+1}       ; sub delta' LSB
            ld {res_sub+1},A      ; store LSB
            ld A,median           ; get median' MSB
            sbc A,delta           ; sub delta' MSB with LSB's carry
            ld res_sub,A          ; store MSB
            jrnc no_ovfmin        ; test if an overflow occured
            clr A             ; if yes then the MIN value is set to 0000h
            ld min,A            ; (saturation)
            ld {min+1},A
```

```
no_ovfmin
        ld A,res_sub        ; else there is no overflow, then
        ld min,A            ; the computed value is the MIN value to keep.
        ld A,{res_sub+1}
        ld {min+1},A

        push A              ; save Accumulator in stack
        push X              ; save X register  in stack
        call check_min_max  ; Then we check if the value is within the range
                ; set by max and min.
        pop A               ; restore context before the CALL
        pop X               ; restore context before the CALL
        ret                 ; The result depends of the C flag.
```