PM0044
Programming manual

STM8 CPU programming manual

Introduction

The STM8 family of HCMOS microcontrollers is designed and built around an enhanced industry standard 8-bit core and a library of peripheral blocks, which include ROM, Flash, RAM, EEPROM, I/O, Serial Interfaces (SPI, USART, I2C,...), 16-bit Timers, A/D converters, comparators, power supervisors etc. These blocks may be assembled in various combinations in order to provide cost-effective solutions for application-specific products.

The STM8 family forms a part of the STMicroelectronics 8-bit MCU product line, which finds its place in a wide variety of applications such as automotive systems, remote controls, video monitors, car radio and numerous other consumer, industrial, telecom, and multimedia products.

Contents

List of tables

List of figures

1 STM8 architecture

The 8-bit STM8 Core is designed for high code efficiency. It contains 6 internal registers, 20 addressing modes and 80 instructions. The 6 internal registers include two 16-bit Index registers, an 8-bit Accumulator, a 24-bit Program Counter, a 16-bit Stack Pointer and an 8-bit 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 manipulation. These registers simplify branching routines and data/arrays modifications.

The 24-bit Program Counter is able to address up to 16-Mbyte of RAM, ROM or Flash memory. The 16-bit Stack Pointer provides access to a 64K-level Stack. The Core also includes a Condition Code register providing 7 Condition flags that indicate the result of the last instruction executed.

The 20 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 stack pointer relative addressing mode permits optimized C compiler stack model for local variables and parameter passing.

The 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, 8-bit by 8-bit multiplication, 16-bit by 8-bit and 16-bit by 16-bit division, 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 or direct memory-to-memory transfers.

The number of Interrupt vectors can vary up to 32, and the interrupt priority level may be managed by software providing hardware controlled nested capability. Some peripherals include Direct Memory Access (DMA) between serial interfaces and memory. Support for slow memories allows easy external code execution through serial or parallel interface (ROMLESS products for instance).

The STM8 has a high energy-efficient architecture, based on a Harvard architecture and pipelined execution. A 32-bit wide program memory bus allows most of the instructions to be fetched in 1 CPU cycle. Moreover, as the average instruction length is 2 bytes, this allows for a reduction in the power consumption by only accessing the program memory half of the time, on average. The pipelined execution allowed the execution time to be minimized, ensuring high system performance, when needed, together with the possibility to reduce the overall energy consumption, by using different power saving operating modes. Power-saving can be managed under program control by placing the device in SLOW, WAIT, SLOW-WAIT, ACTIVE-HALT or HALT mode (see product datasheet for more details).

Additional blocks

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

Boot ROM	Memory area containing the bootloader code
Flash	Flash-based devices
RAM	Sizes up to several Kbytes
Data EEPROM	Sizes up to several Kbytes. Erase/programming operations do not require additional external power sources.
Timers	Different versions based on 8/16-bit free running or autoreload timer/counter are available. They can be coupled with either input captures, output compares or PWM facilities. PWM functions can have software programmable duty cycle between 0% to 100% in up to 256/65536 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 12-bit resolution.
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 3/4 wire interface ideal for Master and Slave applications such as driving devices with input shift register (LCD driver, external memory,...).
USART	The USART is a fast synchronous/asynchronous interface which features both duplex transmission, NRZ format, programmable baud rates and standard error detection. The USART 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 by 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 bit lines.

1.1 STM8 development support

The STM8 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.

STM8 and ST7 CPUs are supported by a single toolchain allowing easy reuse and portability of the applications between product lines.

1.2 Enhanced STM8 features

- 16-Mbyte linear program memory space with 3 FAR instructions (CALLF, RETF, JPF)
- 16-Mbyte linear data memory space with 1 FAR instruction (LDF)
- Up to 32 24-bit interrupt vectors with optimized context save management
- 16-bit Stack Pointer (SP=SH:S) with stack manipulation instructions and addressing modes
- New register and memory access instructions (EXG, MOV)
- New arithmetic instructions: DIV 16/8 and DIVW 16/16
- New bit handling instructions (CCF, BCPL, BCCM)
- 2 x 16-bit index registers (X=XH:XL, Y=YH:YL). 8-bit data transfers address the low byte. The high-byte is not affected, with a reset value of 0. This allows the use of X/Y as 8-bit values.
- Fast interrupt handling through alternate register files (up to 4 contexts) with standard stack compatible mode (for real time OS kernels)
- 16-bit/8-bit stack operations (X, Y, A, CC stacking)
- 16-bit pointer direct update with 16-bit relative offset (ADDW/SUBW for X/Y/SP)
- 8-bit & 16-bit arithmetic and signed arithmetic support

2 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), operation code.
mem	memory location
imm	immediate value
off	offset
ptr	pointer
pos	position
byte	a byte
word	16-bit value
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
ExtB	Extended byte
MS	Most Significant byte of a 16-bit value (MSB)
LS	Least Significant byte of a 16-bit value (LSB)
A	Accumulator register
X	16-bit X Index register
Y	16-bit Y Index register
reg	A, XL or YL register (1-byte LS part of X/Y), XH or YH (1-byte MS part of X/Y)
ndx	index register, either X or Y
PC	24-bit Program Counter register
SP	16-bit Stack Pointer
S	Stack Pointer LSB
CC	Condition Code register

3 STM8 core description

3.1 Introduction

The CPU has a full 8-bit architecture, with 16-bit operations on index registers (for address computation). Six internal registers allow efficient 8-bit data manipulation. The CPU is able to execute 80 basic instructions. It features 20 addressing modes and can address 6 internal registers and 16 Mbytes of memory/peripheral registers.

3.2 CPU registers

The 6 CPU registers are shown in the programming model in *Figure 1*. Following an interrupt, the register context is saved. The context is saved by pushing registers onto the stack in the order shown in *Figure 2.* They are popped from the stack in the reverse order.

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.

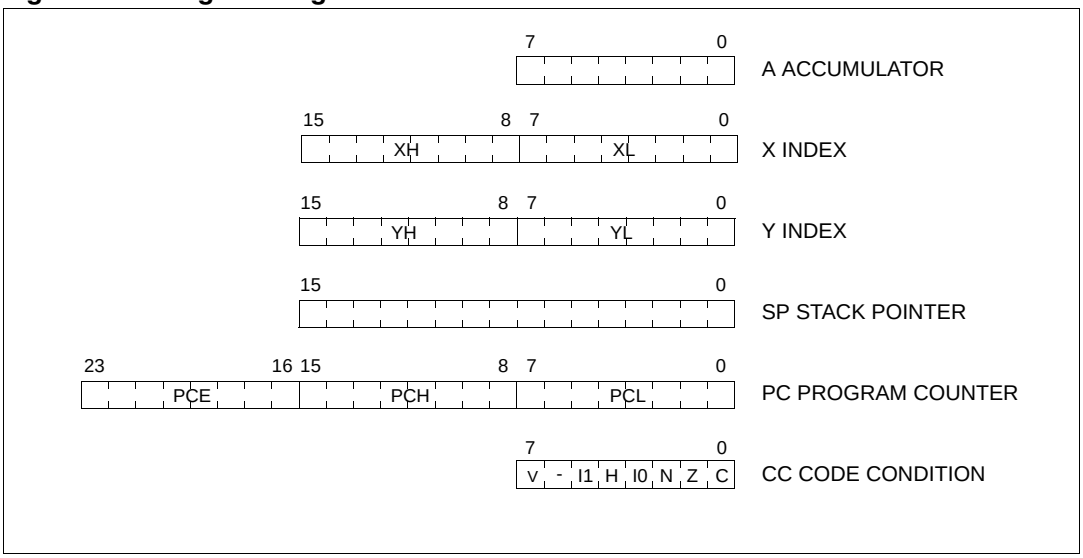
Index registers (X and Y)

These 16-bit registers are used to create effective addresses or as temporary storage area for data manipulations. In most of the cases, the cross assembler generates a PRECODE instruction (PRE) to indicate that the following instruction refers to the Y register. Both X and Y are automatically saved on interrupt routine branch.

Program Counter (PC)

The program counter is a 24-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 STM8 core can access up to 16-Mbytes of memory.

Figure 1. Programming model



Stack Pointer (SP)

The stack pointer is a 16-bit register. It contains the address of the next free location of the stack. Depending on the product, the most significant bits can be forced to a preset value.

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.

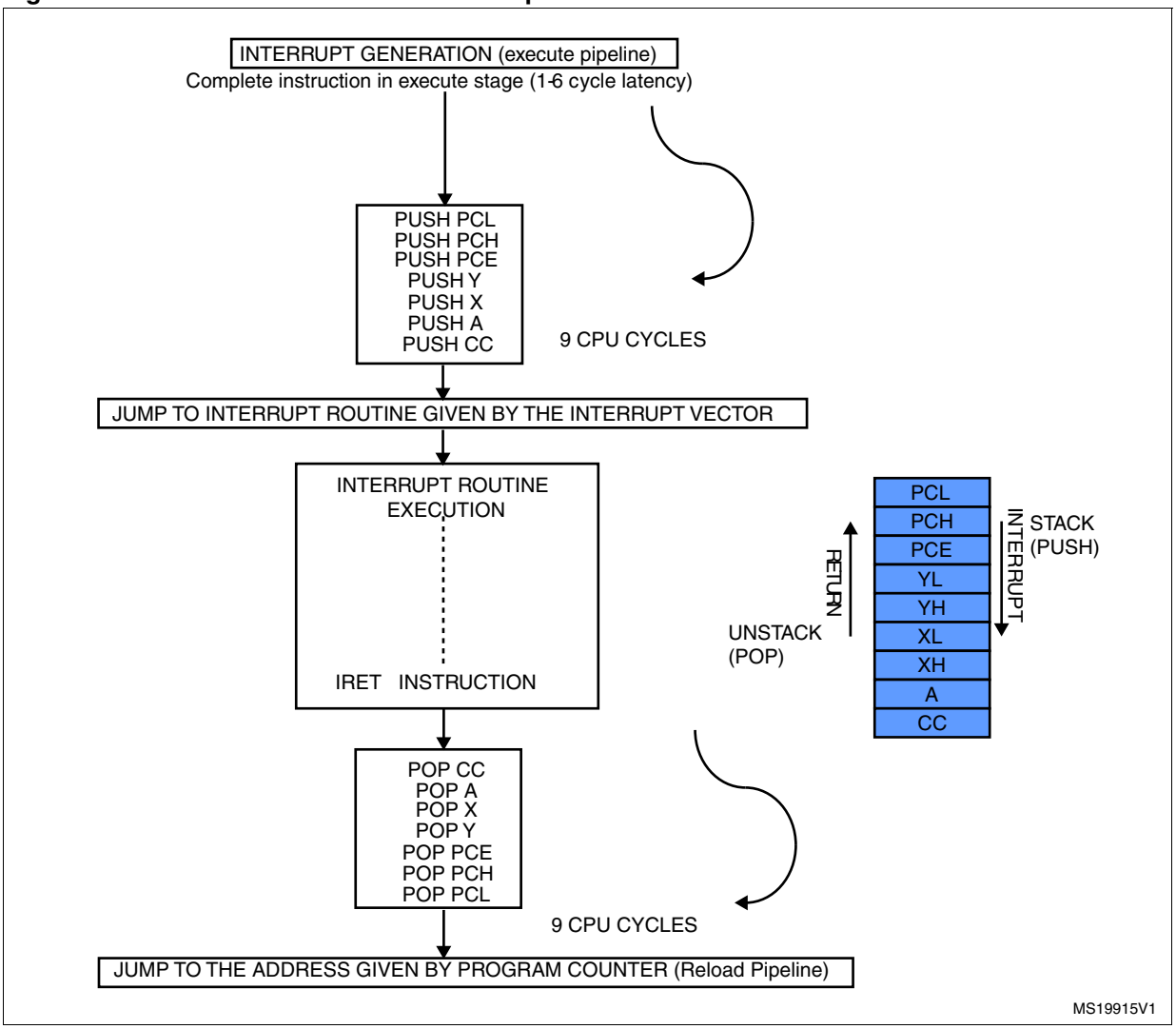
After an MCU reset the Stack Pointer is set to its upper limit value. It is then decremented after data has been pushed onto the stack and incremented after data is popped from the stack. When the lower limit is exceeded, the stack pointer wraps around to the stack upper limit. The previously stored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

A subroutine call occupies two or three locations.

When an interrupt occurs, the CPU registers (CC, X, Y, A, PC) are pushed onto the stack. This operation takes 9 CPU cycles and uses 9 bytes in RAM.

Note: The WFI/HALT instructions save the context in advance. If an interrupt occurs while the CPU is in one of these modes, the latency is reduced.

Figure 2. Context save/restore for interrupts



Global configuration register (CFG_GCR)

The global configuration register is a memory mapped register. It controls the configuration of the processor. It contains the AL control bit:

AL: Activation level

If the AL bit is 0 (main), the IRET will cause the context to be retrieved from stack and the main program will continue after the WFI instruction.

If the AL bit is 1 (interrupt only active), the IRET will cause the CPU to go back to WFI/HALT mode without restoring the context.

This bit is used to control the low power modes of the MCU. In a very low power application, the MCU spends most of the time in WFI/HALT mode and is woken up (through interrupts) at specific moments in order to execute a specific task. Some of these recurring tasks are short enough to be treated directly in an ISR, rather than going back to the main program. In this case, by programming the AL bit to 1 before going to low power (by executing WFI/HALT instruction), the run time/ISR execution is reduced due to the fact that the register context is not saved/restored each time.

Condition Code register (CC)

The Condition Code register is a 8-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.

- **V: Overflow**

  When set, V indicates that an overflow occurred during the last signed arithmetic operation, on the MSB operation result bit. See INC, INCW, DEC, DECW, NEG, NEGW, ADD, ADC, SUB, SUBW, SBC, CP, CPW instructions.

- **I1: Interrupt mask level 1**

  The I1 flag works in conjunction with the I0 flag to define the current interruptability level as shown in the following table. These flags can be set and cleared by software through the RIM, SIM, HALT, WFI, IRET, TRAP and POP instructions and are automatically set by hardware when entering an interrupt service routine.

Table 1. Interruptability levels

Interruptability	Priority	I1	I0
Interruptable Main	Lowest ↕ Highest	1	0
Interruptable Level 1		0	1
Interruptable Level 2		0	0
Non Interruptable		1	1

- **H: Half carry bit**

  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.

  For ADDW, SUBW it is set when a carry occurs from bit 7 to 8, allowing to implement byte arithmetic on 16-bit index registers.

- **I0: Interrupt mask level 0**

  See Flag I1

- **N: Negative**

  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).

- **Z: Zero**

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

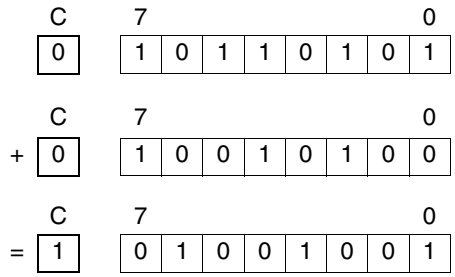
- **C: Carry**

  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 (bit 7 for 8-bit result/destination or bit 15 for 16-bit result). 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 or complemented by software using SCF, RCF, CCF instructions.

  Example: Addition

  $B5 + $94 = "C" + $49 = $149



The results of each instruction on the Condition Code register are shown by tables in *Section 7: STM8 instruction set*. The following table is an example:

V	I1	H	I0	N	Z	C
V	0		0	N	Z	1

where

Nothing = Flag not affected

Flag name = Flag affected

0 = Flag cleared

1 = Flag set

4 STM8 memory interface

4.1 Program space

The program space is 16-Mbyte and linear. To distinguish the 1, 2 and 3 byte wide addressing modes, naming has been defined as shown in *Figure 3*:

- "Page" [0xXXXX00 to 0xXXXXFF]: 256-byte wide memory space with the same two most significant address bytes (XXXX defines the page number).
- "Section" [0xXX0000 to 0xXXFFFF]: 64-Kbyte wide memory space with the same most significant address byte (XX defines the section number).

The reset and interrupt vector table are placed at address 0x8000 for the STM8 family. (Note: the base address may be different for later implementations.) The table has 32 4-byte entries: RESET, Trap, NMI and up to 29 normal user interrupts. Each entry consists of the reserved op-code 0x82, followed by a 24-bit value: PCE, PCH, PCL address of the respective Interrupt Service Routine. The main program and ISRs can be mapped anywhere in the 16 Mbyte memory space.

CALL/CALLR and RET must be used only in the same section. The effective address for the CALL/RET is used as an offset to the current PCE register value. For the JP, the effective address 16 or 17-bit (for indexed addressing) long, is added to the current PCE value. In order to reach any address in the program space, the JPF jump and CALLF call instructions are provided with a three byte extended addressing mode while the RETF pops also three bytes from the stack.

As the memory space is linear, sections can be crossed by two CPU actions: next instruction byte fetch (PC+1), relative jumps and, in some cases, by JP (for indexed addressing mode).

Note: For safe memory usage, a function which crosses sections MUST:
- be called by a CALLF
- include only far instructions for code operation (CALLF & JPF)

All label pointers are located in section 0 (JP [ptr.w] example: ptr.w is located in section 0 and the jump address in current section)

Any illegal op-code read from the program space triggers a MCU reset.

4.2 Data space

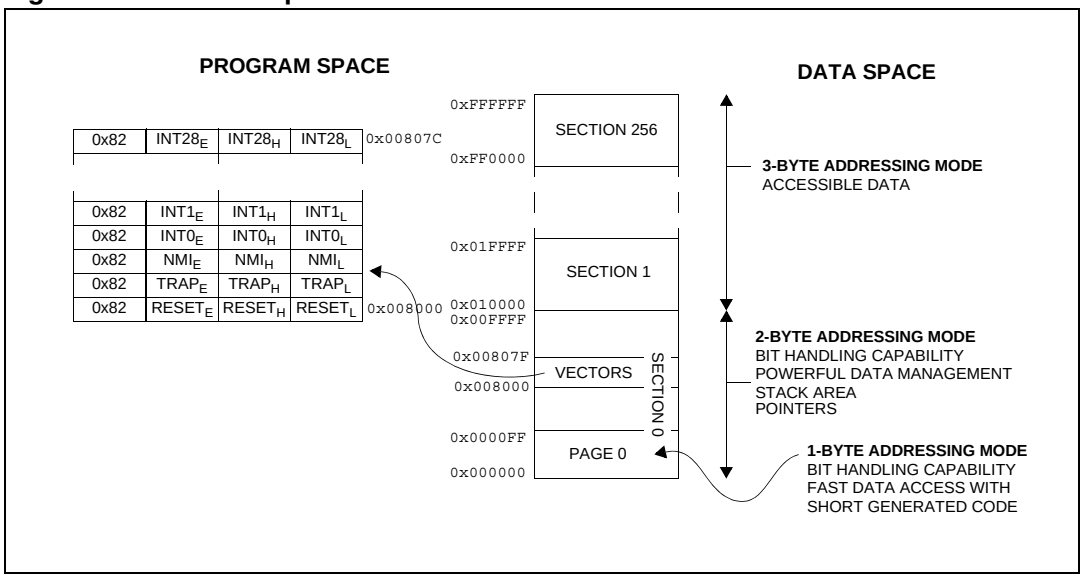
The data space is 16-Mbyte and linear. As the stack must be located in section 0 and as data access outside section 0/1 can be managed only with LDF instructions, frequently used data should be located in section 0 to get the optimum code efficiency.

All data pointers are located in section 0 only.

Indexed addressing (with 16-bit index registers and long offset) allows data access over section 0 and 1.

All the peripherals are memory mapped in the data space.

Figure 3. Address spaces



4.3 Memory interface architecture

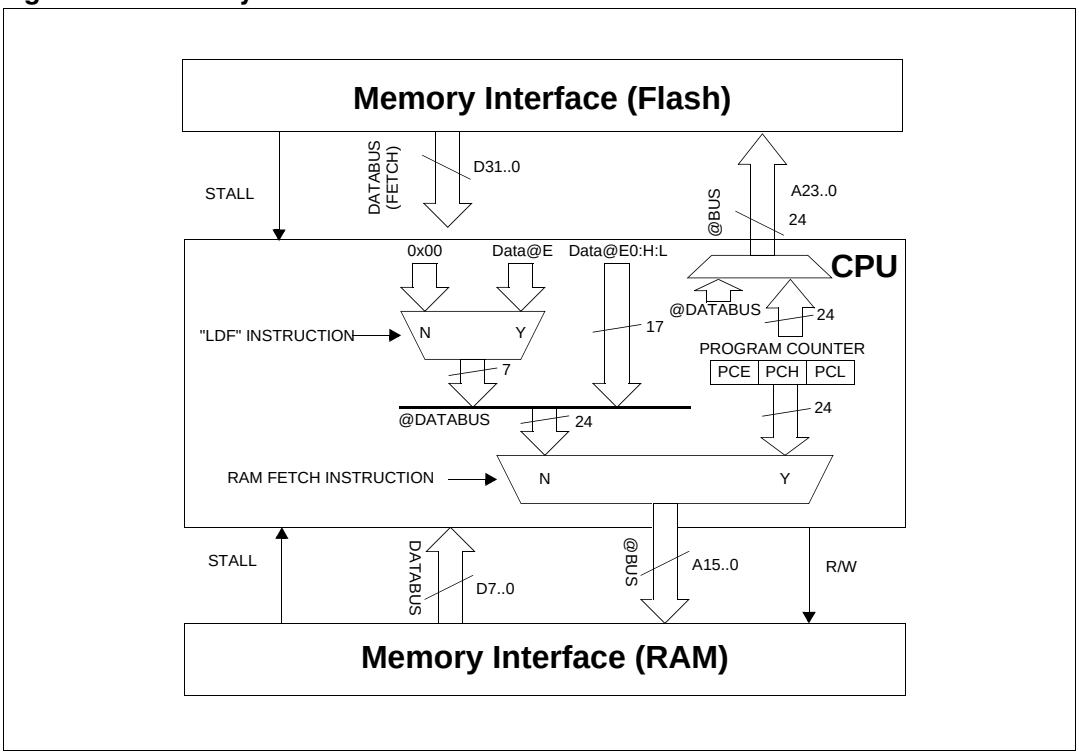
The STM8 uses a Harvard architecture, with separate program and data memory buses. However, the logical address space is unified, all memories sharing the same 16-Mbytes space, non-overlapped. The memory interfaces are shown in *Figure 4*. It consists of two buses: address, data, read/write control signal (R/W) and memory acknowledge signal (STALL).

The STALL acknowledge signal makes the CPU compatible with slow serial or parallel memory interfaces. When the memory interface is slow the CPU waits the memory acknowledge before executing the instruction. So in such a case, the instruction CPU cycle time is prolonged compare to the value given in this manual.

The program memory bus is 32-bit wide, allowing the fetch of most of the instructions in one cycle.

As the address space is unified, the architecture allows data to be stored also in the Flash memory and program to be fetched also from RAM (data bus). In this later case the performance is impacted, besides the fact that data and fetch operation share the same bus, the instructions will be fetched one byte at a time, thus taking longer (1 cycle /byte).

Figure 4. Memory Interface Architecture



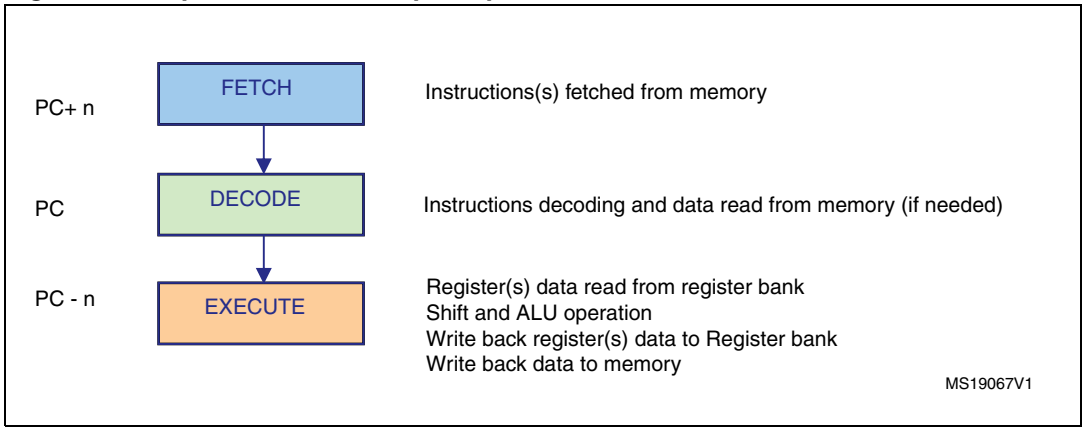
5 Pipelined execution

The STM8 family uses a 3-stage pipeline to increase the speed of the flow of instructions sent to the processor. Pipelined execution allows several operations to be performed simultaneously, rather than serially:

- Fetch
- Decode and address
- Execute

The Program Counter (PC) points always to the instruction in decode stage as shown in *Figure 5*.

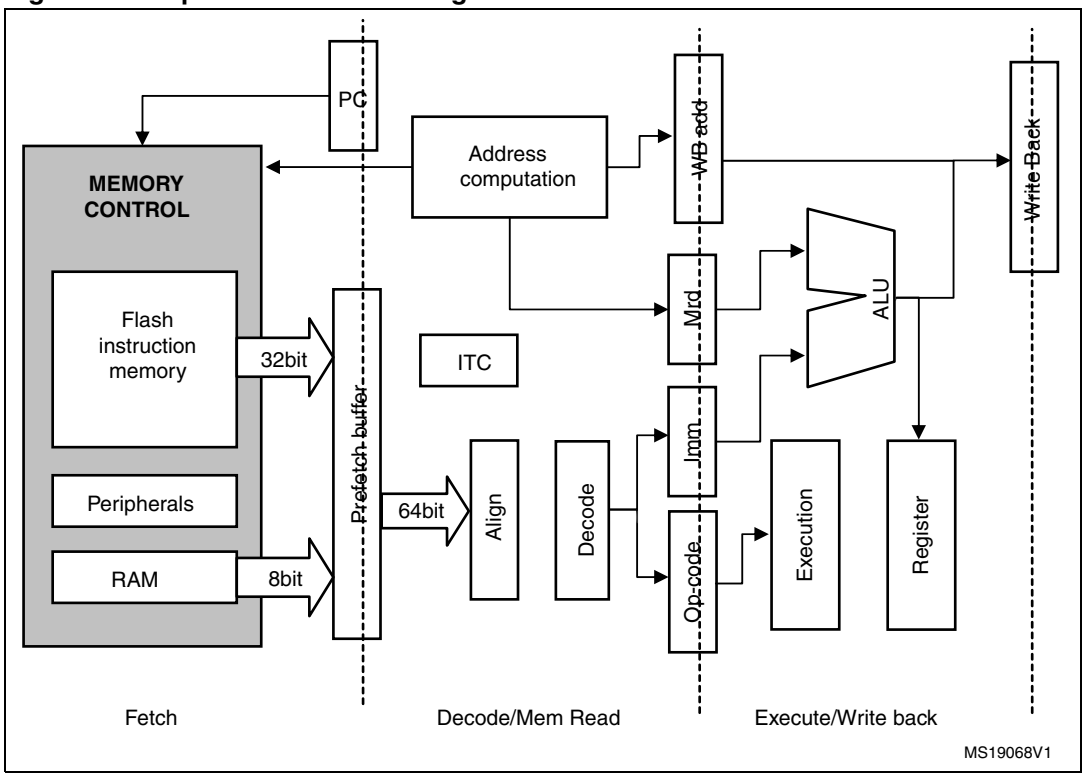
Figure 5. Pipelined execution principle



5.1 Description of pipelined execution stages

Figure 6 and *Section 5.1.1*, *Section 5.1.2*, and *Section 5.1.3* provide a detailed description of each stage of the pipeline execution.

Figure 6. Pipelined execution stages



5.1.1 Fetch stage

The first pipeline stage includes a 64-bit fetch buffer and a 32-bit prefetch buffer, totalling 3 words named F_1, F_2 and F_3. This buffer structure allows any instruction code (up to 5 bytes) to be available for decoding immediately after F_1 (and F_2 when needed) is/are loaded.

The instruction access from Flash Program memory is 32-bit wide and it is performed from an aligned address i.e. 0xXXX0, 0xXXX4, 0xXXX8, or 0xXXXC.

Unlike the decode and execute stages that are performed at every cycle, the fetch stage accesses the program memory only when needed, and stops memory access when the buffer is full. This allows reducing the core power consumption,

Reading program from RAM is similar to reading program from ROM. However, since the RAM data bus is 8-bit wide, 4 consecutive read operations have to be performed to load one F_X word, thus resulting in RAM execution being slower than Flash execution.

5.1.2 Decoding and addressing stage

The decoding stage includes an instruction alignment unit. The alignment unit uses the 64-bit input from the fetch unit and feeds an instruction (from 1 to 5 bytes depending on the instruction) to the decoding unit.

The instruction code consists of 2 parts (see examples in *Table 2*):

- The op-code itself (1 or 2 bytes)
- and a data/address part (0 to 3 bytes).

The op-code is decoded in this stage. When present, the instruction address is used for address computation, whilst the immediate operand is forwarded to the execution stage.

Table 2. Data/address decoding examples

Instruction	Syntax	Op-code	Data/address
Register to register move	LD A, XH	0x95	-
Register load	LD A,($12,SP)	0x7B	0x12
Register store	LD ($12,SP),A	0x6B	0x12
Data load / store with extended address	LDF A,($123456,Y)	0x90 AF	0x12 34 56

Long/unaligned instructions

For long instructions (i.e. 5-bytes instructions), the fetch may need 2 program memory accesses to be completed. In this case, the decoding stage (after decoding the op-code part), is stalled waiting for the fetch stage to complete the 2nd fetch.

