SGS-THOMSON MICROELECTRONICS

ST62 - ST63

PROGRAMMING MANUAL

INTRODUCTION

This manual deals with the description of the instruction set and addressing modes of ST62,63 microcontroller series. The manual is divided in two main sections. The first one includes, after a general family description, the addressing modes description. The second section includes the detailed description of ST62,63 instruction set. Each instruction is described in detail with the differences between each ST6 series.
ST6 software has been designed to fully use the hardware in the most efficient way possible while keeping byte usage to a minimum; in short to provide byte efficient programming capability.

PROGRAMMING MODEL

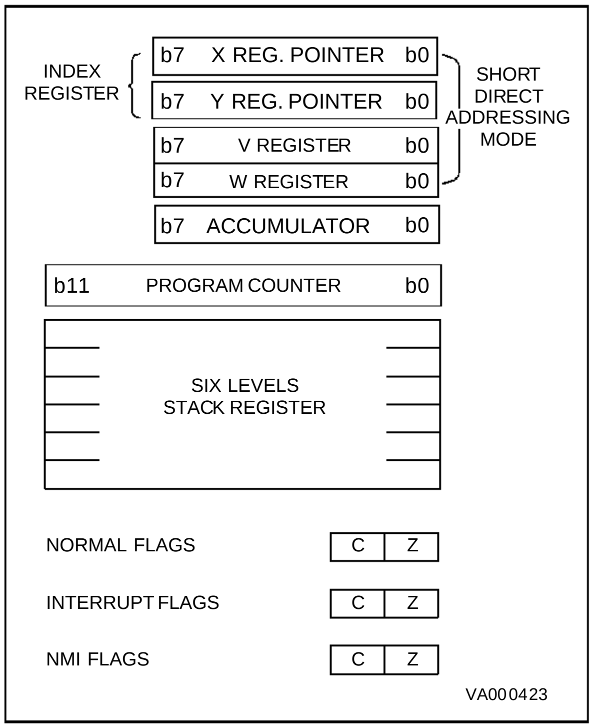
It is useful at this stage to outline the programming model of the ST62,63 series, by which we mean the available memory spaces, their relation to one another, the interrupt philosophy and so on.

Memory Spaces. The ST6 devices have three different memory spaces: data, program and stack. All addressing modes are memory space specific so there is no need for the user to specify which space is being used as in more complex systems. The stack space, which is used automatically with subroutine and interrupt management for program counter storage, is not accessible to the user.

Table 1. ST62,63 Series Core Characteristics

	ST62,63 Series
Stack Levels	6
Interrupt Vectors	5
NMI	YES
Flags Sets	3
Program ROM	2K + 2K• n 20K Max
Data RAM	64 byte • m
Data ROM	64 byte pages in ROM
Carry Flag SUB Instruction	Reset if A > Source
Carry Flag CP Instruction	Set if A < Source

Figure 1. ST6 Family Programming Model



PROGRAMMING MODEL (Continued)

Figure 2. ST62 Data Space Example

b7 … b0	
NOT IMPLEMENTED	000h 03Fh
DATA ROM/EPROM WINDOW 64 BYTE	040h 07Fh
X REGISTER	080h
Y REGISTER	081h
V REGISTER	082h
W REGISTER	083h
DATA RAM 60 BYTES	084h 0BFh
PORT A DATA REGISTER	0C0h
PORT B DATA REGISTER	0C1h
PORT C DATA REGISTER	0C2h
RESERVED	0C3h
PORT A DIRECTION REGISTER	0C4h
PORT B DIRECTION REGISTER	0C5h
PORT C DIRECTION REGISTER	0C6h
RESERVED	0C7h
INTERRUPT OPTION REGISTER	0C8h
DATA ROM WINDOW REGISTER	0C9h
RESERVED	0CAh 0CBh
PORT A OPTION REGISTER	0CCh
PORT B OPTION REGISTER	0CDh
PORT C OPTION REGISTER	0CEh
RESERVED	0CFh
A/D DATA REGISTER	0D0h
A/D CONTROL REGISTER	0D1h
TIMER PSC REGISTER	0D2h
TIMER DATA REGISTER	0D3h
TIMER TSCR REGISTER	0D4h
RESERVED	0D5h 0D7h
WATCHDOG REGISTER	0D8h
RESERVED	0D9h 0FEh
ACCUMULATOR	0FFh

Figure 3. ST62 Program Memory Example

b7 … b0	
NOT IMPLEMENTED	0000h 07FFh
RESERVED	0800h 087Fh
USER PROGRAM ROM 1828 BYTES	0880h 0F9Fh
RESERVED	0FA0h 0FEFh
INTERRUPT VECTOR #4 A/D INTERRUPT	0FF0h 0FF1h
INTERRUPT VECTOR #3 TIMER INTERRUPT	0FF2h 0FF3h
INTERRUPT VECTOR #2 PORT B & C INTERRUPT	0FF4h 0FF5h
INTERRUPT VECTOR #1 PORT A INTERRUPT	0FF6h 0FF7h
RESERVED	0FF8h 0FFBh
INTERRUPT VECTOR #0 NMI INTERRUPT	0FFCh 0FFDh
USER RESET VECTOR	0FFEh 0FFFh

On EPROM versions there are no reserved areas. These reserved bytes are present on ROM/OTP versions.

Data Memory Space. The following registers in the data space have fixed addresses which are hardware selected so as to decrease access times and reduce addressing requirements and hence program length. The Accumulator is an 8 bit register in location 0FFh. The X, Y, V & W registers have the addresses 80h-83h respectively. These are used for short direct addressing, reducing byte requirements in the program while the first two, X & Y, can also be used as index registers in the indirect addressing mode. These registers are part of the data RAM space. In the ST62 and ST63 for data space ROM a 6 bit (64 bytes addressing) window multiplexing in program ROM is available through a dedicated data ROM banking register.

PROGRAMMING MODEL (Continued)

For data RAM and I/O expansion the lowest 64 bytes of data space (00h-03Fh) are paged through a data RAM banking register.

Self-check Interrupt Vector FF8h & FF9h:
jp (self-check interrupt routine)

A jump instruction to the reset and interrupt routines must be written into these locations.

ST62 & ST63 Program Memory Space. The ST62 and ST63 devices can directly address up to 4K bytes (program counter is 12-bit wide). A greater ROM size is obtained by paging the lower 2K of the program ROM through a dedicated banking register located in the data space. The higher 2K of the program ROM can be seen as static and contains the reset, NMI and interrupt vectors at the following fixed locations:

Reset Vector FFEh & FFFh:
jp (reset routine)

NMI Interrupt Vector FFCh & FFDh:
jp (NMI routine)

Non user Vector FFAh & FFBh

Non user Vector FF8h & FF9h

Interrupt #1 Vector FF6h & FF7h jp (Int 1 routine)

Interrupt #2 Vector FF4h & FF5h jp (Int 2 routine)

Interrupt #3 Vector FF2h & FF3h jp (Int 3 routine)

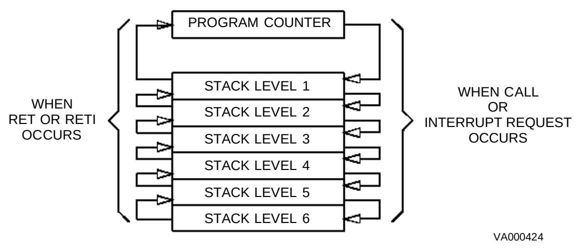
Interrupt #4 Vector FF0h & FF1h jp (Int 4 routine)

Program Counter & Stack Area. The program counter is a twelve bit counter register since it has to cover a direct addressing of 4K byte program memory space. When an interrupt or a subroutine occurs the current PC value is forward "pushed" into a deep LIFO stacking area. On the return from the routine the top (last in) PC value is "popped" out and becomes the current PC value. The ST60/61 series offer a 4-word deep stack for program counter storage during interrupt and sub-routines calls. In the ST62 and ST63 series the stack is 6-word deep.