In case of shorter instructions, this may also happen when they cross a 32-bit boundary.

Indirect addressing

For indirect addressing, the CPU is stalled in this stage to read the pointer from the data memory (i.e. RAM). The number of cycles during which the CPU is stalled depends on the pointer size (short, long or extended addressing mode).

5.1.3 Execution stage

In the execution stage, the operation is executed and the result is stored in the accumulator, index register or RAM.

5.2 Data memory conflicts

3 types of operations perform accesses to the data memory:

- Effective address computation in case of indirect addressing
- Data read: source operand
- Data write: destination for store or read-modify-write operations

In case of simultaneous accesses to the same memory area both in execution stage (write) and decoding stage (read), the decode stage is stalled till the execution stage releases the resource.

5.3 Pipelined execution examples

A few pipelined execution examples are reported below. The numbers of cycles for the decoding and execution stages correspond to the minimum number of cycles needed by the instruction itself. In some cases, depending on the instruction sequence, the cycle taken could be more than that number.

5.4 Conventions

Although the decode and/or execute stage of some instructions may take a different number of cycles, a simplified convention providing a good match with reality, has been used in this section:

- The decode stage of each instruction takes one cycle only
- The execution stage takes a number of cycles equal to

$$C_y = DecCy + ExeCy - 1$$

Where

C_y is the number of execution cycles. In case of decode and execute cycles, It corresponds to the minimum number of cycles needed by the instruction itself, and does not take into account the impact of the instruction sequence.

DecCy is the exact number of decode cycles.

ExeCy is the exact number of execute cycles.

The decode stage of the next instruction starts during the last execution cycle. In instructions performing pipeline flush, the convention is that, in case the branch is taken, the next fetch are performed during the last instruction execution cycle.

The exact number of cycles (see *Table 3*) and the number of cycles obtained using this convention (see *Table 4*) are identical.

Table 3. Example with exact number of cycles

Address	Instruction	Decode cycles	Execute cycles	lgth	Time (cycle)													
					1	2	3	4	5	6	7	8	9	10	11	12	13	14
0xC000	LDW X, [$50.w]	4	1	3	F_1	D	D	D	D	E								
0xC003	ADDW X, #20	2	2	3		F_2	D	D	D	D	D	E	E					
0xC006	LD A, [$30].w	3	1	3			F_3			D	D	D	D	D	D	E		
0xC009	….																	

Table 4. Example with conventional number of cycles

Address	Instruction	Decode cycles	Execute cycles	lgth	Time (cycle)													
					1	2	3	4	5	6	7	8	9	10	11	12	13	14
0xC000	LDW X, [$50.w]	4	3	3	F_1	D	E	E	E	E								
0xC003	ADDW X, #20	3	3	3		F_2	D	D	D	D	E	E	E					
0xC006	LD A, [$30].w	3	3	3			F_3			D	D	D	D	E	E	E		
0xC009	….																	

Table 5. Legend

Symbol/Color	Definition
F	Fetch
D	Decode stalled
D	Decode
E	Execute

5.4.1 Optimized pipeline example – execution from Flash Program memory

In the example shown in *Table 6*, the code is stored in the Flash Program memory (32-bit bus). As a result, 3 cycles are needed to fill the 96-bit prefetch buffer. At each cycle, one word is loaded and stored in F_1, F_2 and F_3. The next fetch operation can start only when all the instructions contained in one of the F_x word are decoded. In fact, at cycle 9, the last instruction contained in F_3 (SWAP A) is decoded, and a fetch operation can start to fill F_3 word.

Table 6. Optimized pipeline example - execution from Flash

Add.	Instruction	Decod. cycles	Exec. cycles	lgth	Cycle													
					1	2	3	4	5	6	7	8	9	10	11	12	13	14
0xC000	NEG A	1	1	1	F_1	D	E											
0xC001	XOR A, $10	1	1	2			D	E										
0xC003	LD A, #20	1	1	2		F_2		D	E									
0xC005	SUB A,$1000	1	1	3					D	E								
0xC008	INC A	1	1	1			F_3			D	E							
0xC009	LD XL, A	1	1	1							D	E						
0xC00A	SRL A	1	1	1								D	E					
0xC00B	SWAP A	1	1	1									D	E				
0xC00C	SLA $15	1	1	2				F_1						D	E			
0xC00E	CP A,#$FE	1	1	2											D	E		
0xC010	MOV $100, #11	1	1	4					F_2							D	E	
0xC014	MOV $101, #22	1	1	4									F_3				D	E

Table 7. Legend

Symbol/Color	Definition
F	Fetch
D	Decode
E	Execute

5.4.2 Optimize pipeline example – execution from RAM

In the example shown in *Table 8*, the RAM is accessed through an 8-bit bus. As a result, 12 cycles are required to fill the 96-bit pre-fetch buffer. Every 4 cycles, one word is loaded and stored in F_x. The decoding of the first word instruction can start only when the F_x word is filled. This occurs for example till the 4th cycle, and the first instruction (*NEG A*) can be decoded only at the 5th cycle.

In case of read/write access to the RAM, the fetch is stalled. This occurs during the 6th cycle since RAM address 10 is read during the decode stage of *XOR A, $10*.

Table 8. Optimize pipeline example – execution from RAM

Add.	Instruction	Decode cycles	Execute cycles	lgth	Cycle																				
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0xC000	NEG A	1	1	1	F_{1_1}	D	D	D	D	E															
0xC001	XOR A, $10	1	1	2		F_{1_2}	F_{1_3}			D	E														
0xC003	LD A, #20	1	1	2				F_{1_4}	F_{2_1}		D	D	D	D	E											
0xC005	SUB A,$1000	1	1	3						FS	F_{2_2}	F_{2_3}	F_{2_4}		D	E									
0xC008	INC A	1	1	1										F_{3_1}		D	D	D	D	E					
0xC009	LD XL, A	1	1	1											FS	F_{3_2}				D	E				
0xC00A	SRL A	1	1	1													F_{3_3}				D	E			
0xC00B	SWAP A	1	1	1														F_{3_4}				D	E		
0xC00C	SLA $15	1	1	2															F_{1_1}	F_{1_2}			D	E	
0xC00E	CP A,#$FE	1	1	2																	F_{1_3}	F_{1_4}		D	E

Table 9. Legend

Symbol/Color	Definition
F	Fetch
FS	Fetch stalled
D	Decode
D	Decode stalled
E	Execute

5.4.3 Pipeline with Call/Jump

In the example shown in *Table 10*, a branch is taken after the JP/CALL instruction, and the fetched instruction(s) are lost (flush). New instructions must be fetched. 3 fetch sequences are required to refill the pre-fetch buffer. The fetch start depends on the instruction being executed.

For a JP instruction, the fetch can start during the first cycle of the "dummy" execution.

For the CALL instruction, it starts after the last cycle of the CALL execution.

Table 10. Example of pipeline with Call/Jump

Add.	Instruction	Decode cycles	Execute cycles	lgth	Cycle										
					1	2	3	4	5	6	7	8	9	10	11
0xC000	INC A	1	1	1	F_1	D	E								
0xC001	JP label	1	1	3			D	E							
0xC004	LDW X,[$5432.w]	X	X	4		F_2	Flush								
0xD010	label: NEG A	1	1	1				F_1	D	E					
0xD011	CALL label2	1	2	3						D	E	E			
0xD014	LDW X,[$5432.w]	X	X	4					F_2			Flush			
0xD018	LDW X,[$7895.w]	X	X	4						F_3	FS				
0xE030	label2: INCW X	1	1	1									F_1	D	E

Table 11. Legend

Symbol/Color	Definition
F	Fetch
FS	Fetch stalled
D	Decode
E	Execute

5.4.4 Pipeline stalled

The decode stage can be stalled when the execution lasts more than one cycle.

The flush is due to the branch. Fetching the branch address is performed during the second execution cycle of the BTJF instruction.

The Decode operation can also be stalled when the memory target is modified during the previous instruction. In the example given in *Table 12*, the INCW Y instruction writes the X

register during the first execution cycle. As a result, in this cycle, the next instruction (LD A,(X)) cannot be decoded since it reads the X register.

Table 12. Example of stalled pipeline

Address	Instruction	Decode cycles	Execute cycles	lgth	Time (cycles)											
					1	2	3	4	7	8	9	10	11	12	13	14
0xC000	SUB SP, #20	1	1	2	F_1	D	E									
0xC002	LD A, #20	1	1	2			D	E								
0xC004	BTJT 0x10, #5, to	1	2	5		F_2	F_3	D	E	E						
0xC009	INC A	1	1	1					D	D	E					
0xC00A	BTJF 0x20, #3, to	1	2	5				F_1			D	E	E			
0xC00F	NOP	X	X	1								Flush				
0xC010	LDW X,[$5432.w]	X	X	4					F_2							
0xC014	LDW X,[$1234.w]	X	X	4						F_3						
0xD020	to: INCW Y	1	1	2									F_1	D	E	
0xD023	LD A,(X)	1	1	2										D	D	E

Table 13. Legend

Symbol/Color	Definition
F	Fetch
D	Decode stalled
D	Decode
E	Execute

5.4.5 Pipeline with 1 wait state

In the example given in *Table 14*, performing the fetch takes 2 cycles, and there is no overlap between the 2 fetch cycles.

If the instruction is decoded/executed during the last 2 fetch cycles, then the wait state is transparent compared to the no-wait state execution.

Table 14. Pipeline with 1 wait state

Address	Instruction	Decode cycles	Execute cycles	lgth	Time (cycle)									
					1	2	3	4	5	6	7	8	9	10
0xC000	NEG A	1	1	1	MS	F_1	D	E						
0xC001	DEC ($10, X)	1	1	3				D	E					
0xC004	LDW X, #20	1	1	3			MS	F_2	D	E	E			
0xC007	LD (X), A	1	1	1						D	D	E		
0xC008	INC A	1	1	1					MS	F_3		D	E	
0xC009	NEG ($5A, Y)	1	1	1									D	E

Table 15. Legend

Symbol/Color	Definition
F	Fetch
D	Decode stalled
D	Decode
MS	Memory stalled
E	Execute

6 STM8 addressing modes

The STM8 core features 18 different addressing modes which can be classified in 8 main groups:

Table 16. STM8 core addressing modes

Addressing mode groups	Example
Inherent	NOP
Immediate	LD A,#$55
Direct	LD A,$55
Indexed	LD A,($55,X)
SP Indexed	LD A,($55,SP)
Indirect	LD A,([$55],X)
Relative	JRNE loop
Bit operation	BSET byte,#5

The STM8 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 three sub-modes called extended, long and short:

- The extended addressing mode ("e") can reach any byte in the 16-Mbyte addressing space, but the instruction size is bigger than the short and long addressing mode. Moreover, the number of instructions with this addressing mode (far) is limited (CALLF, RETF, JPF and LDF)
- The long addressing mode ("w") is the most powerful for program management, when the program is executed in the same section (same PCE value). The long addressing mode is optimized for data management in the first 64-Kbyte addressing space (from 0x000000 to 0x00FFFF) with a complete set of instructions, but the instruction size is bigger than the short addressing mode.
- The short addressing mode ("b") is less powerful because it can only access the page zero (from 0x000000 to 0x0000FF), but the instruction size is more compact.

Table 17. STM8 addressing mode overview

Mode			Syntax	Destination address	Pointer address	Pointer size
Inherent			NOP			
Immediate			LD A,#$55			
Short	Direct		LD A,$10	000000..0000FF		
Long	Direct		LD A,$1000	000000..00FFFF		
Extended	Direct		LDF A,$100000	000000..FFFFFF		
No Offset	Direct	Indexed	LD A,(X)	000000..00FFFF		
Short	Direct	Indexed	LD A,($10,X)	000000..0100FE		

Table 17. STM8 addressing mode overview (continued)

Mode			Syntax	Destination address	Pointer address	Pointer size
Short	Direct	SP Indexed	LD A,($10,SP)	00..(FF+Stacktop)		
Long	Direct	Indexed	LD A,($1000,X)	000000..01FFFE		
Extended	Direct	Indexed	LDF A,($100000,X)	000000..FFFFFF		
Short Pointer Long	Indirect		LD A,[$10.w]	000000..00FFFF	000000..0000FF	2
Long Pointer Long	indirect		LD A,[$1000.w]	000000..00FFFF	000000..00FFFF	2
Long Pointer Extended	indirect		LDF A,[$1000.e]	000000..FFFFFF	000000..00FFFF	3
Short Pointer Long	Indirect	Indexed	LD A,([$10.w],X)	000000..01FFFE	000000..0000FF	2
Long Pointer Long	Indirect	Indexed (X only)	LD A,([$1000.w],X)	000000..01FFFE	000000..00FFFF	2
Long Pointer Extended	Indirect	Indexed	LDF A,([$1000.e],X)	000000..FFFFFF	000000..00FFFF	3
Relative	Direct		JRNE loop	PC+127/-128		
Bit	Long Direct		BSET $1000,#7	000000..00FFFF		
Bit	Long Direct	Relative	BTJT $1000,#7,skip	000000..00FFFF PC+127/-128		

6.1 Inherent addressing mode

All related instructions are 1 or 2 byte. The op-code fully specifies all required information for the CPU to process the operation.

Table 18. Inherent addressing instructions

Instructions	Functions
NOP	No operation
TRAP	S/W Interrupt
WFI, WFE	Wait For Interrupt / Event (Low Power Mode)
HALT	Halt Oscillator (Lowest Power Mode)
RET	Sub-routine Return
RETF	Far Sub-routine Return
IRET	Interrupt Sub-routine Return
SIM	Set Interrupt Mask
RIM	Reset Interrupt Mask
SCF	Set Carry Flag
RCF	Reset Carry Flag
RVF	Reset Overflow Flag
CCF	Complement Carry Flag
LD, LDW	Load
CLR, CLRW	Clear
PUSH, POP, PUSHW, POPW	Push/Pop to/from the stack
INC, DEC, INCW, DECW	Increment/Decrement
TNZ, TNZW	Test Negative or Zero
CPL, NEG, CPLW, NEGW	1's or 2's Complement
MUL	Byte Multiplication
DIV, DIVW	Division
EXG, EXGW	Exchange
SLA, SLL, SRL, SRA, RLC, RRC, SLAW, SLLW, SRLW, SRAW, RLCW, RRCW	Shift and Rotate Operations
SWAP, SWAPW	Swap Nibbles/Bytes

Example:

```
1000  98  RCF  ; Reset carry flag
1001  9D  NOP  ; No operation
1002  9F  LD A,X; Transfer X register content into accumulator
1004  88  PUSH A; Push accumulator content onto the stack
```

6.2 Immediate addressing mode

The data byte required for the operation, follows the op-code.

Table 19. Immediate addressing instructions

Instructions	Functions
LD, MOV, LDW	Load and move operation
CP, CPW	Compare
BCP	Bit Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC, ADDW, SUBW	Arithmetic Operations
PUSH	Stack Operations

These are two byte instructions, one for the op-code and the other one for the immediate data byte.

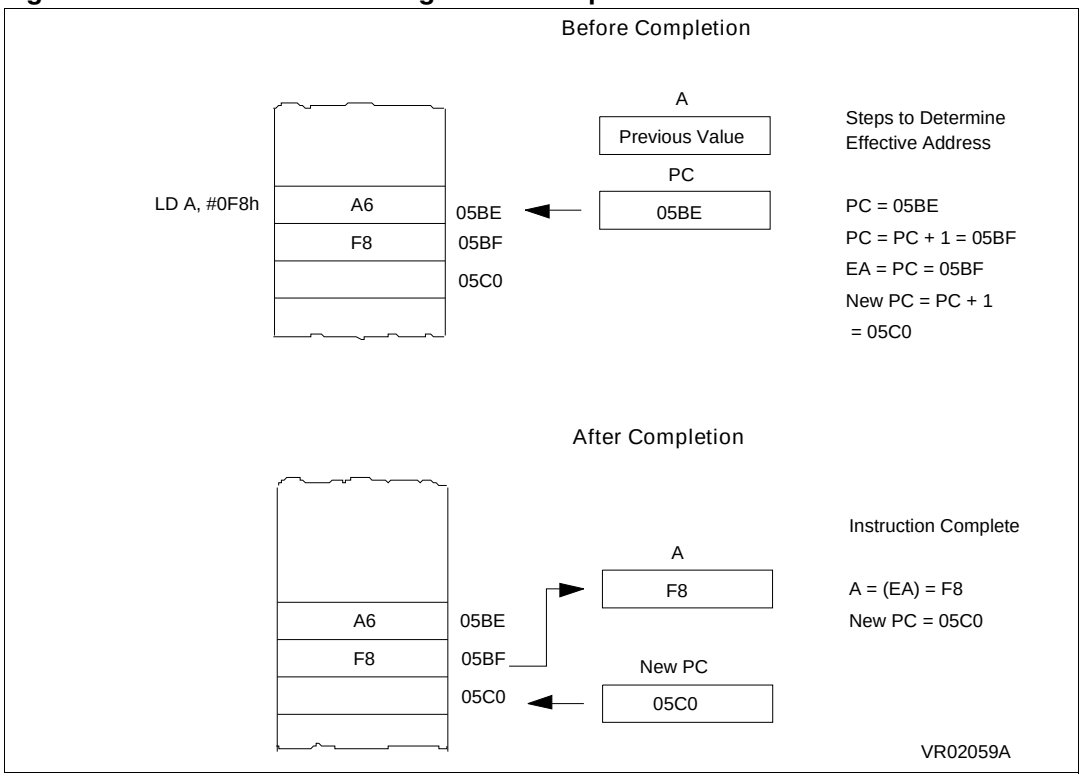
Example:

```
05BA     AEFF      LD          X,#$FF
05BC     A355       CP          X,#$55
05BE     A6F8       LD          A,#$F8
```

Action:

```
Load X = $FF

Compare (X, $55)
A = $F8
```

Figure 7. Immediate addressing mode example



6.3 Direct addressing mode (Short, Long, Extended)

Table 20. Overview of Direct addressing mode instructions

Addressing mode		Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Short	Direct	shortmem	(shortmem)	op + 1	Byte	00..FF
Long	Direct	longmem	(longmem)	op + 1..2	Word	0000..FFFF
Extended	Direct	extmem	(extmem)	op + 1..3	Ext word	000000..FFFFFF

The data byte required for the operation is found by its memory address, which follows the op-code.

Direct addressing mode is made of three sub-modes:

Table 21. Available Long and Short Direct addressing mode instructions

Instructions	Functions
LD, LDW	Load
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC, ADDW, SUBW	Arithmetic Addition/Subtraction operations
BCP	Bit Compare

Table 21. Available Long and Short Direct addressing mode instructions

Instructions	Functions
MOV	Move
CLR	Clear
INC, DEC	Increment/Decrement
TNZ	Test Negative or Zero
CPL, NEG	1's or 2's Complement
SLA, SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles
CALL, JP	Call or Jump subroutine

Table 22. Available Extended Direct addressing mode instructions

Instructions	Function
CALLF, JPF	Call or Jump FAR subroutine
LDF	Far load

Table 23. Available Long Direct addressing mode instructions

Instructions	Function
EXG	Exchange
PUSH, POP	Stack operation

6.3.1 Short Direct addressing mode

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

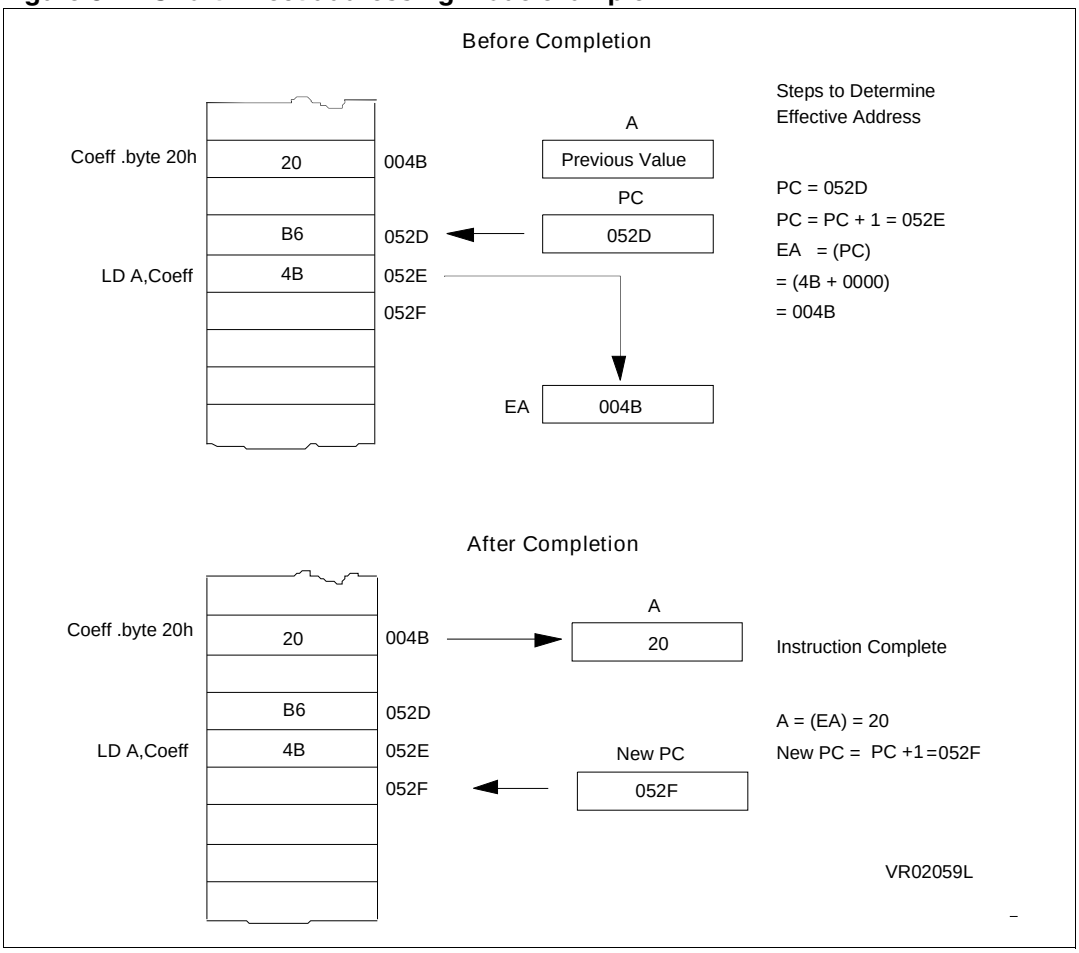
Example:

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

Action:

```
A = (coeff) = ($4B) = $20
```

Figure 8. Short Direct addressing mode example



6.3.2 Long Direct addressing mode

The address is a word, thus allowing 0000 to FFFF addressing space, but requires 2 bytes after the op-code.

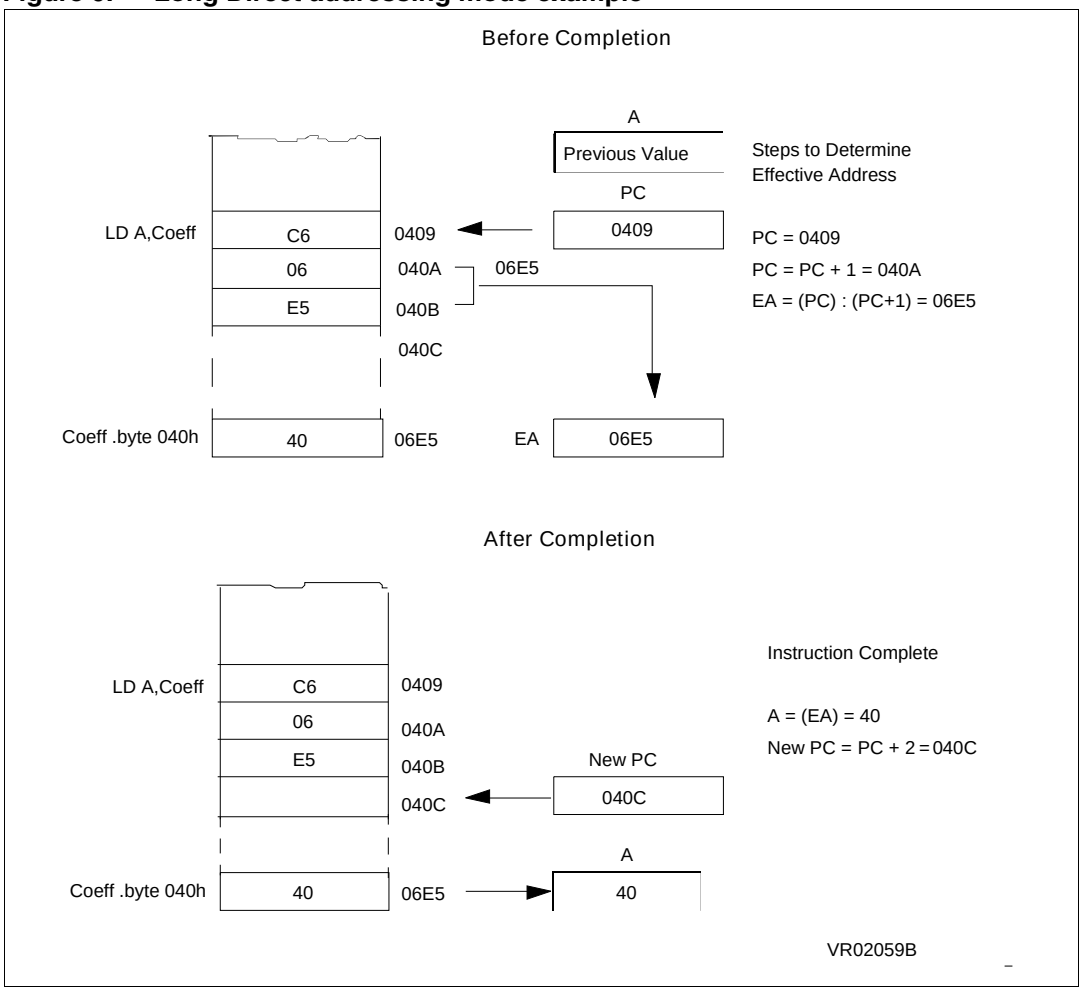
Example:

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

Action:

```
A = (coeff) = ($06E5) = $40
```

Figure 9. Long Direct addressing mode example



6.3.3 Extended Direct addressing mode (only for CALLF and JPF)

The address is an extended word, thus allowing 000000 to FFFFFF addressing space, but requires 3 bytes after the op-code.