Status Flags. Three pairs of status flags, each pair consisting of a Zero flag and a Carry flag, are available. In the ST62 and ST63 an additional third set is available. One pair monitors the normal status while the second monitors the state during interrupts; the third flags set monitors the status during Non Maskable interrupt servicing. The switching from one set to another one is automatic as the interrupt requests (or NMI request for ST62,ST63 only) are acknowledged and when the program returns after an interrupt service routine. After reset, NMI set is active, until the first RETI instruction is executed.

ST62 & ST63 Interrupt Description. The ST62 and ST63 devices have 5 user interrupt vectors (plus one vector for testing purposes). Interrupt vector #0 is connected to the not maskable interrupt input of the core. Interrupts from #1 to #4 can be connected to different on-chip and external sources (see individual datasheets for detailed information). All interrupts can be globally disabled through the interrupt option register. After the reset ST62 and ST63 devices are in NMI mode, so no other interrupts can be accepted and the NMI flags set is in use, until the RETI instruction is performed. If an interrupt is detected, a special cycle will be executed, during this cycle the program counter is loaded with the related interrupt vector address. NMI can interrupt other interrupt routines at any time while normal interrupt can't interrupt each other. If more then one interrupt is waiting service, they will be accepted according to their priority. Interrupt #1 has the highest priority while interrupt #4 the lowest. This priority relationship is fixed.

Figure 4. ST62/ST63 Stack Area

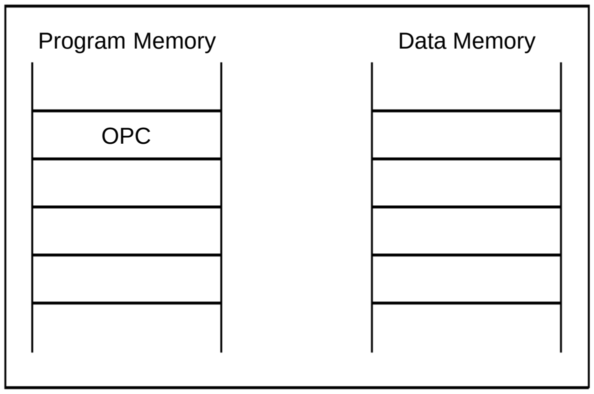


ADDRESSING MODES

The ST6 family gives the user nine addressing modes for access to data locations. Some of these are specifically tailored to particular instruction types or groups while others are designed to reduce program length and operating time by using the hardware facilities such as the X, Y, V & W registers. The data locations can be in either the program memory space or the data memory space when the ST6 is operating due to user software. In addition the ST6 has a stack space for the 12 bit program counter but this is controlled by internal programming and is not accessible by the user. This section will describe all the addressing modes which are provided to the user. The following is the complete list of the ST6 available addressing modes:

- Inherent
- Direct
- Short Direct
- Indirect
- Immediate
- Program Counter Relative
- Extended
- Bit Direct
- Bit Test & Branch

Inherent. For instructions using the inherent addressing mode the opcode contains all the information necessary for execution. All instructions using this mode are **One Byte** instructions.

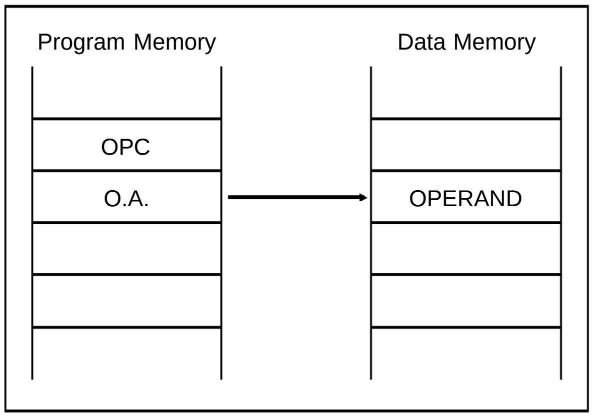


OPC = Opcode

Example:

Instruction	Comments
WAIT	Puts ST6 into the low power WAIT mode
STOP	Puts the ST6 into the lowest power mode
RETI	Returns from interrupt. Pops the PC from the PC stack.Sets the normal set of flags

Direct. In the direct addressing mode the address of the data is given by the program memory byte immediately following the opcode. This data location is in the data memory space. All instructions using this mode are **Two Bytes** instructions, lasting **Four Cycles**.



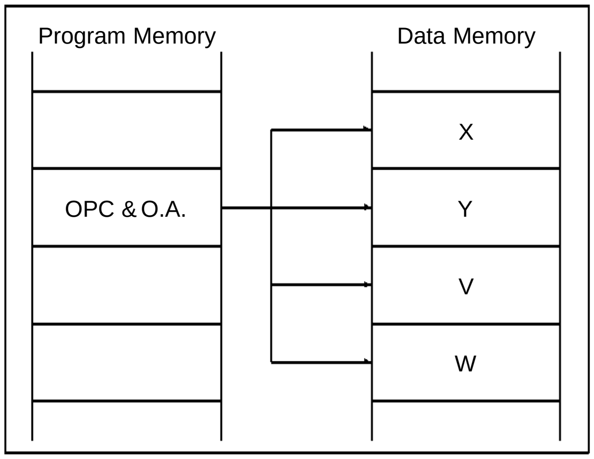
OPC = Opcode
O.A = Operand Address

Example:

Instruction	Comments
LD A,0A3h	Loads the accumulator with the value found in location A3h in the data space.
SUB A,11h	The value found in locations 11h in the data memory is subtracted from the value in the accumulator.

ADDRESSING MODES (Continued)

Short Direct. ST6 core has four fixed location registers in the data space which may be addressed in a short direct manner. The addresses and names of these registers are 80h (X), 81h (Y), 82h (V) and 83h (W). When using this addressing mode the data is in one of these registers and the address is a part of the opcode. All instructions using this mode are **One Byte** instructions, lasting **Four Cycles.**

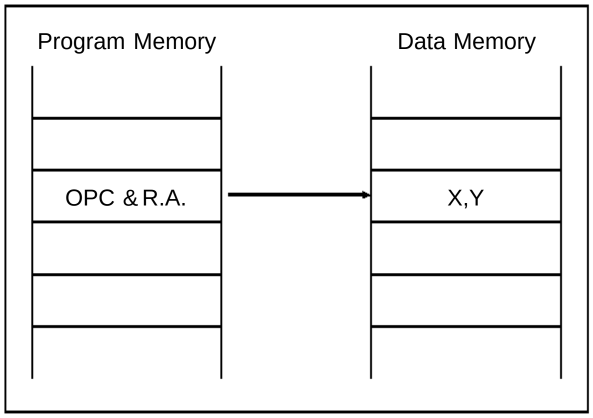


OPC = Opcode
O.A .= Operand Address

Example:

Instruction	Comments
LD A,X	The value of the X register (80h) is loaded into the accumulator.
INC X	The X register is incremented.

Indirect. The indirect mode must use either the X (80h) or Y (81h) register. This register contains the address of the data. The operand is at the data space address pointed to by the content of X or Y registers. All instructions using this mode are **One Byte** instructions, lasting **Four Cycles**.

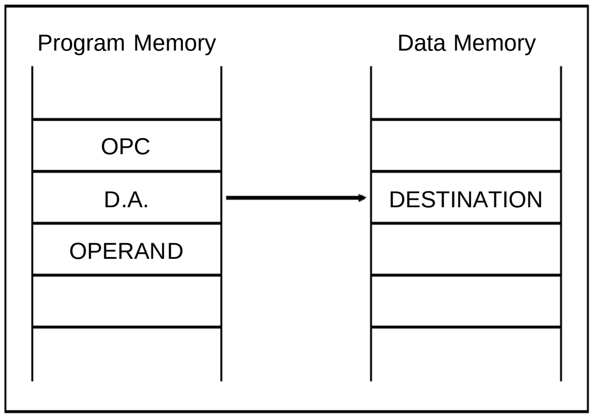


OPC = Opcode
R.A. = Register Address

Example:

Instruction	Comments
LD A,(X)	The value in the registers pointed to by the X register is loaded into the accumulator.
ADD A,(Y)	The value in the register pointed to by the Y register is added to the accumulator value.
INC (Y)	The value in the register pointed to by the Y register is incremented.