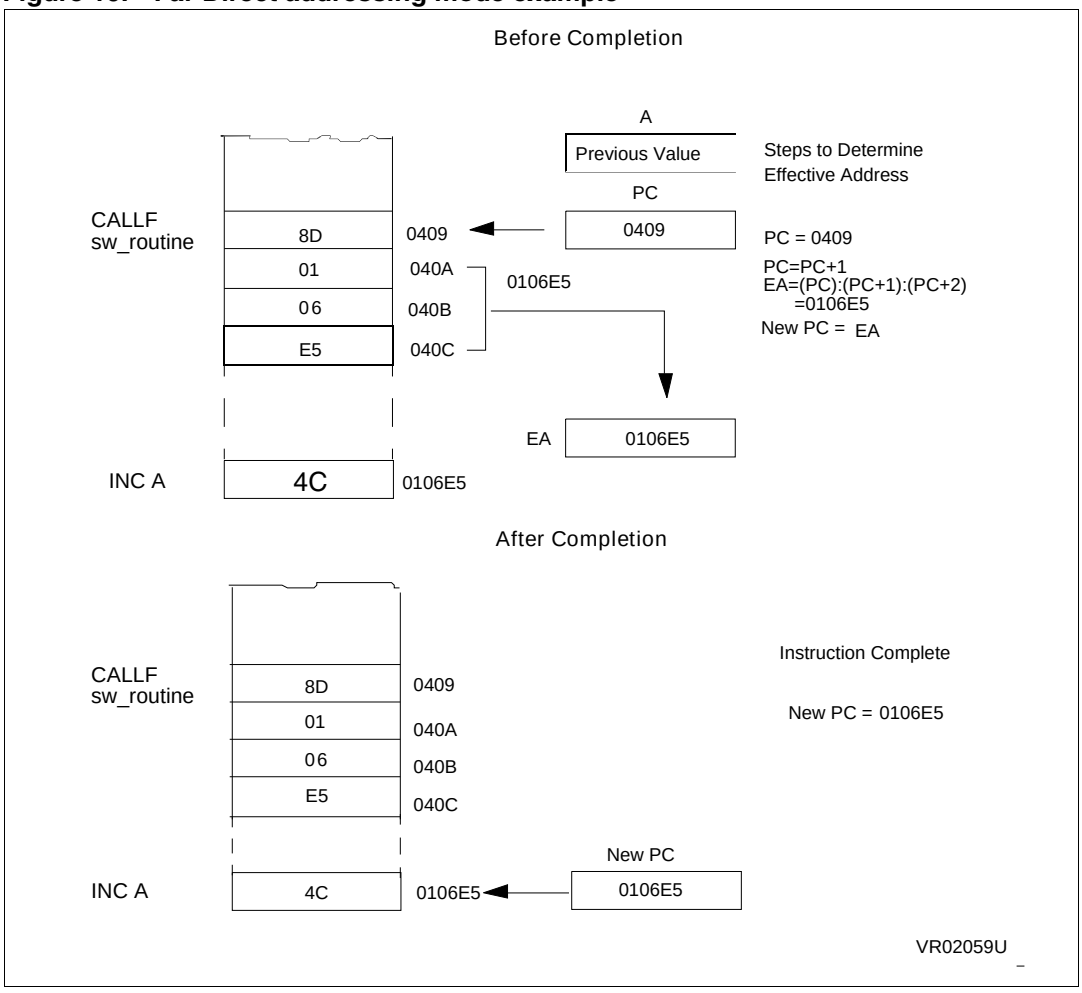
Example:

```
000409    8D0106E5                  CALLF sw_routine
0106E5    4C          sw_routine    INC     A
```

Action:

```
PC = $0106E5
```

Figure 10. Far Direct addressing mode example



6.4 Indexed addressing mode (No Offset, Short, SP, Long, Extended)

Table 24. Overview Indexed addressing mode instructions

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
No offset	Direct	Indexed	(ndx)	(ndx)	---	---	00..FFFF
Short	Direct	Indexed	(shortoff,ndx)	(ptr + ndx)	op + 1	Byte	00..100FE
Stack Pointer	Direct	Indexed	(shortoff,SP)	(ptr + SP)	op + 1	Byte	00..(FF+stacktop)
Long	Direct	Indexed	(longoff,ndx)	(ptr.w + ndx)	op + 1..2	Word	000000..01FFFE
Extended	Direct	Indexed	(extoff,ndx)	(ptr.e + ndx)	op + 1..3	Ext Word	000000..FFFFFF

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

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

Table 25. No Offset, Long, Short and SP Indexed instructions

Instructions	Functions
LD, LDW	Load
CLR	Clear
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC, ADDW, SUBW	Arithmetic Addition/Subtraction operations
INC, DEC	Increment/Decrement
TNZ	Test Negative or Zero
CPL, NEG	1's or 2's Complement
SLA, SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles

Table 26. No Offset, Long, Short Indexed Instructions

Instructions	Functions
CALL, JP	Call or Jump subroutine

Table 27. Extended Indexed Instructions only

Instructions	Functions
LDF	Far Load

6.4.1 No Offset Indexed addressing mode

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

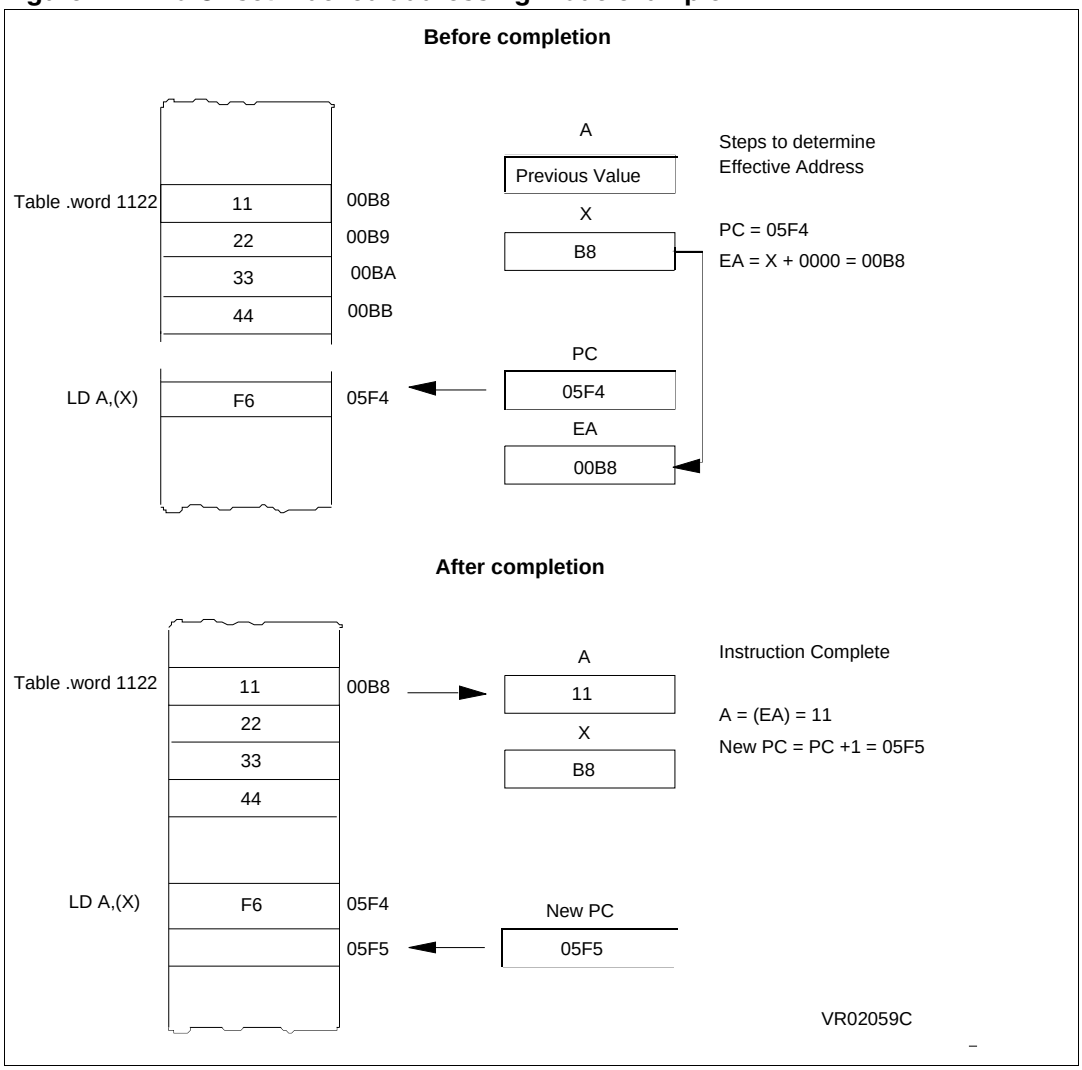
Example:

```
00B8      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 11. No Offset Indexed addressing mode example



6.4.2 Short Indexed addressing mode

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

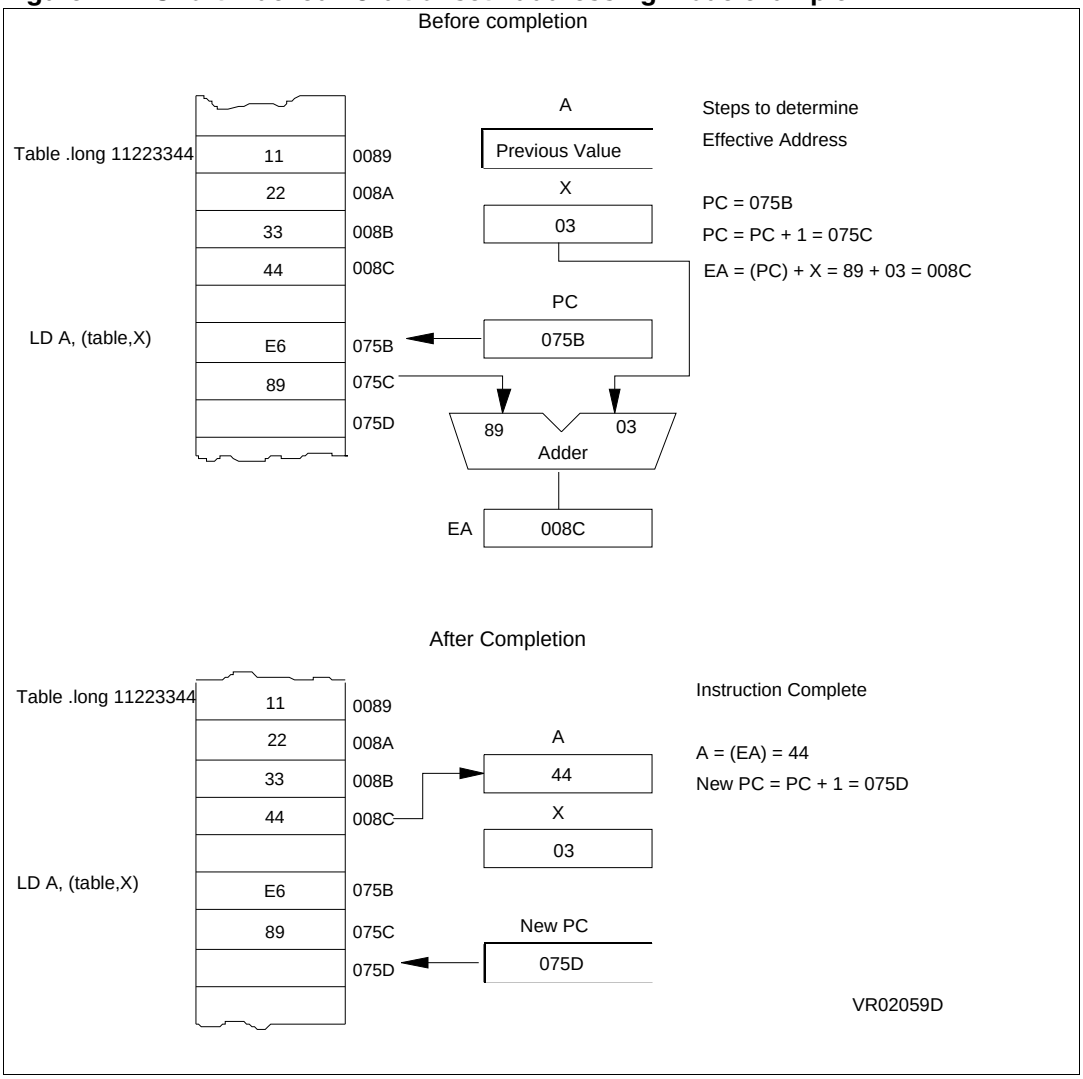
Example:

```
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 12. Short Indexed - 8-bit offset - addressing mode example



6.4.3 SP Indexed addressing mode

The offset is a byte, thus require only one byte after the op-code, but only allow 00..(FF + stack top) addressing space.

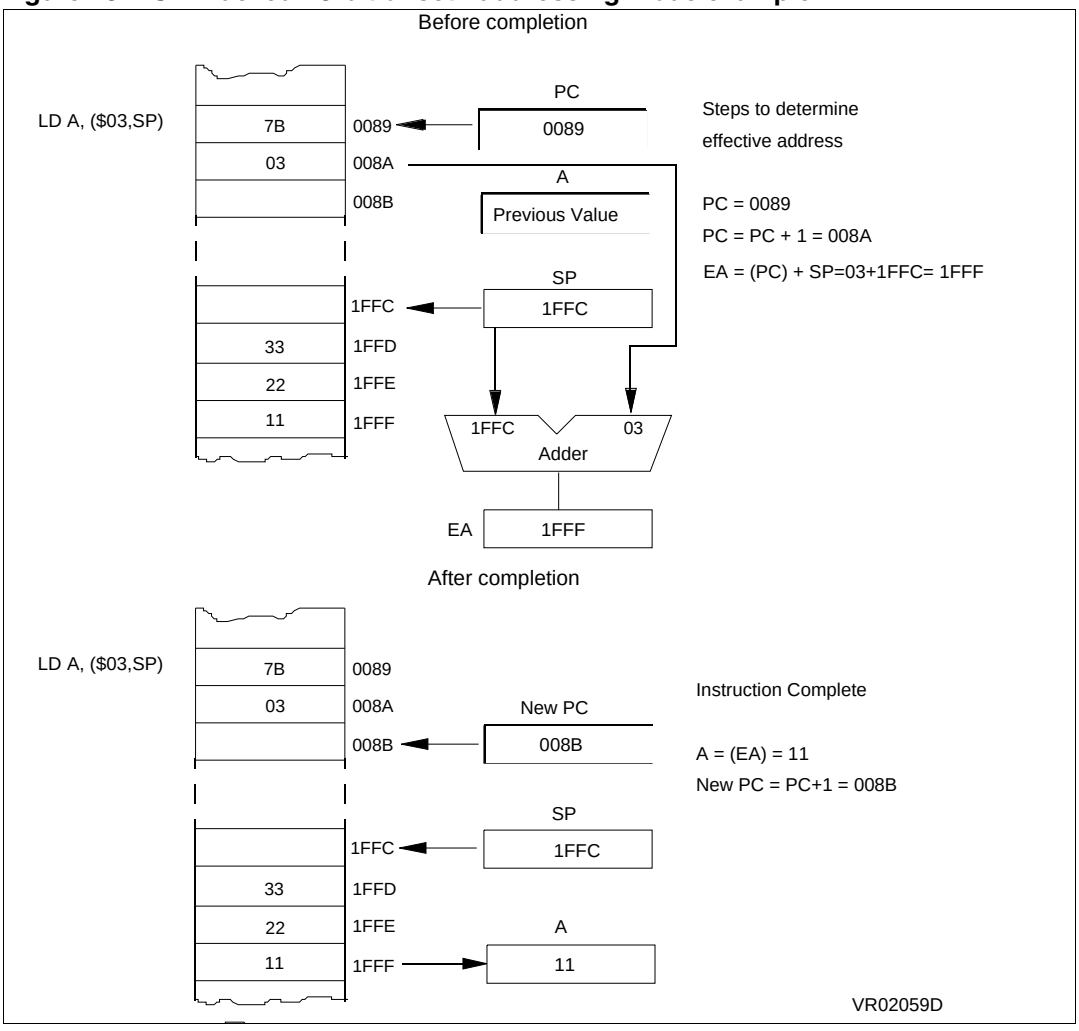
Example:

```
0086  4B11     PUSH #$11
0087  4B22     PUSH #$22
0088  4B33     PUSH #$33
0089  7B03     LD A,($03,SP)
```

Action:

```
A = ($03, SP) = ($03, $1FFC) = ($1FFF) = $11
```

Figure 13. SP Indexed - 8-bit offset - addressing mode example



6.4.4 Long Indexed addressing mode

The offset is a word, thus allowing up to 128 KB addressing space, but requires 2 bytes after the op-code.

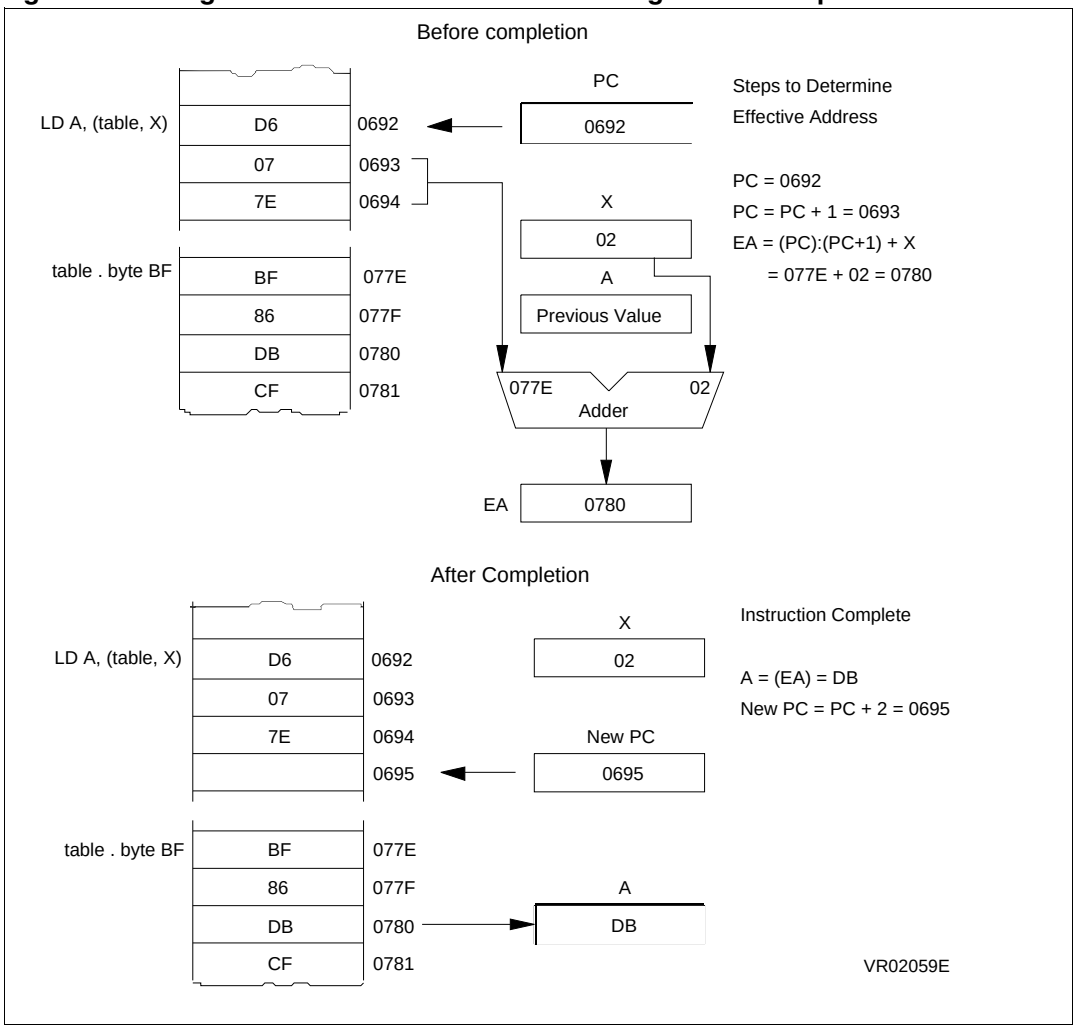
Example:

```
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 14. Long Indexed - 16-bit offset - addressing mode example



6.4.5 Extended Indexed (only LDF instruction)

The offset is an extended word, thus allowing 16Mbyte addressing space (from 000000 to FFFFFF), but requires 3 bytes after the op-code.

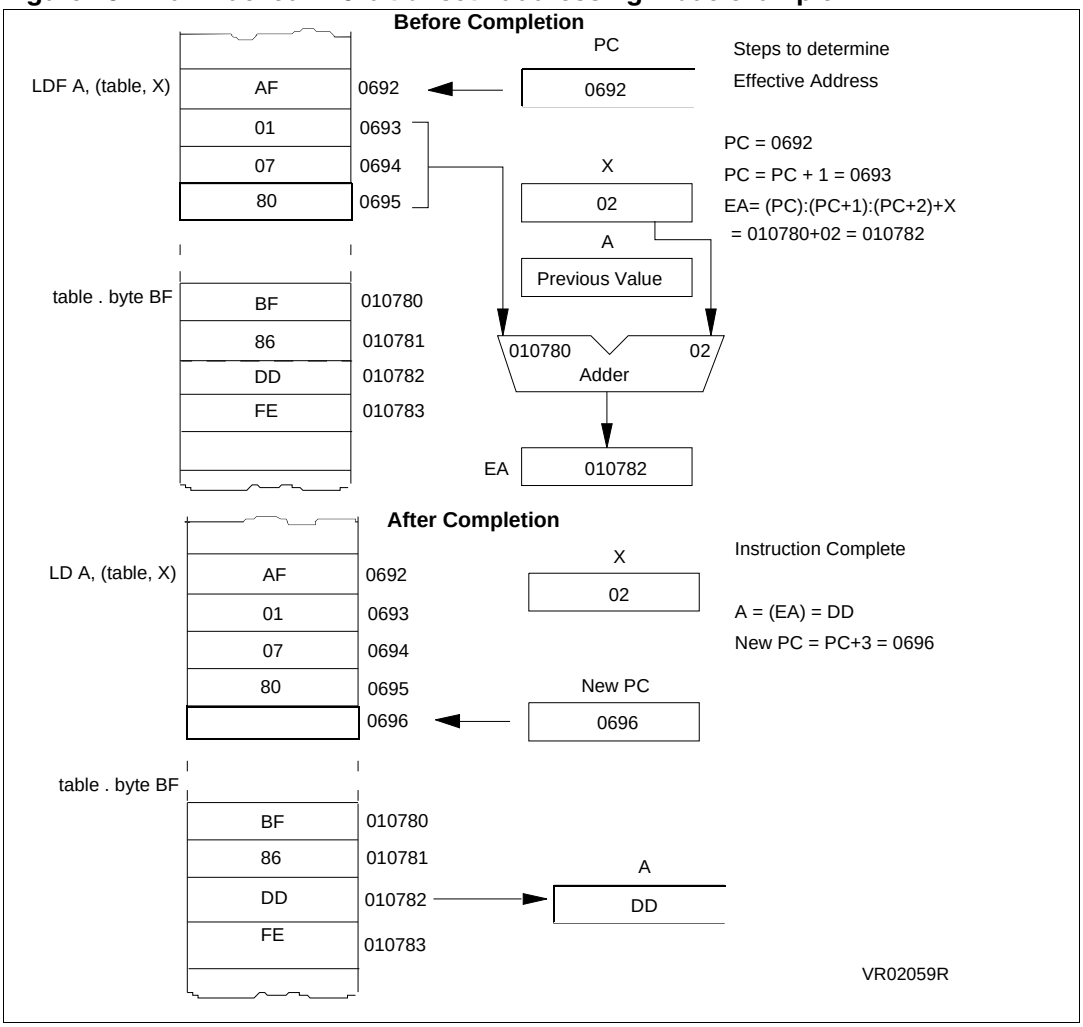
Example:

```
0690    AE02              LD    X,#2
0692    AF010780          LDF A,(table,X)
010780 BF       table     dc.b    $BF
       86                 dc.b    $86
       DDFE               dc.w   $DDFE
```

Action:

```
X = 2, A = (table, X) = ($010780,X) = ($010780+2)) = ($010782) = $DD
```

Figure 15. Far Indexed - 16-bit offset - addressing mode example



6.5 Indirect (Short Pointer Long, Long Pointer Long)

Table 28. Overview of Indirect addressing instructions

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

The data byte required for 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 three sub-modes:

Table 29. Available Long Pointer Long and Short Pointer Long Indirect Instructions

Instructions	Functions
LD, LDW	Load
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC	Arithmetic Addition/Subtraction operations
BCP	Bit Compare
CALL, JP	Call or Jump subroutine

Table 30. Available Long Pointer Long Indirect Instructions

Instructions	Functions
CLR	Clear
TNZ	Test Negative or Zero
CPL, NEG	1's or 2's Complement
SLA, SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles
INC, DEC	Increment/Decrement

6.6 Short Pointer Indirect Long addressing mode

The pointer address is a byte, the pointer size is a word, thus allowing up to 128 KB addressing space, and requires 1 byte after the op-code.

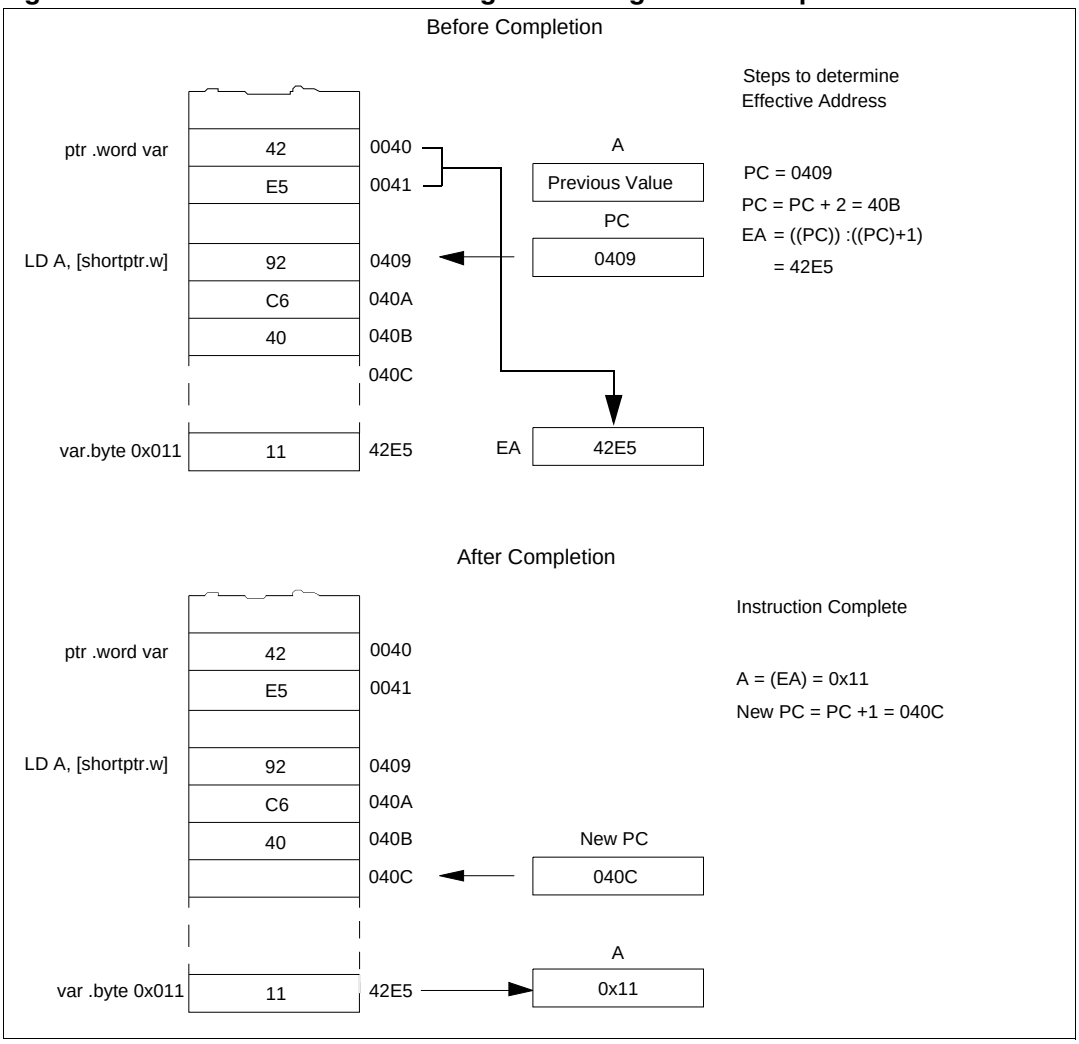
Example:

```
0040    42E5    ptr   dc.w   var
0409    92C640        LD     A,[shortptr.w]
42E5    11      var   dc.b   $11
```

Action:

```
A = [shortptr.w] = ((shortptr.w)) = (($40.w)) = ($42E5) =
$11
```

Figure 16. Short Pointer Indirect Long addressing mode example



6.7 Long Pointer Indirect Long addressing mode

The pointer address is a word, the pointer size is a word, thus allowing 64 KB addressing space, and requires 2 bytes after the op-code.

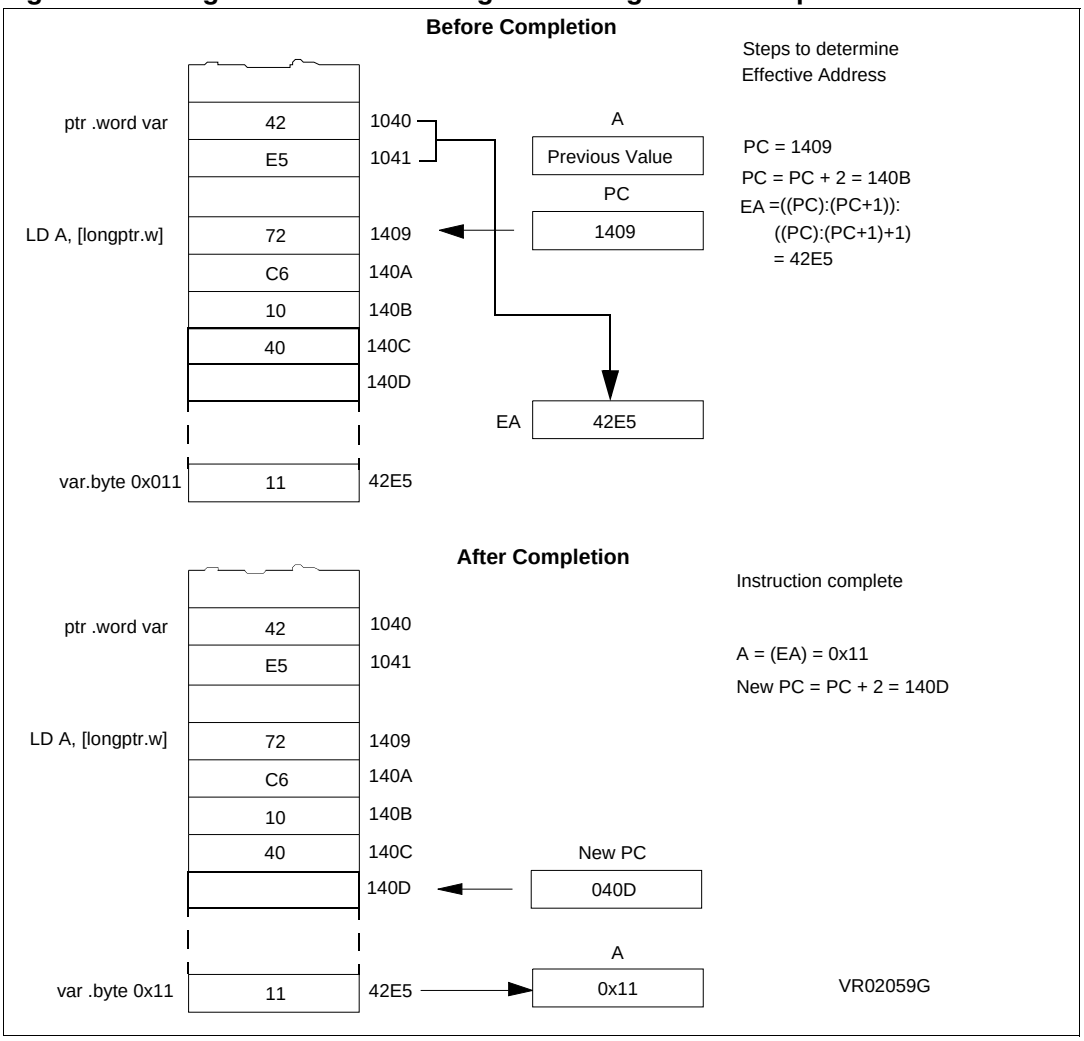
Example:

```
1040    42E5          ptr     dc.w    var
1409    72C61040              LD      A,[longptr.w]
42E5    11            var     dc.b    $11
```

Action:

```
A = [longptr.w] = ((longptr.w)) = (($1040.w)) = ($42E5) = $11
```

Figure 17. Long Pointer Indirect Long addressing mode example



6.8 Indirect Indexed (Short Pointer Long, Long Pointer Long, Long Pointer Extended) addressing mode

Table 31. Overview of Indirect indexed instructions

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Short Pointer Long	Indirect	Indexed	([shortptr.w],ndx)	((shortptr.w) + ndx)	00..FF	Word	000000.01FFFE
Long Pointer Long	Indirect	Indexed	([longptr.w],ndx)	([longptr.w] +ndx)	00..FFFF	Word	000000.01FFFE
Long Pointer Extended	Indirect	Indexed	([longptr.e],ndx)	([longptr.e] +ndx)	00..FFFF	Extword	000000.FFFFFE

This is a combination of indirect and indexed addressing mode. The data byte required for 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 four sub-modes:

Table 32. Available Long Pointer Long and Short Pointer Long Indirect Indexed instructions

Instructions	Functions
LD, LDW	Load
CP	Compare
AND, OR, XOR	Logical Operations
ADC, ADD, SUB, SBC	Arithmetic Addition/Subtraction operations
BCP	Bit Compare
CALL, JP	Call or Jump subroutine

Table 33. Available Long Pointer Long Indirect Indexed instructions

Instructions	Functions
CLR	Clear
TNZ	Test Negative or Zero
CPL, NEG	1's or 2's Complement
SLA,SLL, SRL, SRA, RLC, RRC	Shift and Rotate Operations
SWAP	Swap Nibbles
INC, DEC	Increment/Decrement

Table 34. Long Pointer Extended Indirect Indexed instructions instruction

Instructions	Functions
LDF	Far load

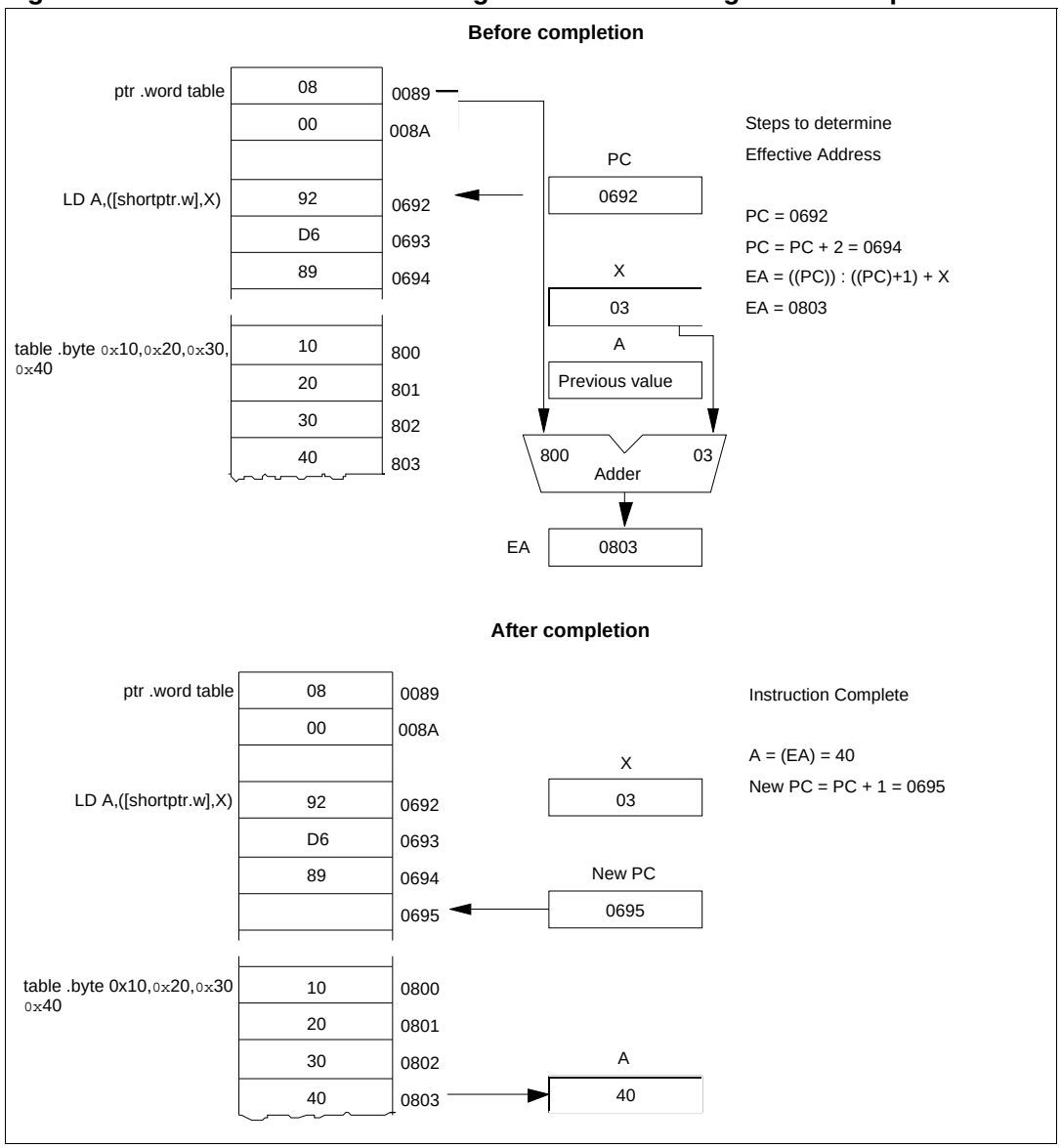
6.9 Short Pointer Indirect Long Indexed addressing mode

The pointer address is a byte, the pointer size is a word, thus allowing up to 128 KB addressing space, and requires 1 byte after the op-code.

Example:

```
0089   0800     ptr   dc.w   table


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


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

Figure 18. Short Pointer Indirect Long Indexed addressing mode example



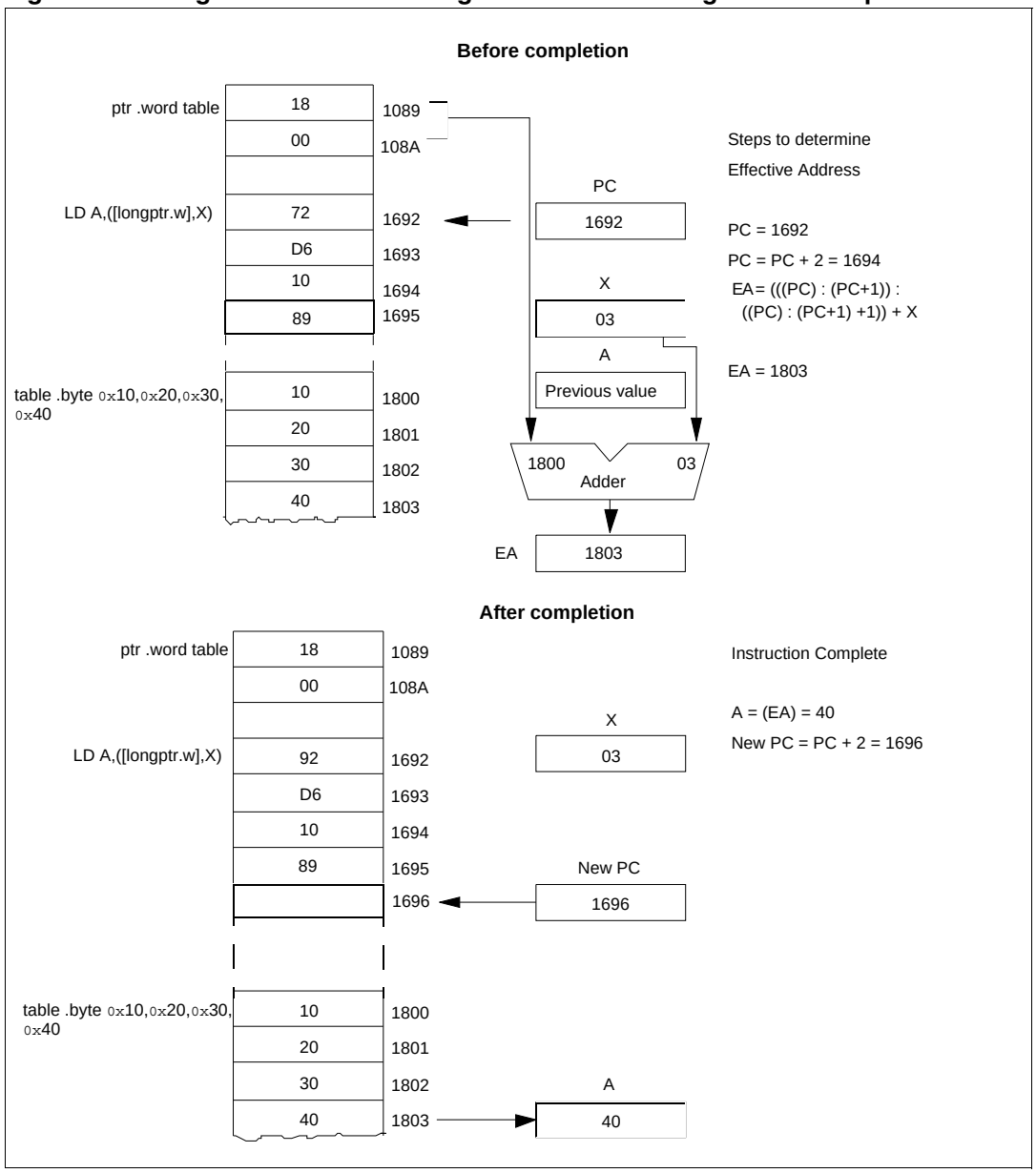
6.10 Long Pointer Indirect Long Indexed addressing mode

The pointer address is a word, the pointer size is a word, thus allowing up to 128 KB addressing space, and requires 2 bytes after the op-code.

Example:

```
1089   1800      ptr   dc.w  table


1800   10203040  table dc.b  $10,$20,$30,$40


1690   AE03            LD    X,#3
1692   72D61089        LD    A,([longptr.w],X)
                       X = 3
                       A = ([longptr.w],X) = ((longptr.w), X) =
                       (($1089.w), 3)
                       = ($1800,3) = ($1803) = $40
```

Figure 19. Long Pointer Indirect Long Indexed addressing mode example



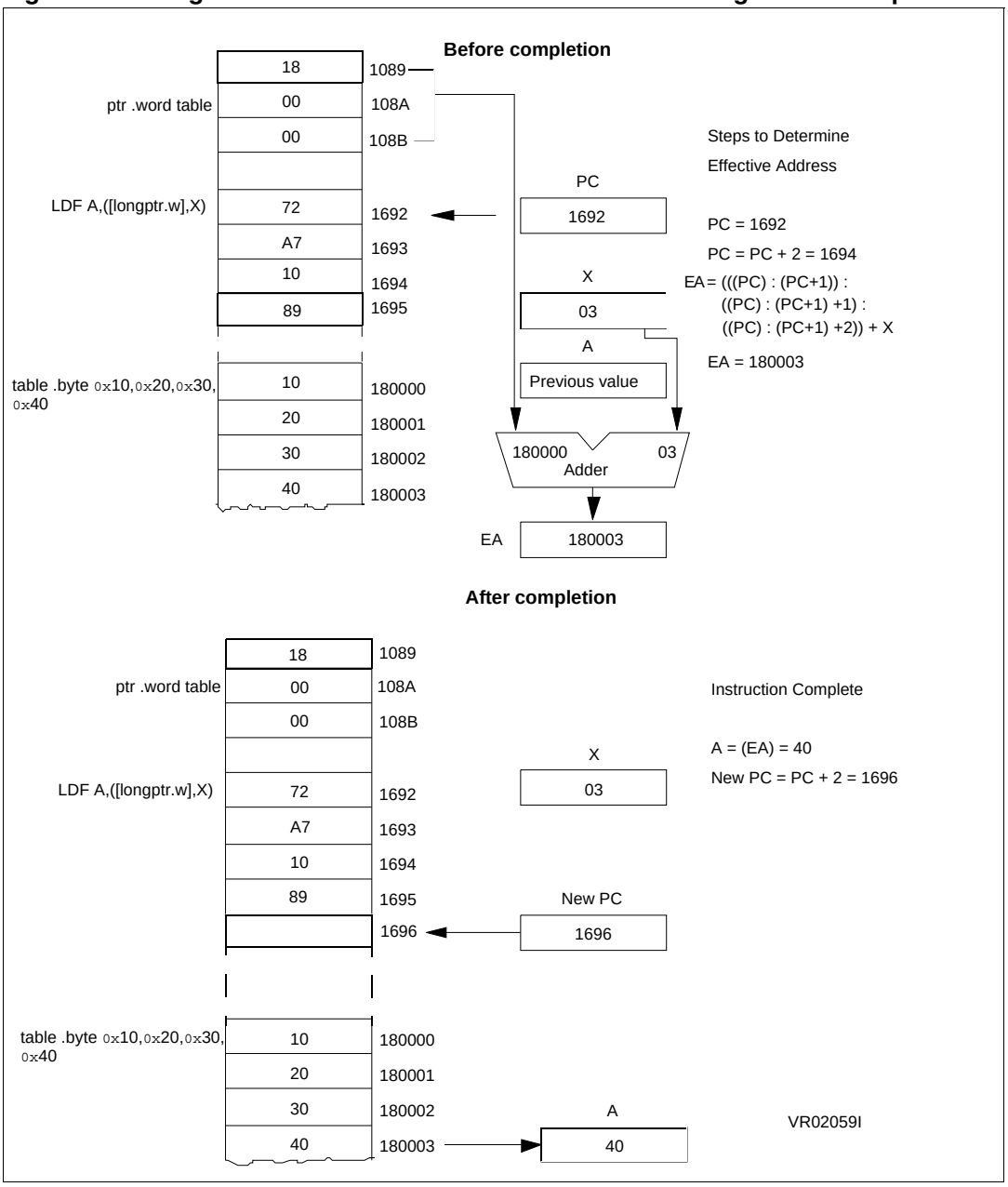
6.11 Long Pointer Indirect Extended Indexed addressing mode

The pointer address is a word, the pointer size is an extended word, thus allowing 16-Mbyte addressing space, and requires 2 bytes after the op-code.

Example:

```
1089   180000   ptr    dc.b     page(table), high(table), low(table)
180000 10203040 table  dc.b   $10,$20,$30,$40


1690   AE03             LD     X,#3
1692   72A71089         LDF    A,([longptr.e],X)
                        X = 3
                        A = ([longptr.e],X) = ((longptr.e), X) =
                        (($1089.e), 3)
                        = ($180000,3) = ($180003) = $40
```

Figure 20. Long Pointer Indirect Extended Indexed addressing mode example



6.12 Relative Direct addressing mode

Table 35. Overview of Relative Direct addressing mode instructions

Addressing mode		Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Direct	Relative	off	PC = PC + off	op + 1	---	PC +127/-128

This addressing mode is used to modify the PC register value, by adding an 8-bit signed offset to it. The offset added to the PC register value is relative to the start of the next instruction.

Table 36. Available Relative Direct instructions

Instructions	Functions
JRxx	Conditional Jump
JRA	Jump Relative Always
CALLR	Call Relative

The offset follows the op-code.

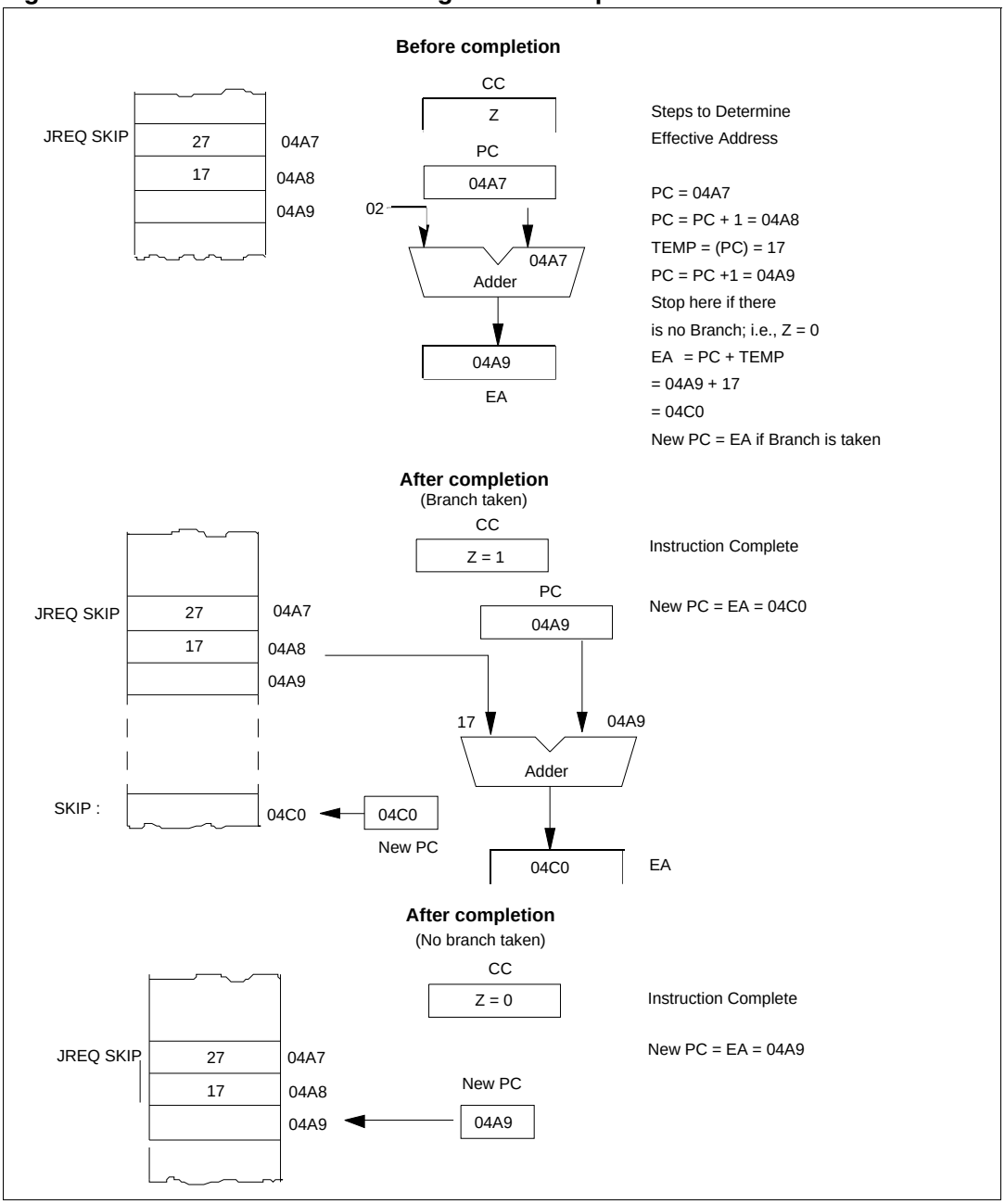
Example:

```
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
elsePC = PC= $04A9
```

Figure 21. Relative Direct addressing mode example



6.13 Bit Direct (Long) addressing mode

Table 37. Overview of Bit Direct addressing mode instruction

Addressing mode		Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Bit	Long Direct	longmem, #pos	(longmem)	op + 1..2	Word	0000..FFFF

The data byte required for the operation is found by its memory address, which follows the op-code. The bit used for the operation is selected by the bit selector which is encoded in the instruction op-code.

Table 38. Available Bit Direct instructions

Instructions	Functions
BRES	Bit Reset
BSET	Bit Set
BCPL	Bit Complement
BCCM	Copy Carry Bit to Memory

The address is a word, thus allowing 0000 to FFFF addressing space, but requires 2 bytes after the op-code. The bit selector #n (n=0 to 7) selects the n^{th} bit from the byte pointed to by the address.

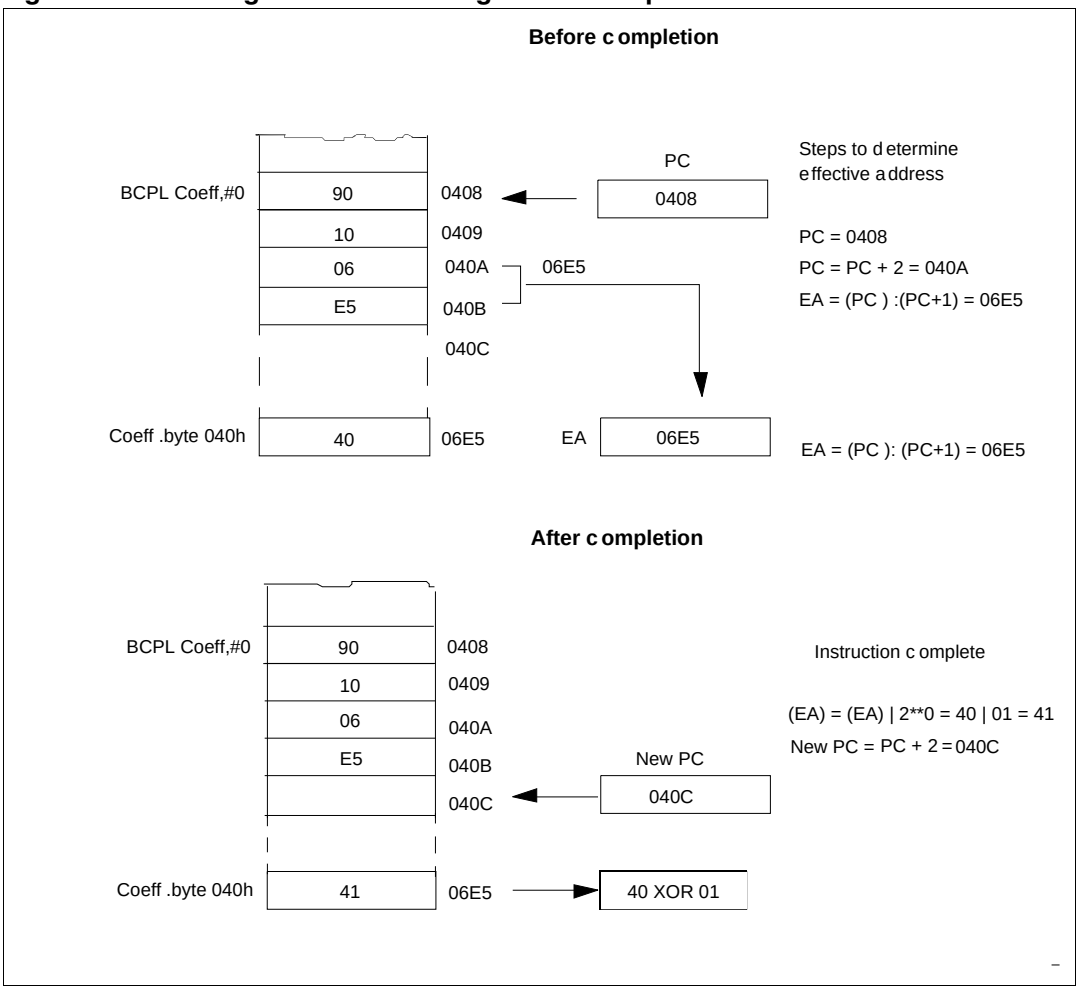
Example:

```
0408    721006E5              BCPL   coeff, #0
06E5    40              coeff dc.b   $ 40
```

Action:

```
(coeff) = ($06E5) XOR 2**0 = $40 XOR $01 = $41
```

Figure 22. Bit Long Direct addressing mode example



6.14 Bit Direct (Long) Relative addressing mode

Table 39. Overview of Bit Direct (Long) Relative addressing mode

Addressing mode			Syntax	EA formula	Ptr Adr	Ptr Size	Dest adr
Bit	Long Direct	Relative	longmem, #pos, off	(longmem)	op + 1..2	Word	0000..FFFF
				PC = PC + off	op + 3	Byte	PC +127/-128

This addressing mode is a combination between the Bit Direct addressing mode (for data addressing) and Relative Direct mode (for PC computation).

The data byte required for the operation is found by its memory address, which follows the op-code. The bit used for the test operation is selected by the bit selector which is encoded in the instruction op-code. Following the logical test operation, the PC register value can be modified, by adding an 8-bit signed offset to it.

Table 40. Available Bit Direct Relative instructions

Instructions	Functions
BTJT, BTJF	Bit Test and Jump

The data address is a word, thus allowing 0000 to FFFF addressing space (requires 2 bytes after the op-code). The bit selector #n (n=0 to 7) selects the n^{th} bit from the byte pointed to by the address. The offset follows the op-code and data address.

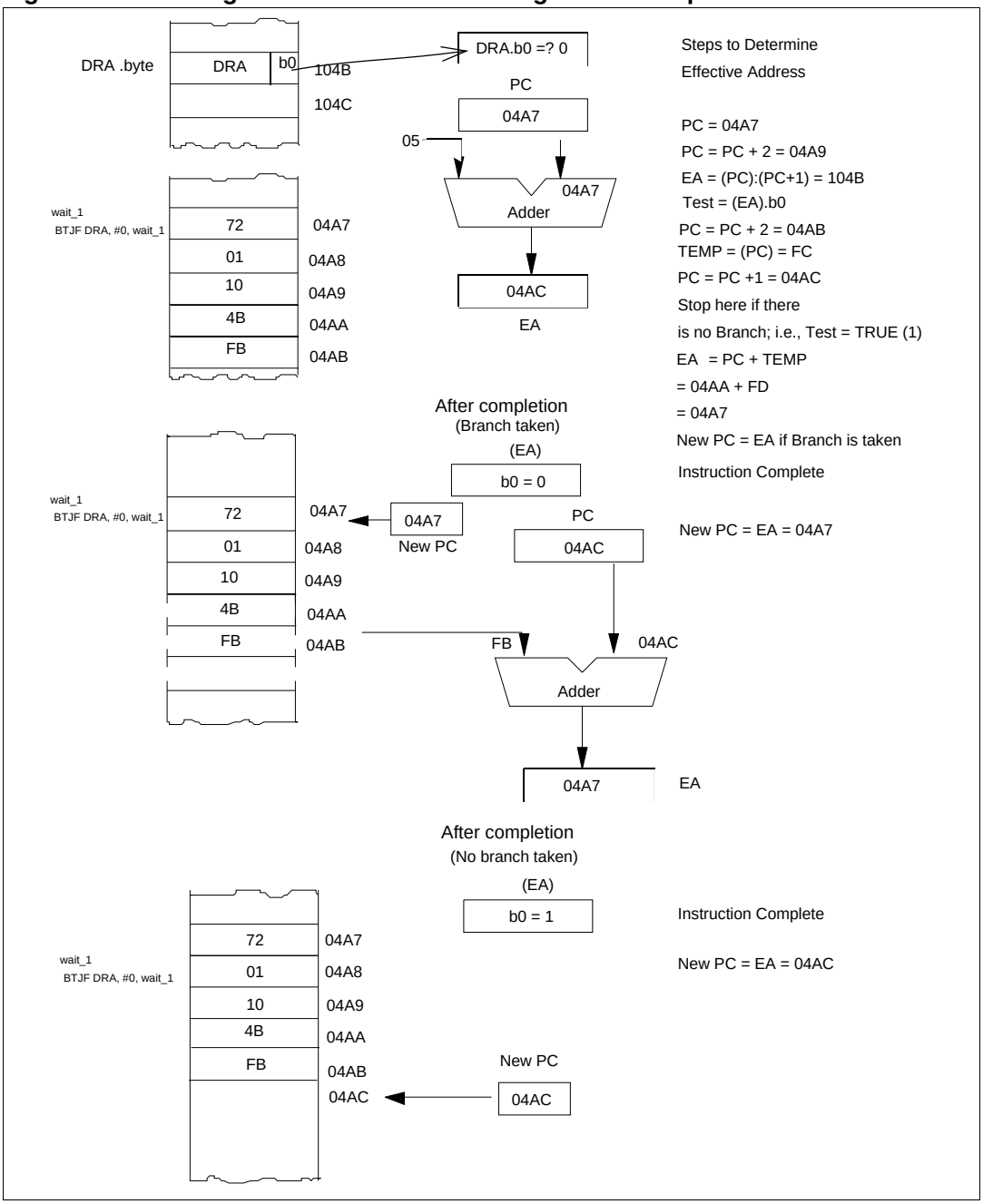
Example:

```
104B  00                  DRA   dc.b            $00   ; Port A data
                                                register  (input
                                                value)

                          bit0  equ             $0 ; data bit 0


04A7  7201104BFB wait_1   BTJF  DRA, bit0, wait_1
04AC   ....       cont_0
```

Action:

```
      Test = select_bit(0, ($4B)) = select_bit(0, DRA)
      if (Test /= 1)      then  PC = PC + $FB    = $0004AC - $05 =
                                                 $0004A7
                          else   PC = PC         = $0004AC
```

Figure 23. Bit Long Direct Relative addressing mode example



7 STM8 instruction set

7.1 Introduction

This chapter describes all the STM8 instructions. There are 96 and they are described in alphabetical order. However, they can be classified in 13 main groups as follows:

Table 41. Instruction groups

Load and Transfer	LD	LDF	CLR	MOV	EXG	LDW	CLRW	EXGW			
Stack operation	PUSH	POP	PUSH W	POPW							
Increment/ Decrement	INC	DEC	INCW	DECW							
Compare and Tests	CP	TNZ	BCP	CPW	TNZW						
Logical operations	AND	OR	XOR	CPL	CPLW						
Bit Operation	BSET	BRES	BCPL	BCCM							
Conditional Bit Test and Branch	BTJT	BTJF									
Arithmetic operations	NEG	ADC	ADD	SUB	SBC	MUL	DIV	DIVW	NEGW	ADDW	SUBW
Shift and Rotates	SLL	SRL	SRA	RLC	RRC	SWAP	SLLW	SRLW	SRAW	RLCW	RRCW
	SWAP	RLWA	RRWA								
Unconditional Jump or Call	JRA	JRT	JRF	JP	JPF	CALL	CALLR	CALLF	RET	RETF	NOP
Conditional Branch/ Execution	JRxx	WFE									
Interrupt management	TRAP	WFI	HALT	IRET							
Condition Code Flag modification	SIM	RIM	SCF	RCF	CCF	RVF					
Breakpoint/ software break	BREAK										

The instructions are described with one to five bytes.