Immediate. In the immediate addressing mode the operand is found in the program ROM in a byte which is the last byte of the instruction. This addressing mode can be used for initializing data space registers and supplying constants. Instructions using this mode can be **Two** or **Three Bytes** instructions, lasting **Four Cycles**.



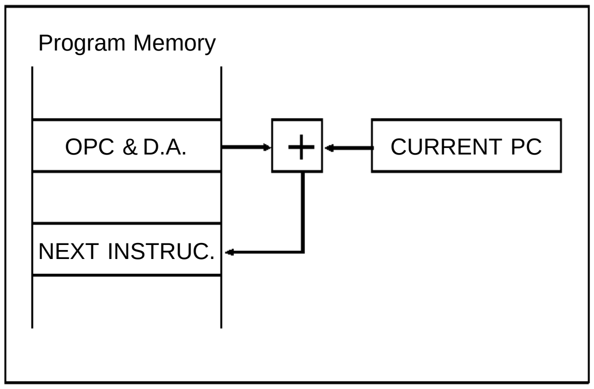
OPC = Opcode
D.A. = Destination Address

ADDRESSING MODES (Continued)

Example:

Instruction	Comments
LDI 34h,DFh	Loads immediate value DFh into data space location 34h.
SUBI A,22h	The immediate value 22h is substracted from the acc.

Program Counter Relative. This addressing mode is used only with conditional branches within the program. The opcode byte contains the data which is a fixed offset value. This offset is added to the program counter to give the address of the next instruction. The offset can have any value in the range -15 to +16. It is determined by the last five bits of the opcode. All instructions using this mode are **One Byte** Instructions, lasting **Two Cycles**.



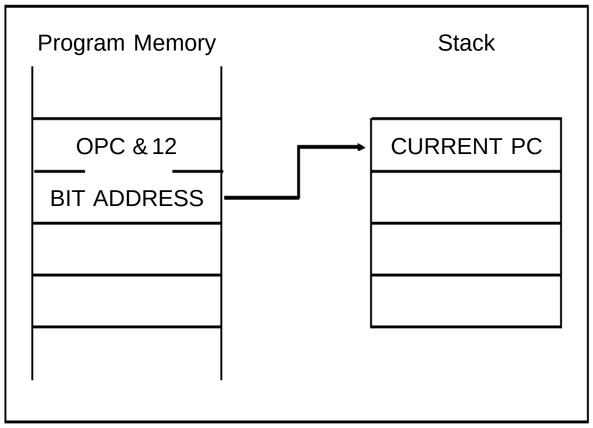
OPC = Opcode
D.A. = Destination Address

Example:

Instruction	Comments
JRC 3	If the carry flag is set then PC = PC+3
JRNZ -7	If the zero flag is not set (i.e the result of a previous instruction is not zero) then PC = PC-7

The relative jump address can be also a label that is automatically handled by the assembler.

Extended. The extended addressing mode is used to make long jumps within the program memory space (4K). The data requires 12 bits and is provided by half of the opcode byte and all of the second byte. All instructions using this mode are **Two Bytes** instructions, lasting **Four Cycles**.



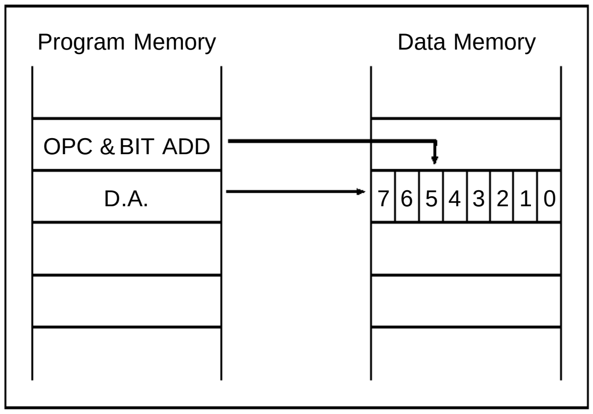
OPC = Opcode

Example:

Instruction	Comments
JP 3FAh	Loads 3FAh into program counter and continues with the instruction at 3FAh.
CALL ROU1	The current PC is pushed onto the stack and PC loaded with the value associated to the ROU1 label

The absolute jump address can be also a label that is automatically handled by the assembler.

Bit Direct. This addressing mode allows the user to set or clear any specified bit in a data memory register. The address of the bit is given in the form: "b,R" where b is the number of the bit and R is the address of the register. The bit is determined by three bits in the opcode and the register address is given by the second byte. All instructions using this mode are **Two Byte** instructions, lasting **Four Cycles**.



OPC = Opcode
D.A. = Destination Address

ADDRESSING MODES (Continued)

Example:

Instruction	Comments
SET 4,A	Sets bit 4 of the accumulator to 1.
RES 0,PORT	Clears bit 0 of PORT register

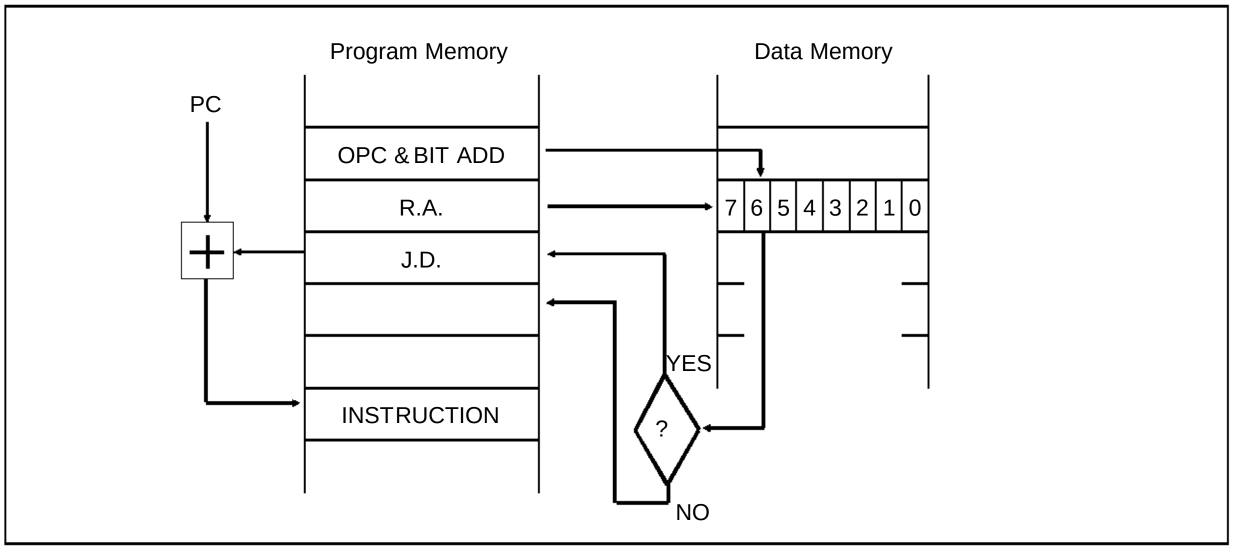
The register address can be associated to a label that is automatically handled by the assembler.

Bit Test & Branch. The bit test addressing mode is used in conditional jump instructions in which the jump depends on the result of a bit test. The opcode specifies the bit to be tested, the byte following the opcode in the register address in data space, and the third byte is the jump displacement, which is in the range -126 to +129. This displacement can be determined using a label, which is converted by the assembler. The state of the tested bit is also copied into the carry flag. All instructions using this mode are **Three Byte** instructions, lasting **Five Cycles**.