PC-1 End of previous instruction

PC Op-code

PC+1..4 Additional word (0 to 4) according to the number of bytes required to compute the effective address(es)

Using a pre-code (two-byte op-codes)

In order to extend the number of available op-codes for an 8-bit CPU (256 op-codes), four different pre-code bytes are defined. These pre-codes modify the meaning of the instruction they precede.

The whole instruction becomes:

PC-1	End of previous instruction
PC	Pre-code
PC+1	Op-code
PC+2	Additional word (0 to 3) according to the number of bytes required to compute the effective address

These pre-bytes are:

0x90 = PDY	Replaces an X based instruction using immediate, direct, indexed or inherent addressing mode by a Y one. It also provides read/modify/write instructions using Y indexed addressing mode with long offset and two bit handling instructions (BCPL and BCCM)
0x92 = PIX	Replaces 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.
0x91 = PIY	Replace an instruction using indirect X indexed addressing mode by a Y one.
0x72 = PWSP	Provide long addressing mode for bit handling and read/modify/write instructions. It also provides indirect addressing mode with two byte pointer for read/modify/write and register/memory instructions. Finally it provides stack pointer indexed addressing mode on register/memory instructions.

7.2 Nomenclature

7.2.1 Operators

←	is loaded with ...
↔	has its value exchanged with ...

7.2.2 CPU registers

A	accumulator
X	X index register (2 bytes)
XL	least significant byte of the X index register (1 byte)
XH	most significant byte of the X index register (1 byte)
Y	Y index register (2 bytes)
YL	least significant byte of the Y index register (1 byte)
YH	most significant byte of the Y index register (1 byte)
PC	program counter register (3 bytes)
PCL	low significant byte of the program counter register (1 byte)
PCH	high significant byte of the program counter register (1 byte)
PCE	extended significant byte of the program counter register (1 byte)
SP	stack pointer register (2 bytes)
CC	Condition code register (1 byte)
CC.V	overflow flag of the code condition register (1 bit)
CC.I0	interrupt mask bit 0 of the code condition register (1 bit)
CC.H	half carry flag of the code condition register (1 bit)
CC.I1	interrupt mask bit 1 of the code condition register (1 bit)
CC.N	negative flag of the code condition register (1 bit)
CC.Z	zero flag of the code condition register (1 bit)
CC.C	carry flag of the code condition register (1 bit)

7.2.3 Code condition bit value notation

-	bit not affected by the instruction
1	bit forced to 1 by the instruction
0	bit forced to 0 by the instruction
X	bit modified by the instruction

7.2.4 Memory and addressing

M(...)	content of a memory location
R	8-bit operation result value
R(...)	8-bit operation result value stored into the register or memory shown inside parentheses
R*n*	bit n of the operation result value (0≤n≤7)
XX.B	bit *B* of the *XX* register or memory location
imm.b	byte immediate value
imm.w	16-bit immediate value
shortmem	memory location with short addressing mode (1 byte)
longmem	memory location with long addressing mode (2 bytes)
extmem	memory location with extended addressing mode (3 bytes)
shortoff	short offset (1 byte)
longoff	long offset (2 bytes)
extoff	extended offset (3 bytes)
[shortptr.w]	short pointer (1 byte) on long memory location (2 bytes). Assembler notation = [$12.w].
[longptr.w]	long pointer (2 bytes) on long memory location (2 bytes). Assembler notation = [$1234.w]
[longptr.e]	long pointer (2 bytes) on extended memory location (3 bytes). Assembler notation = [$1234.e]

7.2.5 Operation code notation

ee	extended order byte of 24-bit extended address
ww	high order byte of 16-bit long address or middle order byte of 24-bit extended address
bb	short address or low order byte of 16-bit long address or 24-bit extended address
ii	immediate data byte or low order byte of 16-bit immediate data
iw	high order byte of 16-bit immediate data
rr	relative offset byte in a range of [-128..+127]

7.3 Instruction set summary

Table 42. Instruction set summary

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
ADC	Add with carry	Set if the carry from R6 is different from the carry bit C	-	Set if there is a carry from bit 3 to 4 cleared otherwise	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Set if there is a carry from R7 cleared otherwise	ADC A,($12,SP)	A ← A + M(SP+shortoff) + CC.C	19 bb	1	
ADD	Add without carry	Set if the carry from R6 is different from the carry bit C	-	Set if there is a carry from bit 3 to 4 cleared otherwise	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Set if there is a carry from R7 cleared otherwise	ADD A,($12,SP)	A ← A + M(SP+shortoff)	1B bb	1	
		-	-	-	-	-	-	-	ADD SP,#$12	SP ← SP + imm.b	5B ii	2	
ADDW	Add word without carry	Set if the carry from R14 is different from the carry bit C	-	Set if there is a carry from bit 7 to 8 cleared otherwise	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Set if there is a carry from R15 cleared otherwise	ADDW X,($12,SP)	X ←-X + M(SP+shortoff)	72 FB bb	2	
AND	Logical AND	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	AND A,($12,SP)	A ← A AND M(SP+shortoff)	14 bb	1	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
BCCM	Copy carry in memory bit	-	-	-	-	-	-	-	BCCM $1234,#1	M(longmem).bit ← CC.C	90 1n ww bb n= 2*bit	1	
BCP	Logical bit compare		-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	BCP A,($12,SP)	test {A AND M(SP+shortoff) } N and Z are updated accordingly	15 bb	1	
BCPL	Complement bit in memory	-	-	-	-	-	-	-	BCPL $1234,#1	M(longmem).bit ← M(longmem).$\overline{bit}$	90 1n ww bb n= 2*bit	1	
BREAK	Software breakpoint	-	-	-	-	-	-	-	SW-BREAK		8B	1	Flush
BRES	Bit reset	-	-	-	-	-	-	-	BRES $1234,#1	M(longmem).bit ← 0	72 1n ww bb n= 1 + 2*bit	1	
BSET	Bit set	-	-	-	-	-	-	-	BSET $1234,#1	M(longmem).bit ← 1	72 1n ww bb n= 2*bit	1	
BTJF	Bit test and relative jump if condition is false	-	-	-	-	-	-	tested bit	BTJF $1234,#1,label	if M(longmem).bit=0 then PC ← PC + 4 + rr else PC ← PC + 4	72 0n ww bb n= 1 + 2*bit	2/3	Flush (2)
BTJT	Bit test and relative jump if condition is true	-	-	-	-	-	-	tested bit	BTJT $1234,#1,label	if M(longmem).bit=1 then PC ← PC + 4 + rr else PC ← PC + 4	72 0n ww bb n= 2*bit	2/3	Flush (2)
CALL	Call to Subroutine with address in same section	-	-	-	-	-	-	-	CALL [$1234.w]	PC← PC + 4 M(SP--) ← PCL M(SP--) ← PCH PCH← M(longmem) PCL← M(longmem + 1)	72 CD ww bb	6	Flush
CALLF	Call to subroutine with extended address	-	-	-	-	-	-	-	CALLF $123456	PC ← PC+4 M(SP--) ← PCL M(SP--) ← PCH M(SP--) ← PCE PC ← extmem	8D ee ww bb	5	Flush
CALLR	Call Subroutine relative	-	-	-	-	-	-	-	CALLR label	PC ← PC + 4 M(SP--) ← PCL M(SP--) ← PCH PC ← PC + rr	AD bb	4	Flush
CCF	Complement carry flag	-	-	-	-	-	-	C	CCF	CC.C ← CC.$\overline{C}$	8C	1	
CLR	Clears the destination byte	-	-	-	-	0	1	-	CLR ([$1234.w],X)	M(M(longmem).w + X) ← 0x00	72 6F ww bb	4	
CLRW	Clears the destination index register	-	-	-	-	0	1	-	CLRW X	X ← 0x0000	5F	1	
CP	Compare	Set if A-mem (signed values) overflows, cleared otherwise	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Set if A<mem (unsigned values) cleared otherwise	CP A,($12,SP)	test { A - M(SP+shortoff) }	11 bb	1	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
CPW	Compare word	Set if Xmmem (signed values) overflows, cleared otherwise	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Set if X<mem (unsigned values) cleared otherwise	CPW X,($12,SP)	test { X - M(SP+shortoff) }	13 bb	2	
CPL	Logical 1's complement			-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	1	CPL ([$1234.w],X)	M(M(longmem).w +X) ← FF - M(M(longmem).w+X) or M(M(longmem).w+X) XOR FF	72 63 ww bb	4	
CPLW	Logical 1's complement			-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	1	CPLW X	X ← FFFF - X or X XOR FFFF	53	2	
DEC	Decrement byte by one	Set if sign overflow cleared otherwise	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	DEC ([$1234.w],X)	M(M(longmem).w + X) ← M(M(longmem).w + X) - 1	72 6A ww bb	4	
DECW	Decrement word by one	Set if sign overflow cleared otherwise	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	-	DECW X	X← X - 1	5A	1	
DIV	16 by 8 Unsigned division	0	-	0	-	0	Set if Q=$0000 cleared otherwise	Set if divide by 0 cleared otherwise	DIV X,A	X ← X/A (Quotient) A ← X%A (Remainder)	62	16	
									DIV Y,A	Y ← Y/A (Quotient) A ← Y%A (Remainder)	90 62	16	
DIVW	16 by 16 Unsigned division	0	-	0	-	0	Set if Q=$0000 cleared otherwise	Set if divide by 0 cleared otherwise	DIVW X,Y	X ← X/Y (Quotient) Y ← X%Y (Remainder)	65	16	
EXG	Data byte exchange	-	-	-	-	-	-	-	EXG A,$1234	A ↔ M(longmem)	31 ww bb	3	
									EXG A,XL	A ↔ XL	41	1	
									EXG A,YL	A ↔ YL	61	1	
EXGW	Data word exchange	-	-	-	-	-	-	-	EXGW X,Y	X ↔ Y	51	1	
HALT	Halt oscillator (CPU + Peripherals)	-	1	-	0	-	-	-	HALT	CC.I0 ← 0 , CC.I1 ← 1 Oscillator stopped till an interrupt occurs	8E	10	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
INC	Increment byte by one	Set if sign overflow cleared otherwise	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	INC ([$1234.w],X)	M(M(longmem).w + X) ← M(M(longmem).w + X) + 1	72 6C ww bb	4	
INCW	Increment word by one	Set if sign overflow cleared otherwise	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	-	INCW X	X ← X + 1	5C	2	
INT	Interrupt	-	-	-	-	-	-	-	INT $123456	PC ← extmem	82 ee ww bb	2	
IRET	Interrupt return	Updated according to the value pop from the stack into CC register							IRET	(++SP) CC ← M(++SP) A ← M(++SP) X ← M(++SP); SP++ Y ← M(++SP); SP++ PCE ← M(++SP) PCH ← M(++SP) PCL ← M(++SP)	80	11	Flush
JP	Jump to an address in section 0	-	-	-	-	-	-	-	JP ([$1234.w],X)	PC ← M(longmem).w + X	72 DC ww bb	5	Flush
JPF	Jump to an extended address	-	-	-	-	-	-	-	JPF $123456	PC ← extmem	AC ee ww bb	2	Flush
JRA	Unconditional relative jump	-	-	-	-	-	-	-	JRA Label	PC ← PC + 2+ rr	20 bb	2	Flush
JRC	Jump if C = 1	-	-	-	-	-	-	-	JRC Label	if CC.C =1 then PC ← PC + 2+ rr else PC ← PC + 2	25 bb	1/2	Flush (2)
JREQ	Jump if Z = 1(equal)	-	-	-	-	-	-	-	JREQ Label	if CC.Z = 1 then PC ← PC + 2+ rr else PC ← PC + 2	27 bb	1/2	Flush (2)
JRF	Never Jump	-	-	-	-	-	-	-	JRF Label	----------	21 bb	1	
JRH	Jump if H = 1	-	-	-	-	-	-	-	JRH Label	if CC.H = 1 then PC ← PC + 2+ rr else PC ← PC + 2	90 29 bb	1/2	Flush (2)
JRIH	Jump if Port INT pin = 1	-	-	-	-	-	-	-	JRIH Label	if Port INT pin =1 then PC ← PC + 2+ rr else PC ← PC + 2	90 2F bb	1/2	Flush (2)
JRIL	Jump if Port INT pin = 0	-	-	-	-	-	-	-	JRIL Label	if Port INT pin = 0 then PC ← PC + 2+ rr else PC ← PC + 2	90 2E bb	1/2	Flush (2)
JRM	Jump if Interrupts are masked	-	-	-	-	-	-	-	JRM Label	if I0 AND I1 = 1 then PC ← PC + 2 + rr else PC ← PC + 2	90 2D bb	1/2	Flush (2)
JRMI	Jump if N = 1(minus)	-	-	-	-	-	-	-	JRMI Label	if CC.N = 1 then PC ← PC + 2+ rr else PC ← PC + 2	2B bb	1/2	Flush (2)
JRNC	jump if C = 0	-	-	-	-	-	-	-	JRNC Label	if CC.C =0 then PC ← PC + 2+ rr else PC ← PC + 2	24 bb	1/2	Flush (2)

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
JRNE	Jump if Z =0 (not equal)	-	-	-	-	-	-	-	JRNE Label	if CC.Z = 0 then PC ← PC + 2+ rr else PC ← PC + 2	26 bb	1/2	Flush (2)
JRNH	Jump if H = 0	-	-	-	-	-	-	-	JRNH Label	if CC.H = 0 then PC ← PC + 2+ rr else PC ← PC + 2	90 28 bb	1/2	Flush (2)
JRNM	Jump if Interrupts are not masked	-	-	-	-	-	-	-	JRNM Label	if I0 AND I1= 0 then PC ← PC + 2 + rr else PC ← PC + 2	90 2C bb	1/2	Flush (2)
JRNV	jump if V = 0	-	-	-	-	-	-	-	JRNV Label	if CC.C =0 then PC ← PC + 2+ rr else PC ← PC + 2	28 bb	1/2	Flush (2)
JRPL	Jump if N = 0 (plus)	-	-	-	-	-	-	-	JRPL Label	if CC.N = 0 then PC ← PC + 2+ rr else PC ← PC + 2	2A bb	1/2	Flush (2)
JRSGE	Jump if (N xor V) = 0	-	-	-	-	-	-	-	JRSGE Label	if (CC.N xor CC.V) = 0 then PC ← PC + 2+ rr else PC ← PC + 2	2E bb	1/2	Flush (2)
JRSGT	Jump if (Z or (N xor V)) = 0	-	-	-	-	-	-	-	JRSGT Label	if (CC.Z or (CC.N xor CC.V)) = 0 then PC ← PC + 2+ rr else PC ← PC + 2	2C bb	1/2	Flush (2)
JRSLE	Jump if (Z or (N xor V)) = 1	-	-	-	-	-	-	-	JRSLE Label	if (CC.Z or (CC.N xor CC.V)) = 1 then PC ← PC + 2+ rr else PC ← PC + 2	2D bb	1/2	Flush (2)
JRSLT	Jump if (N xor V) = 1	-	-	-	-	-	-	-	JRSLT Label	if (CC.N xor CC.V) = 1 then PC ← PC + 2+ rr else PC ← PC + 21	2F bb	1/2	Flush (2)
JRT	Jump relative	-	-	-	-	-	-	-	JRT Label	PC ← PC + 2+ rr	20 bb	2	Flush
JRUGE	Jump if C = 0	-	-	-	-	-	-	-	JRUGE Label	if CC.C = 0 then PC ← PC + 2+ rr else PC ← PC + 2	24 bb	1/2	Flush (2)
JRUGT	Jump if (C+Z = 0)	-	-	-	-	-	-	-	JRUGT Label	if (CC.C = 0 and CC.Z = 0) then PC ← PC + 2+ rr else PC ← PC + 2	22 bb	1/2	Flush
JRULE	Jump if (C+Z =1)	-	-	-	-	-	-	-	JRULE Label	if (CC.C = 1 and CC.Z = 1) then PC ← PC + 2+ rr else PC ← PC + 2	23 bb	1/2	Flush
JRULT	Jump if C = 1	-	-	-	-	-	-	-	JRULT Label	if CC.C = 1 then PC ← PC + 2+ rr else PC ← PC + 21	25 bb	1/2	Flush (2)
JRV	Jump if V = 1	-	-	-	-	-	-	-	JRV Label	if CC.V =1 then PC ← PC + 2+ rr else PC ← PC + 2	29 bb	1/2	Flush
LD	A register load	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	LD A,($12,SP)	A ← M(SP+shortoff)	7B bb	1	
	A register store								LD ($12,SP),A	M(SP+shortoff) ← A	6B bb	1	
	Register to register move					-	-		LD A, XH	A ← XH	95	1	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
LDF	Data load / store with extended address	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	LDF A,($123456,X)	A ← M(X+extoff)	AF ee ww bb	1	
									LDF A,($123456,Y)	A ← M(Y+extoff)	90 AF ee ww bb	1	
									LDF A,([$1234.e],X)	A ← M(X+[longptr.e])	92 AF ww bb	5	
									LDF ($123456,X),A	M(X+extoff) ← A	A7 ee ww bb	1	
									LDF ($123456,Y),A	M(Y+extoff) ← A	90 A7 ee ww bb	1	
									LDF ([1234.e],X),A	M(X+[longptr.e]) ← A	92 A7 ww bb	5	
LDW	X register load	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	-	LDW X,($12,SP)	X ← M(SP+shortoff)	1E bb	2	
	X register store								LDW ($12,SP),X	M(SP+shortoff) ← X	1F bb	2	
	Y register load								LDW Y,($12,SP)	Y ← M(SP+shortoff)	16 bb	2	
	Y register store								LDW ($12,SP),Y	M(SP+shortoff) ← Y	17 bb	2	
	SP register load / store					-	-		LDW SP,X	SP ← X	94	1	
									LDW X,SP	X ← SP	96	1	
	Index register move								LDW X, Y	X ← Y	93	1	
MOV	Data byte move	-	-	-	-	-	-	-	MOV $1234,#$12	M(longmem) ← imm.b	35 ii ww bb	1	
									MOV $12,$34 MOV mem1,mem2	M(mem1.b) ← M(mem2.b)	44 b2 b1	1	
									MOV $1234,$5678 MOV mem1,mem2	M(mem1.w) ← M(mem2.w)	45 w2 b2 w1 b1	1	
MUL	8 by 8 multiplication (unsigned)	-	-	0	-	-	-	0	MUL X,A	X ← X*A	42	4	
									MUL Y,A	Y ← Y*A	90 42	4	
NEG	Logical 2's complement	Set if M=$80 cleared otherwise	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Cleared if R=$00 set otherwise	NEG ([$1234.w],X)	M(M(longmem) + X) ← 00 - M(M(longmem) + X)	72 60 ww bb	4	
NEGW	Logical 2's complement	Set if X=$8000 cleared otherwise	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Cleared if R=$0000 set otherwise	NEGW X	X ← 0000 - X	50	2	
NOP	No operation	-	-	-	-	-	-	-	NOP	---------	9D	1	
OR	Logical OR	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	OR A,($12,SP)	A ← A OR M(SP+shortoff)	1A bb	1	
POP	Pop data byte from stack	-	-	-	-	-	-	-	POP $1234	M(longmem) ← M(++SP)	32 ww bb	1	
	Pop code condition register								POP CC	CC ← M(++SP)	86	1	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
POPW	Pop index register from stack	-	-	-	-	-	-	-	POPW X	XH ← M(++SP) XL ← M(++SP)	85	2	
PUSH	Push data byte onto stack	-	-	-	-	-	-	-	PUSH $1234	M(SP--) ← M(longmem)	3B ww bb	1	
									PUSH #$12	M(SP--) ← imm.b	4B bb	1	
PUSHW	Push index register onto stack	-	-	-	-	-	-	-	PUSHW X	M(SP--) ← XL M(SP--) ← XH	89	2	
RCF	Reset carry flag	-	-	-	-	-	-	0	RCF	CC.C ← 0	98	1	
RET	Subroutine return from section 0	-	-	-	-	-	-	-	RET	PCH ← M(++SP) PCL ← M(++SP)	81	4	Flush
RETF	Subroutine return from extended address	-	-	-	-	-	-	-	RETF	PCE ← M(++SP) PCH ← M(++SP) PCL ← M(++SP)	87	5	Flush
RIM	Reset interrupt mask/ Interrupt enable	-	1	-	0	-	-	-	RIM	CC.I1 ← 1	9A	1	
RLC	Rotate left logical through carry	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Bit 7 of the byte before rotation	RLC ([$1234.w],X)	R0 ← CC.C R1 ← bit 0 R2 ← bit 1 R3 ← bit 2 R4 ← bit 3 R5 ← bit 4 R6 ← bit 5 R7 ← bit 6 CC.C ← bit 7	72 69 ww bb	4	
RLCW	Rotate word left logical through carry	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Bit 7 of the byte before rotation	RLCW X	R0 ← CC.C R1 ← bit 0 R2 ← bit 1 ... R13 ← bit 12 R14 ← bit 13 R15 ← bit 14 CC.C ← bit 15	59	2	
RLWA	Rotate word left through Accumulator	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	-	RLWA X	A ← XH XH ← XL XL ← A	02	1	
RRC	Rotate right logical through carry	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Bit 0 of the byte before rotation	RRC ([$1234.w],X)	R7 ← CC.C R6 ← bit 7 R5 ← bit 6 R4 ← bit 5 R3 ← bit 4 R2 ← bit 3 R1 ← bit 2 R0 ← bit 1 CC.C ← bit 0	72 66 ww bb	4	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
RRCW	Rotate word right logical through carry	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Bit 0 of the byte before rotation	RRCW X	R15 ← CC.C R14 ← bit 15 R13 ← bit 14 ... R2 ← bit 3 R1 ← bit 2 R0 ← bit 1 CC.C ← bit 0	56	2	
RRWA	Rotate word right through Accumulator	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	-	RRWA X	A ← XL XL ← XH XH ← A	01	1	
RVF	Reset overflow flag	0	-	-	-	-	-	-	RVF	CC.V ← 0	9C	1	
SBC	Subtract with carry	Set if the signed subtraction generates an overflow, cleared otherwise	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Set if there is a carry from R7 cleared otherwise	SBC A,($12,SP)	A ← A -M(SP+shortoff) - CC.C	12 bb	1	
SCF	Set Carry Flag	-	-	-	-	-	-	1	SCF	CC.C ← 1	99	1	
SIM	Set interrupt mask/ Disable interrupts	-	1	-	1	-	-	-	SIM	CC.I0 ← 1 CC.I1 ← 1	9B	1	
SLA	Shift left arithmetic	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Bit 7 of the byte before shifting	SLA ([$1234.w],X)	R0 ← 0 R1 ← bit 0 R2 ← bit 1 R3 ← bit 2 R4 ← bit 3 R5 ← bit 4 R6 ← bit 5 R7 ← bit 6 CC.C ← bit 7	72 68 ww bb	4	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
SLAW	Shift word left arithmetic	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Bit 15 of the byte before shifting	SLAW X	R0 ← 0 R1 ← bit 0 R2 ← bit 1 R3 ← bit 2 ..... R14 ← bit 13 R15 ← bit 14 CC.C ← bit 15	58	2	
SLL	Shift left logical	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Bit 7 of the byte before shifting	SLL ([$1234.w],X)	R0 ← 0 R1 ← bit 0 R2 ← bit 1 R3 ← bit 2 R4 ← bit 3 R5 ← bit 4 R6 ← bit 5 R7 ← bit 6 CC.C ← bit 7	72 68 ww bb	4	
SLLW	Shift word left logical	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Bit 15 of the byte before shifting	SLLW X	R0 ← 0 R1 ← bit 0 R2 ← bit 1 R3 ← bit 2 ..... R14 ← bit 13 R15 ← bit 14 CC.C ← bit 15	58	2	
SRA	Shift right arithmetic	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Bit 0 of the byte before shifting	SRA ([$1234.w],X)	CC.C ← bit 0 R0 ← bit 1 R1 ← bit 2 R2 ← bit 3 R3 ← bit 4 R4 ← bit 5 R5 ← bit 6 R6 ← bit 7 R7 ← bit 7 (unchanged)	72 67 ww bb	4	
SRAW	Shift word right arithmetic	Set if R7 set cleared otherwise	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Bit 0 of the byte before shifting	SRAW X	CC.C ← bit 0 R0 ← bit 1 R1 ← bit 2 R2 ← bit 3 .... R12 ← bit 13 R13 ← bit 14 R14 ← bit 15 R15 ← bit 15 (unchanged)	57	2	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
SRL	Shift right logical	-	-	-	-	Set if R7 set cleared otherwise	Set if R=$00 cleared otherwise	Bit 0 of the byte before shifting	SRL ([$1234.w],X)	CC.C ← bit 0 R0 ← bit 1 R1 ← bit 2 R2 ← bit 3 R3 ← bit 4 R4 ← bit 5 R5 ← bit 6 R6 ← bit 7 R7 ← 0	72 64 ww bb	4	
SRLW	Shift word right arithmetic	-	-	-	-	Set if R15 set cleared otherwise	Set if R=$0000 cleared otherwise	Bit 0 of the byte before shifting	SRLW X	CC.C ← bit 0 R0 ← bit 1 R1 ← bit 2 R2 ← bit 3 .... R12 ← bit 13 R13 ← bit 14 R14 ← bit 15 R15 ← 0	54	2	
SUB	Subtract without carry	Set if the signed operation generates an overflow, cleared otherwise	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	Set if there is a carry from R7 cleared otherwise	SUB A,($12,SP)	A ← A -M(SP+shortoff)	10 bb	1	
		-	-	-	-	-	-	-	SUB SP,#$12	SP ← SP + imm.b	52 ii	2	
SUBW	Subtract word without carry	Set if X< mem (unsigned 16-bit values), cleared otherwise	-	Set if dst(7:0)< mem(7:0) (unsigned values) cleared otherwise	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	Set if dst < mem (unsigned values) cleared otherwise	SUBW X,($12,SP)	X ← X -M(SP+shortoff)	72 F0 bb	2	
SWAP	Swap nibbles	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	SWAP ([$1234.w],X)	R0 ↔ R4 R1 ↔ R5 R2 ↔ R6 R3 ↔ R7	72 6E ww bb	4	

Table 42. Instruction set summary (continued)

Mnemo	Description	Effect on CC register							Syntax example	Operation	Example op-code(s)	Cycles(1)	Pipe
		V	I1	H	I0	N	Z	C					
SWAPW	Swap bytes	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	-	SWAPW X	R0 ↔ R8 R1 ↔ R9 R2 ↔ R10 R3 ↔ R11 R4 ↔ R12 R5 ↔ R13 R6 ↔ R14 R7 ↔ R15	5E	1	
TNZ	Test for negative or zero	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	TNZ ([$1234.w],X)	CC.N ← R7 CC.Z ← 1 if R=$00 ← 0 otherwise	72 6D ww bb	4	
TNZW	Test word for negative or zero	-	-	-	-	Set if R15 is set cleared otherwise	Set if R=$0000 cleared otherwise	-	TNZW X	CC.N ← R15 CC.Z ← 1 if R=$0000 ← 0 otherwise	5D	2	
TRAP	Software interrupt	-	1	-	1	-	-	-	TRAP	PC ← PC+1 M(SP--) ← PCL M(SP--) ← PCH M(SP--) ← PCE M(SP--) ← YL M(SP--) ← YH M(SP--) ← XL M(SP--) ← XH M(SP--) ← A M(SP--) ← CC PC ← TRAP vector address	83	9	Flush
WFE	Wait for event (CPU stopped, Low power mode)	-	-	-	-	-	-	-	WFE	CPU clock stopped till the event input is activated. Internal peripherals are still running	72 8F	1	
WFI	Wait for interrupt (CPU stopped, Low power mode)	-	1	-	0	-	-	-	WFI	CC.I0 ← 0, CC.I1 ← 1 CPU clock stopped till an interrupt occurs. Internal peripherals are still running	8F	10	
XOR	Logical exclusive OR	-	-	-	-	Set if R7 is set cleared otherwise	Set if R=$00 cleared otherwise	-	XOR A,($12,SP)	A ← A XOR M(SP+shortoff)	18 bb	1	

1. Number of cycles corresponding to the example op-code.
2. If branch taken.

7.4 Instruction set

The following pages give a detailed description of each STM8 instruction.

ADC Addition with Carry ADC

Syntax ADC A, src e.g. ADC A,#$15

Operation A <= A+ src + C

Description The source byte, along with the carry flag, is added to the contents of the accumulator and the result is stored in the accumulator. This instruction is useful for addition of operands that are larger than eight.

The source is

a memory or data byte.

Instruction overview:

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
ADC	A	Mem	V	-	H	-	N	Z	C

V ⇒ $(A7.M7 + M7.\overline{R7} + \overline{R7}.A7) \oplus (A6.M6 + M6.\overline{R6} + \overline{R6}.A6)$
Set if the signed operation generates an overflow, cleared otherwise.

H ⇒ $A3.M3 + M3.\overline{R3} + \overline{R3}.A3$
Set if a carry occurred from bit 3 of the result, cleared otherwise.

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ $A7.M7 + M7.\overline{R7} + \overline{R7}.A7$
Set if a carry occurred from bit 7 of the result, cleared otherwise.

Detailed description:

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	ADC A,#$55	1	2		A9	XX			✗
A	shortmem	ADC A,$10	1	2		B9	XX			✗
A	longmem	ADC A,$1000	1	3		C9	MS	LS		✗
A	(X)	ADC A,(X)	1	1		F9				✗
A	(shortoff,X)	ADC A,($10,X)	1	2		E9	XX			✗
A	(longoff,X)	ADC A,($1000,X)	1	3		D9	MS	LS		✗
A	(Y)	ADC A,(Y)	1	2	90	F9				✗
A	(shortoff,Y)	ADC A,($10,Y)	1	3	90	E9	XX			✗
A	(longoff,Y)	ADC A,($1000,Y)	1	4	90	D9	MS	LS		✗
A	(shortoff,SP)	ADC A,($10,SP)	1	2		19	XX			
A	[shortptr.w]	ADC A,[$10.w]	4	3	92	C9	XX			✗
A	[longptr.w]	ADC A,[$1000.w]	4	4	72	C9	MS	LS		
A	([shortptr.w],X)	ADC A,([$10.w],X)	4	3	92	D9	XX			✗
A	([longptr.w],X)	ADC A,([$1000.w],X)	4	4	72	D9	MS	LS		
A	([shortptr.w],Y)	ADC A,([$10.w],Y)	4	3	91	D9	XX			✗

See also: ADD, SUB, SBC, MUL, DIV

ADD Addition ADD

Syntax ADD A,src e.g. ADD A,#%11001010

Operation A <= A+ src

Description The source byte is added to the contents of the accumulator and the result is stored in the accumulator. The source is a memory or data byte.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
ADD	A	Mem	V	-	H	-	N	Z	C

V ⇒ $(A7.M7 + M7.\overline{R7} + \overline{R7}.A7) \oplus (A6.M6 + M6.\overline{R6} + \overline{R6}.A6)$
Set if the signed operation generates an overflow, cleared otherwise.

H ⇒ $A3.M3 + M3.\overline{R3} + \overline{R3}.A3$
Set if a carry occurred from bit 3 of the result, cleared otherwise.

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ $A7.M7 + M7.\overline{R7} + \overline{R7}.A7$
Set if a carry occurred from bit 7 of the result, cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	ADD A,#$55	1	2		AB	XX			✗
A	shortmem	ADD A,$10	1	2		BB	XX			✗
A	longmem	ADD A,$1000	1	3		CB	MS	LS		✗
A	(X)	ADD A,(X)	1	1		FB				✗
A	(shortoff,X)	ADD A,($10,X)	1	2		EB	XX			✗
A	(longoff,X)	ADD A,($1000,X)	1	3		DB	MS	LS		✗
A	(Y)	ADD A,(Y)	1	2	90	FB				✗
A	(shortoff,Y)	ADD A,($10,Y)	1	3	90	EB	XX			✗
A	(longoff,Y)	ADD A,($1000,Y)	1	4	90	DB	MS	LS		✗
A	(shortoff,SP)	ADD A,($10,SP)	1	2		1B	XX			
A	[shortptr.w]	ADD A,[$10.w]	4	3	92	CB	XX			✗
A	[longptr.w]	ADD A,[$1000.w]	4	4	72	CB	MS	LS		
A	([shortptr.w],X)	ADD A,([$10.w],X)	4	3	92	DB	XX			✗
A	([longptr.w],X)	ADD A,([$1000.w],X)	4	4	72	DB	MS	LS		
A	([shortptr.w],Y)	ADD A,([$10.w],Y)	4	3	91	DB	XX			✗

See also: ADDW, ADC, SUB, SBC, MUL, DIV

ADDW Word Addition with index registers ADDW

Syntax ADDW dst,src e.g. ADDW X,#$1000

Operation dst <= dst + src

Description The source (16-bit) is added to the contents of the destination, which is an index register (X/Y) and the result is stored in the same index register. The source is a 16-bit memory or data word. The ADDW instruction can also be used to add an immediate value to the stack pointer (SP).

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
ADDW	X	Mem	V	-	H	-	N	Z	C
ADDW	Y	Mem	V	-	H	-	N	Z	C
ADDW	SP	Imm	-	-	-	-	-	-	-

V ⇒ $(A15.M15 + M15.\overline{R15} + \overline{R15}.A15) \oplus (A14.M14 + M14.\overline{R14} + \overline{R14}.A14)$
Set if the signed operation generates an overflow, cleared otherwise.

H ⇒ $X7.M7 + M7.\overline{R7} + \overline{R7}.X7$
Set if a carry occurred from bit 7 of the result, cleared otherwise.

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

C ⇒ $X15.M15 + M15.\overline{R15} + \overline{R15}.X15$
Set if a carry occurred from bit 15 of the result, cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	#word	ADDW X,#$1000	2	3		1C	MS	LS		
X	longmem	ADDW X,$1000	2	4	72	BB	MS	LS		
X	(shortoff,SP)	ADDW X,($10,SP)	2	3	72	FB	XX			
Y	#word	ADDW Y,#$1000	2	4	72	A9	MS	LS		
Y	longmem	ADDW Y,$1000	2	4	72	B9	MS	LS		
Y	(shortoff,SP)	ADDW Y,($10,SP)	2	3	72	F9	XX			
SP	#byte	ADDW SP,#$9	2	2		5B	XX			

See also: ADD, ADC, SUB, SBC, MUL, DIV

AND Logical AND AND

Syntax AND A,src e.g. AND A,#%00110101

Operation A <= A AND src

Description The source byte, is ANDed with the contents of the accumulator and the result is stored in the accumulator. The source is a memory or data byte.

Truth table:

AND	0	1
0	0	0
1	0	1

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
AND	A	Mem	-	-	-	-	N	Z	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	AND A,#$55	1	2		A4	XX			✗
A	shortmem	AND A,$10	1	2		B4	XX			✗
A	longmem	AND A,$1000	1	3		C4	MS	LS		✗
A	(X)	AND A,(X)	1	1		F4				✗
A	(shortoff,X)	AND A,($10,X)	1	2		E4	XX			✗
A	(longoff,X)	AND A,($1000,X)	1	3		D4	MS	LS		✗
A	(Y)	AND A,(Y)	1	2	90	F4				✗
A	(shortoff,Y)	AND A,($10,Y)	1	3	90	E4	XX			✗
A	(longoff,Y)	AND A,($1000,Y)	1	4	90	D4	MS	LS		✗
A	(shortoff,SP)	AND A,($10,SP)	1	2		14	XX			
A	[shortptr.w]	AND A,[$10.w]	4	3	92	C4	XX			✗
A	[longptr.w]	AND A,[$1000.w]	4	4	72	C4	MS	LS		
A	([shortptr.w],X)	AND A,([$10.w],X)	4	3	92	D4	XX			✗
A	([longptr.w],X)	AND A,([$1000.w],X)	4	4	72	D4	MS	LS		
A	([shortptr.w],Y)	AND A,([$1000],Y)	4	3	91	D4	XX			✗

See also: OR, XOR, CPL, NEG

BCCM Copy Carry Bit to Memory BCCM

Syntax BCCM dst, #pos (pos=0..7) e.g. BCCM $1234,#1

Operation dst(pos) <= CC.C

Description Copies the Carry flag of the Condition Code (CC) register in the bit position of the memory location given by the destination address. M(longmem).bit <- CC.C

Instruction overview

mnem	dst	bit position	Affected condition flags						
			V	I1	H	I0	N	Z	C
BCCM	Mem	#pos	-	-	-	-	-	-	-

Detailed description

dst	pos = 0..7	Asm	cy	lgth	Op-code(s)					ST7
longmem	n =1+2*pos	BCCM $1000,#2	1	4	90	1n	MS	LS		

See also: LD, RCF, SCF

BCP Logical Bit Compare BCP

Syntax BCP A,src

Operation {N, Z} <= A AND src

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

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
BCP	A	Mem	-	-	-	-	N	Z	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	BCP A,#$55	1	2		A5	XX			✗
A	shortmem	BCP A,$10	1	2		B5	XX			✗
A	longmem	BCP A,$1000	1	3		C5	MS	LS		✗
A	(X)	BCP A,(X)	1	1		F5				✗
A	(shortoff,X)	BCP A,($10,X)	1	2		E5	XX			✗
A	(longoff,X)	BCP A,($1000,X)	1	3		D5	MS	LS		✗
A	(Y)	BCP A,(Y)	1	2	90	F5				✗
A	(shortoff,Y)	BCP A,($10,Y)	1	3	90	E5	XX			✗
A	(longoff,Y)	BCP A,($1000,Y)	1	4	90	D5	MS	LS		✗
A	(shortoff,SP)	BCP A,($10,SP)	1	2		15	XX			
A	[shortptr.w]	BCP A,[$10.w]	4	3	92	C5	XX			✗
A	[longptr.w]	BCP A,[$1000.w]	4	4	72	C5	MS	LS		
A	([shortptr.w],X)	BCP A,([$10.w],X)	4	3	92	D5	XX			✗
A	([longptr.w],X)	BCP A,([$1000.w],X)	4	4	72	D5	MS	LS		
A	([shortptr.w],Y)	BCP A,([$10.w],Y)	4	3	91	D5	XX			✗

See also: CP, TNZ

BCPL Bit Complement BCPL

Syntax BCPL dst, #pos (pos=0..7) e.g. BCPL PADR,#4

Operation dst(pos) <= 1 - dst(pos)

Description Complements the bit position in destination location. Leaves all other bits unchanged.

M(longmem).bit <- -M(longmem).bit

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
BCPL	Mem	-	-	-	-	-	-	-

Detailed description

dst	pos = 0..7	Asm	cy	lgth	Op-code(s)					ST7
longmem	n = 2*pos	BCPL $1000,#2	1	4	90	1n	MS	LS		

See also: CPL, BRES, BSET

BREAK Software break BREAK

Syntax

Operation

Description In debug mode, the CPU is stalled and can be restarted by the debugger. This instruction equals a NOP when the debugger is not connected.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
SIM	-	1	-	1	-	-	-

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	BREAK	1	1		8B				✗

BRES Bit Reset BRES

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

Operation dst <= dst AND COMPLEMENT (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	Affected condition flags						
			V	I1	H	I0	N	Z	C
BRES	Mem	#pos	-	-	-	-	-	-	-

Detailed description

dst	pos = 0..7	Asm	cy	lgth	Op-code(s)					ST7
longmem	n=1+2*pos	BRES $1000,#7	1	4	72	1n	MS	LS		

See also: BSET

BSET Bit Set BSET

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

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	Affected condition flags						
			V	I1	H	I0	N	Z	C
BSET	Mem	#pos	-	-	-	-	-	-	-

Detailed description

dst	pos = 0..7	Asm	cy	lgth	Op-code(s)					ST7
longmem	n=2*pos	BSET $1000,#1	1	4	72	1n	MS	LS		

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.: BTJFPADR,#3,skip

Operation PC = PC+lgth
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 represents a signed offset to be added to the current PC/instruction address (relative jump). This instruction is used for boolean variable manipulation, hardware register flag tests, or I/O polling. This instruction is fast, compact, and does not affect any registers. Very useful for boolean variable manipulation.

Instruction overview

mnem	dst	bit position	jump label	Affected condition flags						
				V	I1	H	I0	N	Z	C
BTJF	Mem	#pos	rel	-	-	-	-	-	-	C

C ⇒Tested bit is saved in the C flag.

Detailed description

dst	pos = 0..7	Asm	cy	lgth	Op-code(s)					ST7
longmem	n = 1+2*pos	BTJF $1000,#1,loop	2/3	5	72	0n	MS	LS	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+lgth
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 represents a signed offset to be added to the current PC/instruction address (relative jump). This instruction is used for boolean variable manipulation, hardware register flag tests, or I/O polling.

Instruction overview

mnem	dst	bit position	jump label	Affected condition flags						
				V	I1	H	I0	N	Z	C
BTJT	Mem	#pos	rel	-	-	-	-	-	-	C

C ⇒ Tested bit is saved in the C flag.

Detailed description

dst	pos = 0..7	Asm	cy	lgth	Op-code(s)					ST7
longmem	n= 2*pos	BTJT $1000,#1,loop	2/3	5	72	0n	MS	LS	XX	

See also: BTJF

CALL

CALL Subroutine (Absolute)

CALL

Operation

PC = PC+lgth
(SP--) = PCL
(SP--) = PCH
PC = dst

Description

The current PC register value is pushed onto the stack, then PC is loaded with the destination address in same section of memory. The CALL destination and the instruction following the CALL should be in the same section as PCE is not stacked. The corresponding RET instruction should be executed in the same section. This instruction should be used versus CALLR when developing a program.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
CALL	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
longmem	CALL $1000	4	3		CD	MS	LS		✗
(X)	CALL(X)	4	1		FD				✗
(shortoff,X)	CALL($10,X)	4	2		ED	XX			✗
(longoff,X)	CALL($1000,X)	4	3		DD	MS	LS		✗
(Y)	CALL(Y)	4	2	90	FD				✗
(shortoff,Y)	CALL($10,Y)	4	3	90	ED	XX			✗
(longoff,Y)	CALL($1000,Y)	4	4	90	DD	MS	LS		✗
[shortptr.w]	CALL[$10.w]	6	3	92	CD	XX			✗
[longptr.w]	CALL[$1000.w]	6	4	72	CD	MS	LS		
([shortptr.w],X)	CALL([$10.w],X)	6	3	92	DD	XX			✗
([longptr.w],X)	CALL([$1000.w],X)	6	4	72	DD	MS	LS		
([shortptr.w],Y)	CALL([$10.w],Y)	6	3	91	DD	XX			✗

See also: RET, CALLR, CALLF

CALLF CALL Far Subroutine CALLF

Syntax CALLF dst e.g. CALLF label

Operation
PC = PC+lgth
(SP--) = PCL
(SP--) = PCH
(SP--) = PCE
PC = dst

Description The current PC register value is pushed onto the stack, then PC is loaded with the destination address.This instruction is used with extended memory addresses. For safe memory usage, a function which crosses sections must be called by CALLF.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
CALLF	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
extmem	CALLF $35AA00	5	4		8D	ExtB	MS	LS	
[longptr.e]	CALLF [$2FFC.e]	8	4	92	8D	MS	LS		

See also: RETF, CALL, JPF

CALLR CALL Subroutine Relative CALLR

Syntax CALLR dst e.g. CALLR chk_pol

Operation
PC = PC+lgth
(SP--) = PCL
(SP--) = PCH
PC = PC + dst

Description The current PC register value is pushed onto the stack, then PC is loaded with the relative destination address. This instruction is used, once a program is debugged, to shrink the overall program size. The CALLR destination and the corresponding RET instruction address must be in the same section, as PCE is not stacked.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
CALLR	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
shortmem	CALLR $10	4	2		AD	XX			✗

See also: CALL, RET

CCF Complement Carry Flag CCF

Syntax CCF

Operation CC.C <- $\overline{CC.C}$

Description Complements the Carry flag of the Condition Code (CC) register.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
CCF	-	-	-	-	-	-	C

C =$\overline{C}$,
Complements the carry flag of the CC register.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	CCF	1	1		8C				

See also: RCF, SCF

CLR Clear CLR

Syntax CLR dst e.g. CLR A

Operation dst <= 00

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

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
CLR	Mem	-	-	-	-	0	1	-
CLR	A					0	1	

N: 0

Cleared

Z: 1

Set

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	CLR A	1	1		4F				✗
shortmem	CLR $10	1	2		3F	XX			✗
longmem	CLR $1000	1	4	72	5F	MS	LS		
(X)	CLR (X)	1	1		7F				✗
(shortoff.X)	CLR ($10,X)	1	2		6F	XX			✗
(longoff,X)	CLR ($1000,X)	1	4	72	4F	MS	LS		
(Y)	CLR (Y)	1	2	90	7F				✗
(shortoff,Y)	CLR ($10,Y)	1	3	90	6F	XX			✗
(longoff,Y)	CLR ($1000,Y)	1	4	90	4F	MS	LS		
(shortoff,SP)	CLR ($10,SP)	1	2		0F	XX			
[shortptr.w]	CLR [$10]	4	3	92	3F	XX			✗
[longptr.w]	CLR [$1000].w	4	4	72	3F	MS	LS		
([shortptr.w],X)	CLR ([$10],X)	4	3	92	6F	XX			✗
([longptr.w].X]	CLR ([$1000.w],X)	4	4	72	6F	MS	LS		
([shortptr.w],Y)	CLR ([$10],Y)	4	3	91	6F	XX			✗

See also: LD

CLRW Clear word CLRW

Syntax CLRW dst e.g. CLRW X

Operation dst <= 00

Description The destination is forced to 0000 value. The destination is an index register.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
CLRW	X	-	-	-	-	0	1	-
CLRW	Y	-	-	-	-	0	1	-

N: 0

Cleared

Z: 1

Set

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	CLRW X	1	1		5F				
Y	CLRW Y	1	2	90	5F				

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 flags of Condition Code (CC) register are updated according to the result.The destination is a register, and the source is a memory or data byte. This instruction generally is used just before a conditional jump instruction.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
CP	Reg	Mem	V	-	-	-	N	Z	C

V ⇒ $(\overline{A7}.M7 + \overline{A7}.R7 + A7.M7.R7) \oplus (\overline{A6}.M6 + \overline{A6}.R6 + A6.M6.R6)$
Set if the signed subtraction of the destination (dst) value from the source (src) value generates a signed overflow (signed result cannot be represented on 8 bits).

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ $(\overline{A7}.M7 + \overline{A7}.R7 + A7.M7.R7)$
Set if the unsigned value of the contents of source (src) is larger than the unsigned value of the destination (dst), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	CP A,#$10	1	2		A1	XX			✗
A	shortmem	CP A,$10	1	2		B1	XX			✗
A	longmem	CP A,$1000	1	3		C1	MS	LS		✗
A	(X)	CP A,(X)	1	1		F1				✗
A	(shortoff,X)	CP A,($10,X)	1	2		E1	XX			✗
A	(longoff,X)	CP A,($1000,X)	1	3		D1	MS	LS		✗
A	(Y)	CP A,(Y)	1	2	90	F1				✗
A	(shortoff,Y)	CP A,($10,Y)	1	3	90	E1	XX			✗
A	(longoff,Y)	CP A,($1000,Y)	1	4	90	D1	MS	LS		✗
A	(shortoff,SP)	CP A,($10,SP)	1	2		11	XX			
A	[shortptr.w]	CP A,[$10.w]	4	3	92	C1	XX			✗
A	[longptr.w]	CP A,[$1000.w]	4	4	72	C1	MS	LS		
A	([shortptr.w],X)	CP A,([$10.w],X)	4	3	92	D1	XX			✗
A	([longptr.w],X)	CP A,([$1000.w],X)	4	4	72	D1	MS	LS		
A	([shortptr.w],Y)	CP A,([$10.w],Y)	4	3	91	D1	XX			✗

See also: CPW, TNZ, BCP

CPW Compare word CPW

Syntax CPW dst,src e.g. CPW Y,(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 flags of Condition Code (CC) register are updated according to the result. The destination is an index register, and the source is a memory or data word. This instruction generally is used just before a conditional jump instruction.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
CPW	Reg	Mem	V	-	-	-	N	Z	C

V ⇒ $(\overline{X15}.M15 + \overline{X15}.R15 + X15.M15.R15) \oplus (\overline{X14}.M14 + \overline{X14}.R14 + X14.M14.R14)$
Set if the signed subtraction of the destination (dst) value from the source (src) value generates a signed overflow (signed result cannot be represented on 16 bits).

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ $(\overline{X15}.M15 + \overline{X15}.R15 + X15.M15.R15)$
Set if the unsigned value of the contents of source (src) is larger than the unsigned value of the destination (dst), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	#word	CPW X,#$10	2	3		A3	MS	LS		✗
X	shortmem	CPW X,$10	2	2		B3	XX			✗
X	longmem	CPW X,$1000	2	3		C3	MS	LS		✗
X	(Y)	CPW X,(Y)	2	2	90	F3				✗
X	(shortoff,Y)	CPW X,($10,Y)	2	3	90	E3	XX			✗
X	(longoff,Y)	CPW X,($1000,Y)	2	4	90	D3	MS	LS		✗
X	(shortoff,SP)	CPW X,($10,SP)	2	2		13	XX			
X	[shortptr.w]	CPW X,[$10.w]	5	3	92	C3	XX			✗
X	[longptr.w]	CPW X,[$1000.w]	5	4	72	C3	MS	LS		
X	([shortptr.w],Y)	CPW X,([$10.w],Y)	5	3	91	D3	XX			✗

CPW detailed description (Continued)

dst	src	Asm	cy	lgth	Op-code(s)					ST7
Y	#word	CPW Y,#$10	2	4	90	A3	MS	LS		✗
Y	shortmem	CPW Y,$10	2	3	90	B3	XX			✗
Y	longmem	CPW Y,$1000	2	4	90	C3	MS	LS		✗
Y	(X)	CPW Y,(X)	2	1		F3				✗
Y	(shortoff,X)	CPW Y,($10,X)	2	2		E3	XX			✗
Y	(longoff,X)	CPW Y,($1000,X)	2	3		D3	MS	LS		✗
Y	[shortptr.w]	CPW Y,[$10.w]	5	3	91	C3	XX			✗
Y	([shortptr.w],X)	CPW Y,([$10.w],X)	5	3	92	D3	XX			✗
Y	([longptr.w],X)	CPW Y,([$1000.w],X)	5	4	72	D3	MS	LS		

Note: CPW Y, (shortoff, SP) is not implemented, but can be emulated through a macro using EXGW X,Y & CPW X, (shortoff, SP)

See also: CP, TNZW, BCP

CPL Logical 1's 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 to the destination byte. The destination is either a memory byte or a register. This instruction is compact, and does not affect any registers when used with RAM variables.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
CPL	Mem	-	-	-	-	N	Z	1
CPL	Reg	-	-	-	-	N	Z	1

N ⇒ R7

Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$

Set if the result is zero (0x00), cleared otherwise.

C ⇒ 1

Set.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	CPL A	1	1		43				✗
shortmem	CPL$10	1	2		33	XX			✗
longmem	CPL$1000	1	4	72	53	MS	LS		
(X)	CPL(X)	1	1		73				✗
(shortoff.X)	CPL($10,X)	1	2		63	XX			✗
(longoff,X)	CPL($1000,X)	1	4	72	43	MS	LS		
(Y)	CPL(Y)	1	2	90	73				✗
(shortoff,Y)	CPL($10,Y)	1	3	90	63	XX			✗
(longoff,Y)	CPL($1000,Y)	1	4	90	43	MS	LS		
(shortoff,SP)	CPL($10,SP)	1	2		03	XX			✗
[shortptr.w]	CPL[$10]	4	3	92	33	XX			✗
[longptr.w]	CPL[$1000].w	4	4	72	33	MS	LS		
([shortptr.w],X)	CPL([$10],X)	4	3	92	63	XX			✗
([longptr.w].X]	CPL([$1000.w],X)	4	4	72	63	MS	LS		
([shortptr.w],Y)	CPL([$10],Y)	4	3	91	63	XX			✗

See also: NEG, XOR, AND, OR

CPLW Logical 1's Complement Word CPLW

Syntax CPLW dst e.g. CPLW X

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

Description The destination index register is read, then each bit is toggled (inverted) and the result is written back to the destination index register.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
CPLW	Reg	-	-	-	-	N	Z	1

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ 1
Set

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	CPLW X	2	1		53				✗
Y	CPWL Y	2	2	90	53				✗

See also: CPL, NEGW, XOR, AND, OR

DEC Decrement DEC

Syntax DEC dst

Operation dst <= dst - 1

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

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
DEC	Mem	V	-	-	-	N	Z	-
DEC	Reg	V	-	-	-	N	Z	-

V ⇒ (A7.M7 + M7.$\overline{R7}$ + $\overline{R7}$.A7) ⊕ (A6.M6 + M6.$\overline{R6}$ + $\overline{R6}$.A6)
Set if the signed operation generates an overflow, cleared otherwise.

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}$.$\overline{R6}$.$\overline{R5}$.$\overline{R4}$.$\overline{R3}$.$\overline{R2}$.$\overline{R1}$.$\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)				ST7
A	DEC A	1	1		4A			✗
shortmem	DEC $10	1	2		3A	XX		✗
longmem	DEC $1000	1	4	72	5A	MS	LS	
(X)	DEC(X)	1	1		7A			✗
(shortoff.X)	DEC($10,X)	1	2		6A	XX		✗
(longoff,X)	DEC($1000,X)	1	4	72	4A	MS	LS	
(Y)	DEC(Y)	1	2	90	7A			✗
(shortoff,Y)	DEC($10,Y)	1	3	90	6A	XX		✗
(longoff,Y)	DEC($1000,Y)	1	4	90	4A	MS	LS	
(shortoff,SP)	DEC($10,SP)	1	2		0A	XX		
[shortptr.w]	DEC[$10]	4	3	92	3A	XX		✗
[longptr.w]	DEC[$1000].w	4	4	72	3A	MS	LS	
([shortptr.w],X)	DEC([$10],X)	4	3	92	6A	XX		✗
([longptr.w].X]	DEC([$1000.w],X)	4	4	72	6A	MS	LS	
([shortptr.w],Y)	DEC([$10],Y)	4	3	91	6A	XX		✗

See also: DECW, INC

DECW Decrement word DECW

Syntax DECW dst

Operation dst <= dst - 1

Description The value of the destination index register is decremented by one.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
DECW	Reg	V	-	-	-	N	Z	-

V ⇒ $(A15.M15 + M15.\overline{R15} + \overline{R15}.A15) \oplus (A14.M14 + M14.\overline{R14} + \overline{R14}.A14)$
Set if the signed operation generates an overflow, cleared otherwise.

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)				ST7
X	DECW X	1	1		5A			
Y	DECW Y	1	2	90	5A			

See also: INCW, DEC

DIV Divide (unsigned) DIV

Syntax DIV dst,A e.g. DIV X,A

Operation dst <= dst / A (Quotient) A <= dst%A (Remainder)

Description Divides a 16-bit unsigned value, dividend, contained in an index register (X or Y) by an 8-bit value, divisor, contained in A. The quotient is placed in the same index register and the remainder is placed in A.

The register values are unchanged in the case of a division by zero.

Note: *Note: This instruction is interruptible, generating a latency of 1 cycle only.*

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
DIV	X	A	0	-	0	-	0	Z	C
DIV	Y	A	0	-	0	-	0	Z	C

V ⇒ 0

Reset.

H ⇒ 0

Reset.

N ⇒ 0

Reset.

Z ⇒ $\overline{Q15}.\overline{Q14}.\overline{Q13}.\overline{Q12}.\overline{Q11}.\overline{Q10}.\overline{Q9}.\overline{Q8}.\overline{Q7}.\overline{Q6}.\overline{Q5}.\overline{Q4}.\overline{Q3}.\overline{Q2}.\overline{Q1}.\overline{Q0}$
Set if the quotient is zero (0x0000), cleared otherwise.

C ⇒ $\overline{A7}.\overline{A6}.\overline{A5}.\overline{A4}.\overline{A3}.\overline{A2}.\overline{A1}.\overline{A0}$
Set if division by 0, cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	A	DIV X,A	2 to 17	1		62				
Y	A	DIV Y,A	2 to 17	2	90	62				

See also: DIVW, ADD, ADC, SUB, SBC, MUL

DIVW Divide word (unsigned) DIVW

Operation X <= X / Y (Quotient) Y <= X%Y (Remainder)

Description Divides a 16-bit unsigned value, dividend, contained in X register by a 16-bit value, divisor, contained in Y. The quotient is placed in the X register and the remainder is placed in Y register.

The quotient and remainder values are indeterminate in the case of a division by zero.

Note: This instruction is interruptible, generating a latency of 1 cycle only.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
DIV	X	Y	0	-	0	-	0	Z	C

V ⇒ 0

Reset

H ⇒ 0

Reset

N ⇒ 0

Reset

Z ⇒ $\overline{Q15}.\overline{Q14}.\overline{Q13}.\overline{Q12}.\overline{Q11}.\overline{Q10}.\overline{Q9}.\overline{Q8}.\overline{Q7}.\overline{Q6}.\overline{Q5}.\overline{Q4}.\overline{Q3}.\overline{Q2}.\overline{Q1}.\overline{Q0}$
Set if the quotient is zero (0x0000), cleared otherwise.

C ⇒ $\overline{Y15}.\overline{Y14}.\overline{Y13}.\overline{Y12}.\overline{Y11}.\overline{Y10}.\overline{Y9}.\overline{Y8}.\overline{Y7}.\overline{Y6}.\overline{Y5}.\overline{Y4}.\overline{Y3}.\overline{Y2}.\overline{Y1}.\overline{Y0}$
Set if division by 0, cleared otherwise.

Detailed description

dst	src	Asm
X	Y	DIV X,Y

cy	lgth
2 to 17	1

Op-code(s)					ST7
	65				

See also: ADD, ADC, SUB, SBC, MUL, DIV

EXG Exchange register contents EXG

Syntax EXG dst, src e.g. EXG A, XL

Operation dst <=> src

src <= dst

dst<= src

Description Exchanges the contents of registers specified in the instruction as shown below.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
EXG	A	XL	-	-	-	-	-	-	-
EXG	A	YL	-	-	-	-	-	-	-
EXG	A	Mem	-	-	-	-	-	-	-

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	XL	EXG A,XL	1	1		41				
A	YL	EXG A,YL	1	1		61				
A	longmem	EXG A,$1000	3	3		31	MS	LS		

See also: EXGW, LD

EXGW Exchange Index register contents EXGW

Syntax EXG dst, src e.g. EXGW X, Y

Operation dst <=> src

src <= dst
dst<= src

Description Exchanges the contents of registers specified in the instruction as shown below.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
EXGW	X	Y	-	-	-	-	-	-	-

Detailed description

dst	src	Asm
X	Y	EXGW X,Y

cy	lgth
1	1

Op-code(s)					ST7
	51				

See also: EXG, LDW

HALT

HALT Oscillator (CPU + Peripherals)

HALT

Syntax HALT

Operation I1 = 1, I0 = 0, The oscillator is stopped till an interrupt occurs.

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 microcontroller resumes program execution after an external interrupt or reset, by restarting the oscillator, and then, fetching the corresponding external interrupt, which is an I/O interrupt, a specific peripheral interrupt, or the reset vector.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
HALT	-	1	-	0	-	-	-

I1: 1

Set.

I0: 0

Cleared.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	HALT	10	1		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 to the destination byte. The destination is either a memory byte or a register. This instruction is compact, and does not affect any registers when used with RAM variables.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
INC	Mem	V	-	-	-	N	Z	-
INC	A	V	-	-	-	N	Z	-

V ⇒ $(A7.M7 + M7.\overline{R7} + \overline{R7}.A7) \oplus (A6.M6 + M6.\overline{R6} + \overline{R6}.A6)$
Set if the signed operation generates an overflow, cleared otherwise.

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	INC A	1	1		4C				✗
shortmem	INC $10	1	2		3C	XX			✗
longmem	INC $1000	1	4	72	5C	MS	LS		
(X)	INC (X)	1	1		7C				✗
(shortoff.X)	INC ($10,X)	1	2		6C	XX			✗
(longoff,X)	INC ($1000,X)	1	4	72	4C	MS	LS		
(Y)	INC (Y)	1	2	90	7C				✗
(shortoff,Y)	INC ($10,Y)	1	3	90	6C	XX			✗
(longoff,Y)	INC ($1000,Y)	1	4	90	4C	MS	LS		
(shortoff,SP)	INC ($10,SP)	1	2		0C	XX			
[shortptr.w]	INC [$10]	4	3	92	3C	XX			✗
[longptr.w]	INC [$1000].w	4	4	72	3C	MS	LS		
([shortptr.w],X)	INC ([$10],X)	4	3	92	6C	XX			✗
([longptr.w].X]	INC ([$1000.w],X)	4	4	72	6C	MS	LS		
([shortptr.w],Y)	INC ([$10],Y)	4	3	91	6C	XX			✗

See also: INCW, DEC

INCW Increment word INCW

Syntax INCW dst e.g. INCW X

Operation dst <= dst + 1

Description The destination index register value is incremented by one.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
INCW	Reg	V	-	-	-	N	Z	-

V ⇒ (A15.M15 + M15.$\overline{R15}$ + $\overline{R15}$.A15) ⊕ (A14.M14 + M14.$\overline{R14}$ + $\overline{R14}$.A14)
Set if the signed operation generates an overflow, cleared otherwise.

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}$.$\overline{R14}$.$\overline{R13}$.$\overline{R12}$.$\overline{R11}$.$\overline{R10}$.$\overline{R9}$.$\overline{R8}$.$\overline{R7}$.$\overline{R6}$.$\overline{R5}$.$\overline{R4}$.$\overline{R3}$.$\overline{R2}$.$\overline{R1}$.$\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	INCW X	1	1		5C				
Y	INCW Y	1	2	90	5C				

See also: INC, DECW

INT Interrupt INT

Syntax INT dst

Operation PC <= dst

Description This instruction is used only in the interrupt vector table.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
INT	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
extmem	INT $2FFFFC	2	4		82	ExtB	MS	LS	

See also: JP, JPF, CALLF

IRET — Interrupt Return — IRET

Syntax IRET

Operation
CC = (++SP)
A = (++SP)
XH = (++SP)
XL = (++SP)
YH = (++SP)
YL = (++SP)
PCE = (++SP)
PCH = (++SP)
PCL = (++SP)

Description Placed at the end of an interrupt routine, returns to the original program context before the interrupt occurred. All registers, which have been saved/pushed onto the stack are restored/popped. The I bit will be reset if the corresponding bit stored on the stack is zero.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
IRET	V	I1	H	I0	N	Z	C

Condition flags set or reset according to the first byte pulled from the stack.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	IRET	11	1		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 destination address in same section of memory. 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	Affected condition flags						
		V	I1	H	I0	N	Z	C
JP	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
longmem	JP $1000	1	3		CC	MS	LS		✗
(X)	JP(X)	1	1		FC				✗
(shortoff,X)	JP($10,X)	1	2		EC	XX			✗
(longoff,X)	JP($1000,X)	1	3		DC	MS	LS		✗
(Y)	JP(Y)	1	2	90	FC				✗
(shortoff,Y)	JP($10,Y)	2	3	90	EC	XX			✗
(longoff,Y)	JP($1000,Y)	2	4	90	DC	MS	LS		✗
[shortptr.w]	JP[$10.w]	5	3	92	CC	XX			✗
[longptr.w]	JP[$1000.w]	5	4	72	CC	MS	LS		
([shortptr.w],X)	JP([$10.w],X)	5	3	92	DC	XX			✗
([longptr.w],X)	JP([$1000.w],X)	5	4	72	DC	MS	LS		
([shortptr.w],Y)	JP([$10.w],Y)	5	3	91	DC	XX			✗

See also: JRT

JPF — Jump far — JPF

Syntax JPF dst e.g.:JPF test

Operation PC <= dst

Description The unconditional jump simply replaces the content of the PC by a destination with an extended address. Control then passes to the statement addressed by the program counter. For safe memory usage, this instruction must be used, when the operation crosses a memory section.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
JPF	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
extmem	JPF $2FFFFC	2	4		AC	ExtB	MS	LS	
[longptr.e]	JPF [$2FFC.e]	6	4	92	AC	MS	LS		

See also: JP, CALLF

JRA Jump Relative Always JRA

Syntax JRA dst e.g. JRA loop

Operation PC = PC+lgth

PC <= PC + dst, if Condition is True

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. Else, the program continues normally.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
JRA	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
shortoff	JRA $2B	2	2		20	XX			✗

See also: JP

JRxx — Conditional Jump Relative Instruction — JRxx

Syntax JRxx dst e.g. JRxx loop

Operation PC = PC+lgth

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	Affected condition flags						
		V	I1	H	I0	N	Z	C
JRxx	Mem	-	-	-	-	-	-	-

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	90	29
JRIH	Interrupt Line is High			90	2F
JRIL	Interrupt Line is Low			90	2E
JRM	Interrupt Mask		I = 1	90	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	90	28
JRNM	Not Interrupt Mask		I = 0	90	2C
JRNV	Not Overflow		V = 0		28
JRPL	Plus	>= 0	N = 0		2A
JRSGE	Signed Greater or Equal	>=	(N XOR V) = 0		2E
JRSGT	Signed Greater Than	>	(Z OR (N XOR V)) = 0		2C
JRSLE	Signed Lower or Equal	<=	(Z OR (N XOR V)) = 1		2D
JRSLT	Signed Lower Than	<	(N XOR V) = 1		2F
JRT	True		True		20
JRUGE	Unsigned Greater or Equal		C = 0		24
JRUGT	Unsigned Greater Than	>	C = 0 and Z = 0		22
JRULE	Unsigned Lower or Equal	<=	C = 1 or Z = 1		23
JRC	Carry		C = 1		25
JRULT	Unsigned Lower Than		C = 1		25
JRV	Overflow		V = 1		29

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
shortoff	JRxx $15	1/2	2		Op-code	XX			✗
shortoff	JRxx $15	1/2	3	90	Op-code	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. The dst and src can be a register, a byte (low/high) of an index register or a memory/data byte. When half of an index register is loaded, the other half remains unchanged.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
LD	Reg	Mem	-	-	-	-	N	Z	-
LD	Mem	Reg	-	-	-	-	N	Z	-
LD	Reg	Reg	-	-	-	-	-	-	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	LD A,#$55	1	2		A6	XX			✗
A	shortmem	LD A,$50	1	2		B6	XX			✗
A	longmem	LD A,$5000	1	3		C6	MS	LS		✗
A	(X)	LD A,(X)	1	1		F6				✗
A	(shortoff,X)	LD A,($50,X)	1	2		E6	XX			✗
A	(longoff,X)	LD A,($5000,X)	1	3		D6	MS	LS		✗
A	(Y)	LD A,(Y)	1	2	90	F6				✗
A	(shortoff,Y)	LD A,($50,Y)	1	3	90	E6	XX			✗
A	(longoff,Y)	LD A,($5000,Y)	1	4	90	D6	MS	LS		✗
A	(shortoff,SP)	LD A,($50,SP)	1	2		7B	XX			
A	[shortptr.w]	LD A,[$50.w]	4	3	92	C6	XX			✗
A	[longptr.w]	LD A,[$5000.w]	4	4	72	C6	MS	LS		
A	([shortptr.w],X)	LD A,([$50.w],X)	4	3	92	D6	XX			✗
A	([longptr.w],X)	LD A,([$5000.w],X)	4	4	72	D6	MS	LS		
A	([shortptr.w],Y)	LD A,([$50.w],Y)	4	3	91	D6	XX			✗

LD detailed description (Continued)

dst	src	Asm	cy	lgth	Op-code(s)					ST7
shortmem	A	LD $50,A	1	2		B7	XX			✗
longmem	A	LD $5000,A	1	3		C7	MS	LS		✗
(X)	A	LD (X),A	1	1		F7				✗
(shortoff,X)	A	LD ($50,X),A	1	2		E7	XX			✗
(longoff,X)	A	LD ($5000,X),A	1	3		D7	MS	LS		✗
(Y)	A	LD (Y),A	1	2	90	F7				✗
(shortoff,Y)	A	LD ($50,Y),A	1	3	90	E7	XX			✗
(longoff,Y)	A	LD ($5000,Y),A	1	4	90	D7	MS	LS		✗
(shortoff,SP)	A	LD ($50,SP),A	1	2		6B	XX			
[shortptr.w]	A	LD [$50.w],A	4	3	92	C7	XX			✗
[longptr.w]	A	LD [$5000.w],A	4	4	72	C7	MS	LS		
([shortptr.w], X)	A	LD ([$50.w],X),A	4	3	92	D7	XX			✗
([longptr.w],X)	A	LD ([$5000.w],X),A	4	4	72	D7	MS	LS		
([shortptr.w], Y)	A	LD ([$50.w],Y),A	4	3	91	D7	XX			✗

dst	src	Asm	cy	lgth	Op-code(s)					ST7
XL	A	LD XL,A	1	1		97				✗
A	XL	LD A,XL	1	1		9F				✗
YL	A	LD YL,A	1	2	90	97				✗
A	YL	LD A,YL	1	2	90	9F				✗
XH	A	LD XH,A	1	1		95				
A	XH	LD A,XH	1	1		9E				
YH	A	LD YH,A	1	2	90	95				
A	YH	LD A,YH	1	2	90	9E				

See also: LDW, LDF, CLR

LDF Load Far LDF

Syntax LDF dst,src e.g. LDF A,($555555,X)

Operation dst <= src

Description Load the destination byte with the source byte. The dst and src can be a memory location or accumulator register.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
LDF	A	Mem	-	-	-	-	N	Z	-
LDF	Mem	A	-	-	-	-	N	Z	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	extmem	LDF A, $500000	1	4		BC	ExtB	MS	LS	
A	(extoff,X)	LDF A,($500000,X)	1	4		AF	ExtB	MS	LS	
A	(extoff,Y)	LDF A,($500000,Y)	1	5	90	AF	ExtB	MS	LS	
A	([longptr.e],X)	LDF A,([$5000.e],X)	5	4	92	AF	MS	LS		
A	([longptr.e],Y)	LDF A,([$5000.e],Y)	5	4	91	AF	MS	LS		
A	[longptr.e]	LDF A,[$5000.e]	5	4	92	BC	MS	LS		

dst	src	Asm	cy	lgth	Op-code(s)					ST7
extmem	A	LDF $500000,A	1	4		BD	ExtB	MS	LS	
(extoff,X)	A	LDF ($500000,X),A	1	4		A7	ExtB	MS	LS	
(extoff,Y)	A	LDF ($500000,Y),A	1	5	90	A7	ExtB	MS	LS	
([longptr.e],X)	A	LDF ([$5000.e],X),A	4	4	92	A7	MS	LS		
([longptr.e],Y)	A	LDF ([$5000.e],Y),A	4	4	91	A7	MS	LS		
[longptr.e]	A	LDF [$5000.e],A	4	4	92	BD	MS	LS		

See also: LD, CALLF

LDW Load word LDW

Syntax LDW dst,src e.g. LDW X,#$1500

Operation dst <= src

Description Load the destination word (16-bit value) with the source word. The dst and src can be a 16-bit register (X, Y or SP) or a memory/data 16-bit value.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
LD	Reg	Mem	-	-	-	-	N	Z	-
LD	Mem	Reg	-	-	-	-	N	Z	-
LD	Reg	Reg	-	-	-	-	-	-	-
LD	SP	Reg	-	-	-	-	-	-	-
LD	Reg	SP	-	-	-	-	-	-	-

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	#word	LDW X,#$55AA	2	3		AE	MS	LS		✗
X	shortmem	LDW X,$50	2	2		BE	XX			✗
X	longmem	LDW X,$5000	2	3		CE	MS	LS		✗
X	(X)	LDW X,(X)	2	1		FE				✗
X	(shortoff,X)	LDW X,($50,X)	2	2		EE	XX			✗
X	(longoff,X)	LDW X,($5000,X)	2	3		DE	MS	LS		✗
X	(shortoff,SP)	LDW X,($50,SP)	2	2		1E	XX			
X	[shortptr.w]	LDW X,[$50.w]	5	3	92	CE	XX			✗
X	[longptr.w]	LDW X,[$5000.w]	5	4	72	CE	MS	LS		
X	([shortptr.w],X)	LDW X,([$50.w],X)	5	3	92	DE	XX			✗
X	([longptr.w],X)	LDW X,([$5000.w],X)	5	4	72	DE	MS	LS		

dst	src	Asm	cy	lgth	Op-code(s)					ST7
shortmem	X	LDW $50,X	2	2		BF	XX			✗
longmem	X	LDW $5000,X	2	3		CF	MS	LS		✗
(X)	Y	LDW (X),Y	2	1		FF				
(shortoff,X)	Y	LDW ($50,X),Y	2	2		EF	XX			
(longoff,X)	Y	LDW ($5000,X),Y	2	3		DF	MS	LS		

LDW detailed description (Continued)

dst	src	Asm	cy	lgth	Op-code(s)					ST7
(shortoff,SP)	X	LDW ($50,SP),X	2	2		1F				
[shortptr.w]	X	LDW [$50.w],X	5	3	92	CF	XX			✗
[longptr.w]	X	LDW [$5000.w],X	5	4	72	CF	MS	LS		
([shortptr.w],X)	Y	LDW ([$50.w],X),Y	5	3	92	DF	XX			✗
([longptr.w],X)	Y	LDW ([$5000.w],X),Y	5	4	72	DF	MS	LS		

dst	src	Asm	cy	lgth	Op-code(s)					ST7
Y	#word	LDW Y,#$55AA	2	4	90	AE	MS	LS		✗
Y	shortmem	LDW Y,$50	2	3	90	BE	XX			✗
Y	longmem	LDW Y,$5000	2	4	90	CE	MS	LS		✗
Y	(Y)	LDW Y,(Y)	2	2	90	FE				✗
Y	(shortoff,Y)	LDW Y,($50,Y)	2	3	90	EE	XX			✗
Y	(longoff,Y)	LDW Y,($5000,Y)	2	4	90	DE	MS	LS		✗
Y	(shortoff,SP)	LDW Y,($50,SP)	2	2		16	XX			
Y	[shortptr.w]	LDW Y,[$50.w]	5	3	91	CE	XX			✗
Y	([shortptr.w],Y)	LDW Y,([$50.w],Y)	5	3	91	DE	XX			✗

dst	src	Asm	cy	lgth	Op-code(s)					ST7
shortmem	Y	LDW $50,Y	2	3	90	BF	XX			✗
longmem	Y	LDW $5000,Y	2	4	90	CF	MS	LS		✗
(Y)	X	LDW (Y),X	2	2	90	FF				✗
(shortoff,Y)	X	LDW ($50,Y),X	2	3	90	EF	XX			✗
(longoff,Y)	X	LDW ($5000,Y),X	2	4	90	DF	MS	LS		✗
(shortoff,SP)	Y	LDW ($50,SP),Y	2	2		17	XX			
[shortptr.w]	Y	LDW [$50.w],Y	5	3	91	CF	XX			✗
([shortptr.w],Y)	X	LDW ([$50.w],Y),X	5	3	91	DF	XX			✗

dst	src		cy	lgth	Op-code(s)					ST7
Y	X	LDW Y,X	1	2	90	93				✗
X	Y	LDW X,Y	1	1		93				✗
X	SP	LDW X,SP	1	1		96				✗
SP	X	LDW SP,X	1	1		94				✗
Y	SP	LDW Y,SP	1	2	90	96				✗
SP	Y	LDW SP,Y	1	2	90	94				✗

Note: LDW Y,[longptr.w] and LDW [longptr.w],Y are not implemented. They can be emulated using EXGW X,Y.

See also: LD, CLRW

MOV Move MOV

Syntax MOV dst,src e.g. MOV $80,#$AA

Operation dst<= src

Description Moves a byte of data from a source address to a destination address. Data is examined as it is moved. The accumulator is not affected.
There are 3 addressing modes for the MOV instruction:

- An immediate byte to a direct memory location
- A direct memory location to another direct memory location (from $00 to $FF)
- A direct memory location to another direct memory location (from $0000 to $FFFF)

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
MOV	Mem	Imm	-	-	-	-	-	-	-
MOV	Mem	Mem	-	-	-	-	-	-	-

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)						ST7
longmem	#byte	MOV $8000, #$AA	1	4		35	XX	MS	LS		
shortmem	shortmem	MOV $80,$10	1	3		45	XX2	XX1			
longmem	longmem	MOV $8000,$1000	1	5		55	MS2	LS2	MS1	LS1	

See also: LD, EXG

MUL Multiply (unsigned) MUL

Syntax MUL dst,src e.g. MUL X,A

Operation dst:src <= dst x src

Description Multiplies the 8-bit value in index register, low byte, (XL or YL) by the 8-bit value in the accumulator to obtain a 16-bit unsigned result in the index register. After the operation, index register contains the 16-bit result. The accumulator remains unchanged. The initial value of the high byte of the index register (XH or YH) is ignored.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
MUL	X	XL,A	-	-	0	-	-	-	0
MUL	Y	YL,A	-	-	0	-	-	-	0

C: 0

Cleared.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	A	MUL X,A	4	1		42				
Y	A	MUL Y,A	4	2	90	42				

See also: ADD, ADC, SUB, SBC

NEG Negate (Logical 2's 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	Affected condition flags						
		V	I1	H	I0	N	Z	C
NEG	Mem	V	-	-	-	N	Z	C
NEG	A	V	-	-	-	N	Z	C

V ⇒ R7.$\overline{R6}$.$\overline{R5}$.$\overline{R4}$.$\overline{R3}$.$\overline{R2}$.$\overline{R1}$.$\overline{R0}$
Set if there is an arithmetic overflow on the 8-bit representation of the result. The V bit will set when the content of "dst" was $80 (-128) prior to the NEG operation, cleared otherwise.

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}$.$\overline{R6}$.$\overline{R5}$.$\overline{R4}$.$\overline{R3}$.$\overline{R2}$.$\overline{R1}$.$\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ R7+R6+R5+R4+R3+R2+R1+R0
Set if a borrow in the implied subtraction from zero, cleared otherwise. The C bit will be set in all cases except when the contents of "dst" was $00 prior to the NEG operation.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	NEG A	1	1		40				✗
shortmem	NEG $F5	1	2		30	XX			✗
longmem	NEG $F5C2	1	4	72	50	MS	LS		
(X)	NEG(X)	1	1		70				✗
(shortoff,X)	NEG($F5,X)	1	2		60	XX			✗
(longoff,X)	NEG($F5C2,X)	1	4	72	40	MS	LS		
(Y)	NEG(Y)	1	2	90	70				✗
(shortoff,Y)	NEG($F5,Y)	1	3	90	60	XX			✗
(longoff,Y)	NEG($F5C2,Y)	1	4	90	40	MS	LS		
(shortoff,SP)	NEG($F5,SP)	1	2		00	XX			
[shortptr.w]	NEG($F5)	4	3	92	30	XX			✗

NEG detailed description (continued)

dst	Asm	cy	lgth	Op-code(s)					ST7
[longptr.w]	NEG($F5C2.w)	4	4	72	30	MS	LS		
([shortptr.w],X)	NEG([$F5],X)	4	3	92	60	XX			✗
([longptr.w],X)	NEG([$F5C2.w],X)	4	4	72	60	MS	LS		
([shortptr.w],Y)	NEG([$F5],Y)	4	3	91	60	XX			✗

See also: NEGW, CPL, AND, OR, XOR

NEGW Negate word (Logical 2's Complement) NEGW

Syntax NEGW dst e.g. NEGW X

Operation dst <= (dst XOR FFFF) + 1, or 0000 - dst

Description The destination word is read, then each bit is toggled (inverted), and the result is incremented before it is written at the destination word. The destination is an index register.

Instruction overview.

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
NEGW	X	V	-	-	-	N	Z	C
NEGW	Y	V	-	-	-	N	Z	C

V ⇒ R15.$\overline{R14}$.$\overline{R13}$.$\overline{R12}$.$\overline{R11}$.$\overline{R10}$.$\overline{R9}$.$\overline{R8}$.$\overline{R7}$.$\overline{R6}$.$\overline{R5}$.$\overline{R4}$.$\overline{R3}$.$\overline{R2}$.$\overline{R1}$.$\overline{R0}$
Set if there is an arithmetic overflow on the 16-bit representation. The V bit will set when the content of "dst" was $8000 prior to the NEGW operation, cleared otherwise.

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}$.$\overline{R14}$.$\overline{R13}$.$\overline{R12}$.$\overline{R11}$.$\overline{R10}$.$\overline{R9}$.$\overline{R8}$.$\overline{R7}$.$\overline{R6}$.$\overline{R5}$.$\overline{R4}$.$\overline{R3}$.$\overline{R2}$.$\overline{R1}$.$\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ R15+R14+R13+R12+R11+R10+R9+R8+R7+R6+R5+R4+R3+R2+R1+R0
Set if a borrow in the implied subtraction from zero, cleared otherwise. The C bit will be set in all cases except when the contents of "dst" was $0000 prior to the NEGW operation.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	NEGW X	2	1		50				
Y	NEGW Y	2	2	90	50				

See also: NEG, CPLW, AND, OR, XOR

NOP — No operation — NOP

Syntax NOP

Operation

Description This is a single byte instruction that does nothing. This instruction can be used either to disable an instruction, or to build a waiting delay.No register or memory contents are affected by this instruction

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
NOP	-	-	-	-	-	-	-

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	NOP	1	1		9D				✗

See also: JRF

OR — Logical OR — OR

Syntax OR A,src e.g. OR A,#%00110101

Operation A <= A OR src

Description The source byte, is logically ORed with the contents of the accumulator and the result is stored in the accumulator. The source is a memory or data byte.

Truth table

OR	0	1
0	0	1
1	1	1

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
OR	A	Mem	-	-	-	-	N	Z	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	OR A,#$55	1	2		AA	XX			✗
A	shortmem	OR A,$10	1	2		BA	XX			✗
A	longmem	OR A,$1000	1	3		CA	MS	LS		✗
A	(X)	OR A,(X)	1	1		FA				✗
A	(shortoff,X)	OR A,($10,X)	1	2		EA	XX			✗
A	(longoff,X)	OR A,($1000,X)	1	3		DA	MS	LS		✗
A	(Y)	OR A,(Y)	1	2	90	FA				✗
A	(shortoff,Y)	OR A,($10,Y)	1	3	90	EA	XX			✗
A	(longoff,Y)	OR A,($1000,Y)	1	4	90	DA	MS	LS		✗
A	(shortoff,SP)	OR A,($10,SP)	1	2		1A	XX			
A	[shortptr.w]	OR A,[$10.w]	4	3	92	CA	XX			✗
A	[longptr.w]	OR A,[$1000.w]	4	4	72	CA	MS	LS		
A	([shortptr.w],X)	OR A,([$10.w],X)	4	3	92	DA	XX			✗
A	([longptr.w],X)	OR A,([$1000.w],X)	4	4	72	DA	MS	LS		
A	([shortptr.w],Y)	OR A,([$1000],Y)	4	3	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. This instruction is used to restore a register/memory value.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
POP	A	-	-	-	-	-	-	-
POP	CC	V	I1	H	I0	N	Z	C
POP	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	POP A	1	1		84				✗
CC	POP CC	1	1		86				✗
longmem	POP $1000	1	3		32	MS	LS		

See also: PUSH, POPW

POPW Pop word from stack POPW

Syntax POPW dst e.g. POPW X

Operation dst_H <= (++SP)

dst_L <= (++SP)

Description Restore from the stack a data value which will be placed in dst location (index register). The stack pointer is incremented by two. This instruction is used to restore an index register value.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
POPW	X	-	-	-	-	-	-	-
POPW	Y	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	POPW X	2	1		85				✗
Y	POPW Y	2	2	90	85				✗

See also: PUSHW, POP

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 and a memory byte on to the stack.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
PUSH	A	-	-	-	-	-	-	-
PUSH	CC	-	-	-	-	-	-	-
PUSH	Imm	-	-	-	-	-	-	-
PUSH	Mem	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	PUSH A	1	1		88				✗
CC	PUSH CC	1	1		8A				✗
#byte	PUSH #$10	1	2		4B	XX			
longmem	PUSH $1000	1	3		3B	MS	LS		

See also: POP, PUSHW

PUSHW Push word onto the Stack PUSHW

Syntax PUSHW src e.g. PUSHW X

Operation (SP--) <= dst_L

(SP--) <= dst_H

Description Save the dst index register onto the stack. The stack pointer is decremented by two. Used to save an index register value onto the stack.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
PUSHW	X	-	-	-	-	-	-	-
PUSHW	Y	-	-	-	-	-	-	-

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	PUSHW X	2	1		89				✗
Y	PUSHW Y	2	2	90	89				✗

See also: POPW, PUSH

RCF Reset Carry Flag RCF

Syntax RCF

Operation C = 0

Description Clear the carry flag of the Condition Code (CC) register. May be used as a boolean user controlled flags.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
RCF	-	-	-	-	-	-	0

C: 0

Cleared.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	RCF	1	1		98				✗

See also: SCF, RVF

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 instruction of a subroutine in same section.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
RET	-	-	-	-	-	-	-

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	RET	4	1		81				✗

See also: CALL, CALLR

Note: Please note that the RET should be in the same section as the corresponding CALL.

RETF Far Return from subroutine RETF

Syntax RETF

Operation
PCE = (++SP)
PCH = (++SP)
PCL = (++SP)

Description Restore the PC from the stack then restore the Condition Code (CC) register. The stack pointer is incremented three times. This instruction is the last one of a subroutine in extended memory.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
RETF	-	-	-	-	-	-	-

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	RETF	5	1		87				

See also: CALLF

RIM — Reset Interrupt Mask/Enable Interrupt — RIM

Syntax RIM

Operation I1 = 1, I0 = 0

Description Clear the Interrupt mask of the Condition Code (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 WFI and HALT instructions.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
RIM	-	1	-	0	-	-	-

I1: 1

Set.

I0: 0

Cleared.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	RIM	1	1		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.This instruction shifts all bits of the register or memory, one place to the left, through the Carry bit. Bit 0 of the result is a copy of the CC.C value before the operation.

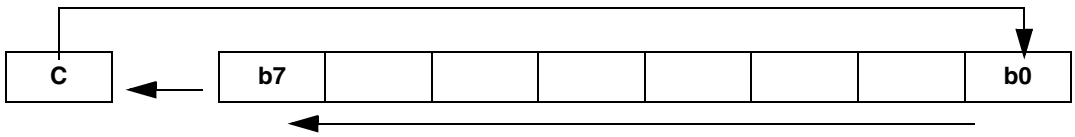
Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
RLC	Reg	-	-	-	-	N	Z	bit7
RLC	Mem	-	-	-	-	N	Z	bit7

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b7
Set if, before the shift, the MSB of register or memory was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	RLC A	1	1		49				✗
shortmem	RLC $10	1	2		39	XX			✗
longmem	RLC $1000	1	4	72	59	MS	LS		
(X)	RLC (X)	1	1		79				✗
(shortoff,X)	RLC ($10,X)	1	2		69	XX			✗
(longoff,X)	RLC ($1000,X)	1	4	72	49	MS	LS		
(Y)	RLC (Y)	1	2	90	79				✗
(shortoff,Y)	RLC ($10,Y)	1	3	90	69	XX			✗
(longoff,Y)	RLC ($1000,Y)	1	4	90	49	MS	LS		
(shortoff,SP)	RLC ($10,SP)	1	2		09	XX			✗
[shortptr.w]	RLC [$10]	4	3	92	39	XX			✗
[longptr.w]	RLC [$1000].w	4	4	72	39	MS	LS		
([shortptr.w],X)	RLC ([$10],X)	4	3	92	69	XX			✗
([longptr.w],X)	RLC ([$1000.w],X)	4	4	72	69	MS	LS		
([shortptr.w],Y)	RLC ([$10],Y)	4	3	91	69	XX			✗

See also: RLCW, RRC, SLL, SRL, SRA, ADC, SWAP, SLA

RLCW — Rotate Word Left Logical through Carry — RLCW

Syntax RLCW dst e.g. RLCW X

Operation

Description The destination is an index register. This instruction shifts all bits of the register one place to the left through Carry bit. Bit 0 of the result is a copy of CC.C value before the operation.

Instruction overview

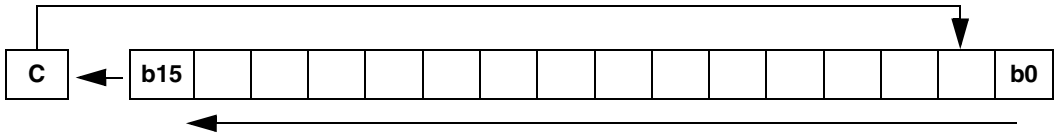
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
RLCW	Reg	-	-	-	-	N	Z	bit15

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b15

Set if, before the shift, the MSB of register or memory was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	RLCW X	2	1		59				✗
Y	RLCW Y	2	2	90	59				✗

See also: RLC, RRCW, SLLW, SRLW, SRAW, SWAPW, SLAW

RLWA — Rotate Word Left through A — RLWA

Syntax RLWA dst e.g. RLWA Y,A

Operation A <= dst_H <= dst_L <= A

Description The destination index register and Accumulator are rotated left by 1-byte.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
RLWA	X	A	-	-	-	-	N	Z	-
RLWA	Y	A	-	-	-	-	N	Z	-

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	A	RLWA X	1	1		02				
Y	A	RLWA Y	1	2	90	02				

See also: RRWA, SWAPW

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.This instruction shifts all bits of the register or memory, one place to the right. Bit 7 of the result is a copy of the CC.C bit value before the operation.

Instruction overview

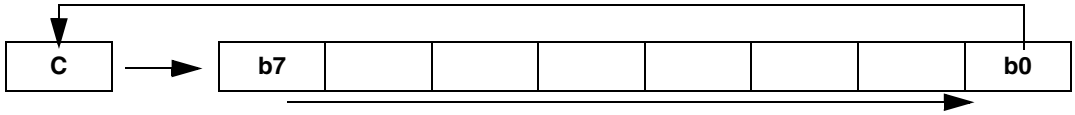
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
RRC	Reg	-	-	-	-	N	Z	bit0
RRC	Mem	-	-	-	-	N	Z	bit0

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b0

Set if, before the shift, the LSB of register or memory was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	RRC A	1	1		46				✗
shortmem	RRC $10	1	2		36	XX			✗
longmem	RRC $1000	1	4	72	56	MS	LS		
(X)	RRC (X)	1	1		76				✗
(shortoff,X)	RRC ($10,X)	1	2		66	XX			✗
(longoff,X)	RRC ($1000,X)	1	4	72	46	MS	LS		
(Y)	RRC (Y)	1	2	90	76				✗
(shortoff,Y)	RRC ($10,Y)	1	3	90	66	XX			✗
(longoff,Y)	RRC ($1000,Y)	1	4	90	46	MS	LS		
(shortoff,SP)	RRC ($10,SP)	1	2		06	XX			✗
[shortptr.w]	RRC [$10]	4	3	92	36	XX			✗
[longptr.w]	RRC [$1000].w	4	4	72	36	MS	LS		
([shortptr.w],X)	RRC ([$10],X)	4	3	92	66	XX			✗
([longptr.w],X)	RRC ([$1000.w],X)	4	4	72	66	MS	LS		
([shortptr.w],Y)	RRC ([$10],Y)	4	3	91	66	XX			✗

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

RRCW Rotate Word Right Logical through Carry RRCW

Syntax RRCW dst e.g. RRCWX

Operation

Description The destination is an index register. This instruction shifts all bits of the register or memory, one place to the right, through the Carry bit. Bit 15 of the result is a copy of the CC.C bit value before the operation.

Instruction overview

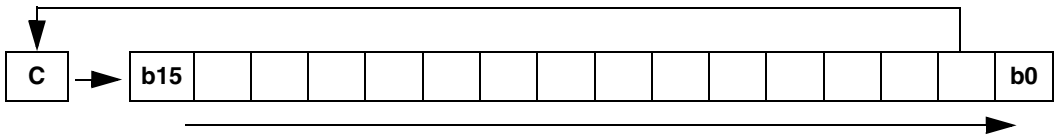
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
RRCW	Reg	-	-	-	-	N	Z	bit0

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b0

Set if, before the shift, the MSB of register or memory was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	RRCW X	2	1		56				✗
Y	RRCW Y	2	2	90	56				✗

See also: RRC, RLCW, SRLW, SLLW, SRAW, SWAPW, SLAW

RRWA Rotate Right Word through A RRWA

Syntax RRWA dst e.g. RRWA Y,A

Operation A => dst_H => dst_L => A

Description The destination index register and Accumulator are rotated right by 1-byte.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
RLWA	X	A	-	-	-	-	N	Z	-
RLWA	Y	A	-	-	-	-	N	Z	-

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	A	RRWA X	1	1		01				
Y	A	RRWA Y	1	2	90	01				

See also: RLWA, SWAPW

RVF Reset overflow flag RVF

Syntax RVF

Operation V = 0

Description Clear the overflow flag of the Condition Code (CC) register. May be used as a boolean user controlled flags.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
RCF	0	-	-	-	-	-	-

V: 0

Cleared.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	RVF	1	1		9C				✗

See also: RCF, SCF

SBC — Subtraction with Carry/Borrow — SBC

Syntax SBC A,src e.g. SBC A,#$15

Operation A <= A- src - C

Description The source byte, along with the carry flag, is subtracted from the contents of the accumulator and the result is stored in the accumulator. The source is a memory or data byte.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
SBC	A	Mem	V	-	-	-	N	Z	C

V ⇒ $(\overline{A7}.M7 + \overline{A7}.R7 + A7.M7.R7) \oplus (\overline{A6}.M6 + \overline{A6}.R6 + A6.M6.R6)$
Set if the signed subtraction generates an overflow, cleared otherwise.

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ $\overline{A7}.M7 + \overline{A7}.R7 + A7.M7.R7$
Set if a borrow request occurred from bit 7 of the result, cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	SBC A,#$55	1	2		A2	XX			✗
A	shortmem	SBC A,$10	1	2		B2	XX			✗
A	longmem	SBC A,$1000	1	3		C2	MS	LS		✗
A	(X)	SBC A,(X)	1	1		F2				✗
A	(shortoff,X)	SBC A,($10,X)	1	2		E2	XX			✗
A	(longoff,X)	SBC A,($1000,X)	1	3		D2	MS	LS		✗
A	(Y)	SBC A,(Y)	1	2	90	F2				✗
A	(shortoff,Y)	SBC A,($10,Y)	1	3	90	E2	XX			✗
A	(longoff,Y)	SBC A,($1000,Y)	1	4	90	D2	MS	LS		✗
A	(shortoff,SP)	SBC A,($10,SP)	1	2		12	XX			
A	[shortptr.w]	SBC A,[$10.w]	4	3	92	C2	XX			✗
A	[longptr.w]	SBC A,[$1000.w]	4	4	72	C2	MS	LS		
A	([shortptr.w],X)	SBC A,([$10.w],X)	4	3	92	D2	XX			✗
A	([longptr.w],X)	SBC A,([$1000.w],X)	4	4	72	D2	MS	LS		
A	([shortptr.w],Y)	SBC A,([$10.w],Y)	4	3	91	D2	XX			✗

See also: ADD,ADC,SUB, MUL

SCF Set Carry Flag SCF

Syntax SCF

Operation C = 1

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

Instruction overview

mnem
SCF

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
SCF	-	-	-	-	-	-	1

C: 1

Set.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	SCF	1	1		99				✗

See also: RCF, RVF

SIM — Set Interrupt Mask/Disable Interrupt — SIM

Syntax sim

Operation I1 = 1, I0 = 1

Description Set the Interrupt mask of the Condition Code (CC) register, which disables interrupts. This instruction is useless at the beginning of reset routine. It need not be used at the beginning of interrupt routines as the interrupt level is set automatically in CC.I[1:0].

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
SIM	-	1	-	1	-	-	-

I1 and I0: 1

Set.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	SIM	1	1		9B				✗

See also: RIM

SLL/SLA — Shift Left Logical/Shift Left Arithmetic — SLL/SLA

Syntax SLL dst e.g. SLL (X)

SLA dst e.g. SLA (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

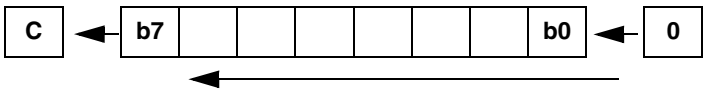
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
SLL/SLA	Mem	-	-	-	-	N	Z	bit7
SLL/SLA	Reg	-	-	-	-	N	Z	bit7

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b7

Set if, before the shift, bit 7 of register or memory was set, cleared otherwise.



Detailed description

dst	Asm(1)	cy	lgth	Op-code(s)					ST7
A	SLL A	1	1		48				✗
shortmem	SLL $15	1	2		38	XX			✗
longmem	SLL $1505	1	4	72	58	MS	LS		
(X)	SLL (X)	1	1		78				✗
(shortoff,X)	SLL ($15,X)	1	2		68	XX			✗
(longoff,X)	SLL ($1505,X)	1	4	72	48	MS	LS		
(Y)	SLL (Y)	1	2	90	78				✗
(shortoff,Y)	SLL ($15,Y)	1	3	90	68	XX			✗
(longoff,Y)	SLL ($1505,Y)	1	4	90	48	MS	LS		
(shortoff,SP)	SLL ($15,SP)	1	2		08	XX			✗
[shortptr.w]	SLL [$15]	4	3	92	38	XX			✗
[longptr.w]	SLL [$1505].w	4	4	72	38	MS	LS		
([shortptr.w],X)	SLL ([$15],X)	4	3	92	68	XX			✗

([longptr.w],X)	SLL ([$1505.w],X)	4	4	72	68	MS	LS		
([shortptr.w],Y)	SLL ([$15],Y)	4	3	91	68	XX			✗

1. For the shift left arithmetic instruction, replace SLL by SLA.

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

SLLW/SLAW — Shift Left Logical Word/Shift Left Arithmetic Word — SLLW/SLAW

Syntax SLLW dst e.g. SLLW X

SLAW dst e.g. SLAW X

Operation

Description The destination is an index register.It double the affected value.

Instruction overview

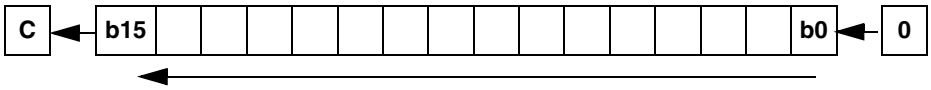
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
SLLW/SLAW	Reg	-	-	-	-	N	Z	bit15

N ⇒ R15
Set if bit 15of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b15

Set if, before the shift, bit 15 of register or memory was set, cleared otherwise.



Detailed description

dst	Asm(1)	cy	lgth	Op-code(s)					ST7
X	SLLW X	2	1		58				
Y	SLLW Y	2	2	90	58				

1. For the shift left arithmetic word instruction, replace SLLW by SLAW.

See also: SLL, SRAW, SRLW, RRCW, RLCW, SWAPW, SLAW

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 performs 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

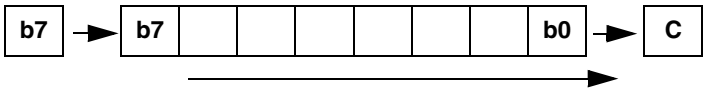
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
SRA	Reg	-	-	-	-	N	Z	bit0
SRA	Mem	-	-	-	-	N	Z	bit0

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b0

Set if, before the shift, the LSB of register or memory was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	SRA A	1	1		47				✗
shortmem	SRA $15	1	2		37	XX			✗
longmem	SRA $1505	1	4	72	57	MS	LS		
(X)	SRA (X)	1	1		77				✗
(shortoff,X)	SRA ($15,X)	1	2		67	XX			✗
(longoff,X)	SRA ($1505,X)	1	4	72	47	MS	LS		
(Y)	SRA (Y)	1	2	90	77				✗
(shortoff,Y)	SRA ($15,Y)	1	3	90	67	XX			✗
(longoff,Y)	SRA ($1505,Y)	1	4	90	47	MS	LS		
(shortoff,SP)	SRA ($15,SP)	1	2		07	XX			✗
[shortptr.w]	SRA [$15]	4	3	92	37	XX			✗
[longptr.w]	SRA [$1505].w	4	4	72	37	MS	LS		
([shortptr.w],X)	SRA ([$15],X)	4	3	92	67	XX			✗
([longptr.w],X)	SRA ([$1505.w],X)	4	4	72	67	MS	LS		
([shortptr.w],Y)	SRA ([$15],Y)	4	3	91	67	XX			✗

See also: SRAW, SRL, SLL, RRC, RLC, SWAP

SRAW — Shift Right Arithmetic Word — SRAW

Syntax SRAW dst e.g. SRAW X

Operation

Description The destination is an index register. It performs a signed division by 2. The sign bit (15) is not modified.

Instruction overview

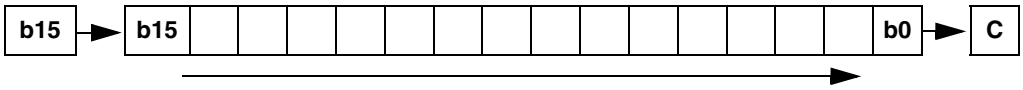
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
SRAW	Reg	-	-	-	-	N	Z	bit0

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

C ⇒ b0

Set if, before the shift, the LSB of register or memory was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	SRAW X	2	1		57				
Y	SRAW Y	2	2	90	57				

See also: SRA, SRLW, SLLW, RRCW, RLCW, SWAPW

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

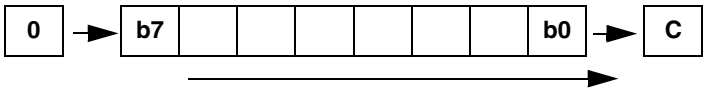
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
SRL	Reg	-	-	-	-	N	Z	bit0
SRL	Mem	-	-	-	-	N	Z	bit0

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b0

Set if, before the shift, the LSB of register or memory was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	SRL A	1	1		44				✗
shortmem	SRL $15	1	2		34	XX			✗
longmem	SRL $1505	1	4	72	54	MS	LS		
(X)	SRL (X)	1	1		74				✗
(shortoff,X)	SRLL ($15,X)	1	2		64	XX			✗
(longoff,X)	SRL ($1505,X)	1	4	72	44	MS	LS		
(Y)	SRL (Y)	1	2	90	74				✗
(shortoff,Y)	SRL ($15,Y)	1	3	90	64	XX			✗
(longoff,Y)	SRL ($1505,Y)	1	4	90	44	MS	LS		
(shortoff,SP)	SRL ($15,SP)	1	2		04	XX			✗
[shortptr.w]	SRL [$15]	4	3	92	34	XX			✗
[longptr.w]	SRL [$1505].w	4	4	72	34	MS	LS		
([shortptr.w],X)	SRL ([$15],X)	4	3	92	64	XX			✗
([longptr.w],X)	SRL ([$1505.w],X)	4	4	72	64	MS	LS		
([shortptr.w],Y)	SRL ([$15],Y)	4	3	91	64	XX			✗

See also: RLC, RRC, SRA, SWAP, SLL, SRLW

SRLW Shift Right Logical Word SRLW

Syntax SRLW dst e.g. SRLW X

Operation

Description The destination is an index register. It performs an unsigned division by 2.

Instruction overview

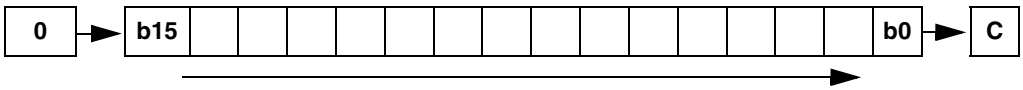
mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
SRLW	Reg	-	-	-	-	N	Z	bit0

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ b0

Set if, before the shift, the LSB of the register was set, cleared otherwise.



Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	SRLW X	2	1		54				
Y	SRLW Y	2	2	90	54				

See also: SRL, RLCW, RRCW, SRLW, SRAW, SWAPW, SLLW

SUB Subtraction SUB

Syntax SUB A,src e.g. SUB A,#%11001010

Operation A <= A- src

Description The source byte is subtracted from the contents of the accumulator/SP and the result is stored in the accumulator/SP. The source is a memory or data byte.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
SUB	A	Mem	V	-	-	-	N	Z	C
SUB	SP	Imm	-	-	-	-	-	-	-

V ⇒ (A7.M7 + A7.R7 + A7.M7.R7) ⊕ (A6.M6 + A6.R6 + A6.M6.R6)
Set if the signed operation generates an overflow, cleared otherwise.

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ R7.R6.R5.R4.R3.R2.R1.R0
Set if the result is zero (0x00), cleared otherwise.

C ⇒ $\overline{A7}$.M7 + $\overline{A7}$.R7 + A7.M7.R7
Set if a borrow request occurred from bit 7, cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	SUB A,#$55	1	2		A0	XX			✗
A	shortmem	SUB A,$10	1	2		B0	XX			✗
A	longmem	SUB A,$1000	1	3		C0	MS	LS		✗
A	(X)	SUB A,(X)	1	1		F0				✗
A	(shortoff,X)	SUB A,($10,X)	1	2		E0	XX			✗
A	(longoff,X)	SUB A,($1000,X)	1	3		D0	MS	LS		✗
A	(Y)	SUB A,(Y)	1	2	90	F0				✗
A	(shortoff,Y)	SUB A,($10,Y)	1	3	90	E0	XX			✗
A	(longoff,Y)	SUB A,($1000,Y)	1	4	90	D0	MS	LS		✗
A	(shortoff,SP)	SUB A,($10,SP)	1	2		10	XX			
A	[shortptr.w]	SUB A,[$10.w]	4	3	92	C0	XX			✗
A	[longptr.w]	SUB A,[$1000.w]	4	4	72	C0	MS	LS		
A	([shortptr.w],X)	SUB A,([$10.w],X)	4	3	92	D0	XX			✗
A	([longptr.w],X)	SUB A,([$1000.w],X)	4	4	72	D0	MS	LS		
A	([shortptr.w],Y)	SUB A,([$10.w],Y)	4	3	91	D0	XX			✗
SP	#byte	SUB SP,#$9	1	2		52	XX			

See also: SUBW, ADD, ADC, SBC, MUL

SUBW Word Subtraction SUBW

Syntax SUBW dst,src e.g. SUBW X, #$5500

Operation dst <= dst - src

Description The source 16-bit word is subtracted from the contents of the destination index register and the result is stored in the same index register. The source is a memory or 16-bit data.

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
SUBW	X	Mem	V	-	H	-	N	Z	C
SUBW	Y	Mem	V	-	H	-	N	Z	C

V ⇒ $(\overline{X15}.M15 + \overline{X15}.R15 + X15.M15.R15) \oplus (\overline{X14}.M14 + \overline{X14}.R14 + X14.M14.R14)$
Set if the signed operation generates an overflow, cleared otherwise.

H ⇒ $\overline{X7}.M7 + \overline{X7}.R7 + X7.M7.R7$
Set if a carry occurred from bit 7, cleared otherwise.

N ⇒ R15
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

C ⇒ $\overline{X15}.M15 + \overline{X15}.R15 + X15.M15.R15$
Set if a borrow request occurred from bit 15, cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
X	#word	SUBW X,#$5500	2	3		1D	MS	LS		
X	longmem	SUBW X,$1000	2	4	72	B0	MS	LS		
X	(shortoff, SP)	SUBW X,($10,SP)	2	3	72	F0	XX			
Y	#word	SUBW Y,#$5500	2	4	72	A2	MS	LS		
Y	longmem	SUBW Y,$1000	2	4	72	B2	MS	LS		
Y	(shortoff, SP)	SUBW Y,($10,SP)	2	3	72	F2	XX			

See also: SUB, ADDW, 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	Affected condition flags						
		V	I1	H	I0	N	Z	C
SWAP	Reg	-	-	-	-	N	Z	-
SWAP	Mem	-	-	-	-	N	Z	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	SWAP A	1	1		4E				✗
shortmem	SWAP $15	1	2		3E	XX			✗
longmem	SWAP $1505	1	4	72	5E	MS	LS		
(X)	SWAP (X)	1	1		7E				✗
(shortoff,X)	SWAPL ($15,X)	1	2		6E	XX			✗
(longoff,X)	SWAP ($1505,X)	1	4	72	4E	MS	LS		
(Y)	SWAP (Y)	1	2	90	7E				✗
(shortoff,Y)	SWAP ($15,Y)	1	3	90	6E	XX			✗
(longoff,Y)	SWAP ($1505,Y)	1	4	90	4E	MS	LS		
(shortoff,SP)	SWAP ($15,SP)	1	2		0E	XX			✗
[shortptr.w]	SWAP [$15]	4	3	92	3E	XX			✗
[longptr.w]	SWAP [$1505].w	4	4	72	3E	MS	LS		
([shortptr.w],X)	SWAP ([$15],X)	4	3	92	6E	XX			✗
([longptr.w],X)	SWAP ([$1505.w],X)	4	4	72	6E	MS	LS		
([shortptr.w],Y)	SWAP ([$15],Y)	4	3	91	6E	XX			✗

See also: SWAPW, RRC, RLC, SLL, SRL, SRA

SWAPW Swap bytes SWAPW

Syntax SWAPW dst e.g. SWAPW Y

Operation

Description The destination index register upper and low bytes are swapped over.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
SWAP	Reg	-	-	-	-	N	Z	-

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	SWAPW X	1	1		5E				✗
Y	SWAPW Y	1	2	90	5E				✗

See also: SWAP, 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 of the Condition Code (CC) register are updated accordingly. This instruction is compact, and does not affect any register when used with RAM variables.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
TNZ	Reg	-	-	-	-	N	Z	-
TNZ	Mem	-	-	-	-	N	Z	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
A	TNZ A	1	1		4D				✗
shortmem	TNZ $15	1	2		3D	XX			✗
longmem	TNZ $1505	1	4	72	5D	MS	LS		
(X)	TNZ (X)	1	1		7D				✗
(shortoff,X)	TNZL ($15,X)	1	2		6D	XX			✗
(longoff,X)	TNZ ($1505,X)	1	4	72	4D	MS	LS		
(Y)	TNZ (Y)	1	2	90	7D				✗
(shortoff,Y)	TNZ ($15,Y)	1	3	90	6D	XX			✗
(longoff,Y)	TNZ ($1505,Y)	1	4	90	4D	MS	LS		
(shortoff,SP)	TNZ ($15,SP)	1	2		0D	XX			✗
[shortptr.w]	TNZ [$15]	4	3	92	3D	XX			✗
[longptr.w]	TNZ [$1505].w	4	4	72	3D	MS	LS		
([shortptr.w],X)	TNZ ([$15],X)	4	3	92	6D	XX			✗
([longptr.w],X)	TNZ ([$1505.w],X)	4	4	72	6D	MS	LS		
([shortptr.w],Y)	TNZ ([$15],Y)	4	3	91	6D	XX			✗

See also: TNZW, CP, BCP

TNZW — Word Test for Negative or Zero — TNZW

Syntax TNZW dst e.g. TNZW X

Operation {N, Z} = Test(dst)

Description The destination 16-bit word, index register, is tested and both N and Z flags of the Condition Code (CC) register are updated accordingly.

Instruction overview

mnem	dst	Affected condition flags						
		V	I1	H	I0	N	Z	C
TNZW	Reg	-	-	-	-	N	Z	-

N ⇒ R15
Set if bit 15 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R15}.\overline{R14}.\overline{R13}.\overline{R12}.\overline{R11}.\overline{R10}.\overline{R9}.\overline{R8}.\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x0000), cleared otherwise.

Detailed description

dst	Asm	cy	lgth	Op-code(s)					ST7
X	TNZW X	2	1		5D				
Y	TNZW Y	2	2	90	5D				

See also: TNZ, CPW

TRAP Software interrupt TRAP

Syntax TRAP

Operation
PC = PC + 1
(SP--) = LSB (PC)
(SP--) = MSB (PC)
(SP--) = Ext(PC)
(SP--) = YL
(SP--) = YH
(SP--) = XL
(SP--) = XH
(SP--) = A
(SP--) = CC
PC = TRAP Interrupt Vector Contents

Description When processed, this instruction forces the trap interrupt to occur and to be processed. It cannot be masked by the I0 or I1 flags.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
TRAP	-	1	-	1	-	-	-

I1 and I0: 1

Set.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	TRAP	9	1		83				✗

See also: IRET

WFE — Wait for Event (CPU stopped, low power mode) — WFE

Syntax WFE

Operation The CPU Clock is stopped till an external event occurs. Internal peripherals are still running. It is used for synchronization with other computing resources (e.g coprocessor).

Description The state of the CPU is frozen, waiting for synchronization with an external event. The CPU clock also is stopped, reducing the power consumption of the microcontroller. Interrupt requests during this period are served normally, depending on the CC.I[1:0] value.

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
WFE	-	-	-	-	-	-	-

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	WFE	1	2	72	8F				

See also: HALT

WFI — Wait for Interrupt (CPU stopped, low power mode) — WFI

Syntax WFI

Operation CC.I1= 1, CC.I0 = 0. The CPU Clock is stopped till an interrupt occurs. Internal peripherals are still running.

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

Instruction overview

mnem	Affected condition flags						
	V	I1	H	I0	N	Z	C
WFI	-	1	-	0	-	-	-

I1: 1

Set.

I0: 0

Cleared.

Detailed description

Addressing mode	Asm	cy	lgth	Op-code(s)					ST7
Inherent	WFI	10	1		8F				✗

See also: HALT

XOR — Logical Exclusive OR — XOR

Syntax XOR A,src e.g. XOR A,#%00110101

Operation A <= A XOR src

Description The source byte, is logically XORed with the contents of the accumulator and the result is stored in the accumulator. The source is a memory or data byte.

Truth table

XOR	0	1
0	0	1
1	1	0

Instruction overview

mnem	dst	src	Affected condition flags						
			V	I1	H	I0	N	Z	C
XOR	A	Mem	-	-	-	-	N	Z	-

N ⇒ R7
Set if bit 7 of the result is set (negative value), cleared otherwise.

Z ⇒ $\overline{R7}.\overline{R6}.\overline{R5}.\overline{R4}.\overline{R3}.\overline{R2}.\overline{R1}.\overline{R0}$
Set if the result is zero (0x00), cleared otherwise.

Detailed description

dst	src	Asm	cy	lgth	Op-code(s)					ST7
A	#byte	XOR A,#$55	1	2		A8	XX			✗
A	shortmem	XOR A,$10	1	2		B8	XX			✗
A	longmem	XOR A,$1000	1	3		C8	MS	LS		✗
A	(X)	XOR A,(X)	1	1		F8				✗
A	(shortoff,X)	XOR A,($10,X)	1	2		E8	XX			✗
A	(longoff,X)	XOR A,($1000,X)	1	3		D8	MS	LS		✗
A	(Y)	XOR A,(Y)	1	2	90	F8				✗
A	(shortoff,Y)	XOR A,($10,Y)	1	3	90	E8	XX			✗
A	(longoff,Y)	XOR A,($1000,Y)	1	4	90	D8	MS	LS		✗
A	(shortoff,SP)	XOR A,($10,SP)	1	2		18	XX			
A	[shortptr.w]	XOR A,[$10.w]	4	3	92	C8	XX			✗
A	[longptr.w]	XOR A,[$1000.w]	4	4	72	C8	MS	LS		
A	([shortptr.w],X)	XOR A,([$10.w],X)	4	3	92	D8	XX			✗
A	([longptr.w],X)	XOR A,([$1000.w],X)	4	4	72	D8	MS	LS		
A	([shortptr.w],Y)	XOR A,([$1000],Y)	4	3	91	D8	XX			✗

See also: AND, OR, CPL, NEG

8 Revision history

Table 43. Document revision history

Date	Revision	Changes
14-Jan-2008	1	Initial release.
05-Jun-2008	2	Modified *Figure 2: Context save/restore for interrupts on page 14*
20-Sep-2011	3	Changed notation for hexadecimal numbers from XXh to 0xXX. Removed CPU register context saving from *Section 3.2: CPU registers*. Added LDF in *Table 22: Available Extended Direct addressing mode instructions*. Updated *Figure 2: Context save/restore for interrupts*. Added BREAK instruction in *Table 41: Instruction groups*. Added *Section 5: Pipelined execution*. *Table 42: Instruction set summary*: updated ADDW, BCCM, BRES, BSET, BTJF, BTJT, CALLR, DEC, DECW, DIV, EXGW, JRA, JRC, JREQ, JRH, JRIH, JRIL, JRM, JRMI, JRNC, JRNE, JRNH, JRNM, JRNV, JRPL, JRSGE, JRSGT, JRSLE, JRSLT, JRUGE, JRULT, LDF, LDW, MOV, NEG, PUSH, RRCW, SBC, SLA, SLAW, SLL, SLLW, SRA, SRAW, SRL, SRLW, SUB. Added BREAK and INT instructions. *Section 7: STM8 instruction set*: updated ADD, ADDW, BCCM, BCP, BCPL, BRES, BSET, BTJF, BTJT, CLR, CP, CPW, DEC, DECW, HALT, INCW, INT, JP, JRxx, MOV, RLWA, RRCW, RRWA, SBC, SRLW, SUB, and SWAPW. Added BREAK instruction. Merged JRA with JRL instructions, SLA with SLL, and SLAW with SLLW.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2011 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com