Example:

Instruction	Comments
JRS 3,PORT,LAB1	If bit three of data memory register associated to PORT label is set then PC=PC+LAB1 (where LAB1 is the jump displacement associated to a label
JRR 0,0Ah,-72	If bit 0 of data memory register OAh is reset to 0 then PC=PC-72.

The register address and the jump displacement can be associated to labels that are automatically handled by the assembler.



OPC = Opcode
R.A. = Relative Address
J.D. = Jump Displacement

ST62 & ST63 INSTRUCTION SET

The ST62,63 instructions can be divided functionally into the following seven groups.

- LOAD AND STORE
- ARITHMETIC AND LOGIC
- CONDITIONAL BRANCH
- JUMP AND CALL
- BIT MANIPULATION
- CONTROL
- IMPLIED

The following summary shows the instructions belonging to each group, the number of operands required for each instructions and the number of machine cycles. The flag behaviour is usually the same for both ST62 and ST63. The only difference is present for CP and SUB instructions as specified in the detailed description.

Note: For the following tables:

Δ: Affected

*: Not Affected

Table 2. Load & Store Instructions

Instruction	Bytes	Cycles	Flags	
			Z	C
LD	1	4	Δ	*
LD rr	2	4	Δ	*
LDI A	2	4	Δ	*
LDI	3	4	*	*

Table 3. Arithmetic & Logic Instructions

Instruction	Bytes	Cycles	Flags	
			Z	C
ADD	2	4	Δ	Δ
ADD (X,Y)	1	4	Δ	Δ
ADDI	2	4	Δ	Δ
AND	2	4	Δ	*
AND (X,Y)	1	4	Δ	*
ANDI	2	4	Δ	*
CLR A	2	4	Δ	Δ
CLR	3	4	*	*
COM	1	4	Δ	Δ
CP	2	4	Δ	Δ
CP (X,Y)	1	4	Δ	Δ
CPI	2	4	Δ	Δ
DEC	1	4	Δ	*
DEC A/rr	2	4	Δ	*
INC	1	4	Δ	*
INC A/rr	2	4	Δ	*
RLC	1	4	Δ	Δ
SLA	2	4	Δ	Δ
SUB	2	4	Δ	Δ
SUB (X,Y)	1	4	Δ	Δ
SUBI	2	4	Δ	Δ

Table 4. Conditional Branch Insructions

Instruction	Bytes	Cycles	Flags	
			Z	C
JRC	1	2	*	*
JRNC	1	2	*	*
JRR	3	5	*	Δ
JRS	3	5	*	Δ
JRZ	1	2	*	*
JRNZ	1	2	*	*

Table 5. Jump & Call Instructions

Instruction	Bytes	Cycles	Flags	
			Z	C
CALL	2	4	*	*
JP	2	4	*	*

ST62 & ST63 INSTRUCTION SET (Continued)

Table 6. Bit Manipulation Instructions

Instruction	Bytes	Cycles	Flags	
			Z	C
RES	2	4	*	*
SET	2	4	*	*

Table 7. Control Instructions

Instruction	Bytes	Cycles	Flags	
			Z	C
NOP	1	2	*	*
RET	1	2	*	*
RETI	1	2	Δ	Δ
STOP	1	2	*	*
WAIT	1	2	*	*

Table 8. Addressing Modes/Instruction Table

Instruction	Inh	Dir	Sh Dir	Ind	Imm	PCR	Ext	Bit Dir	Bit Test	Flags	
										Z	C
ADD		X	X	X						Δ	Δ
AND		X	X	X						Δ	*
CALL							X			*	*
CLR A		X								Δ	Δ
CLR		X								*	*
COM	X									Δ	Δ
CP		X		X	X					Δ	Δ
DEC		X	X	X						Δ	*
INC		X	X	X						Δ	*
JP							X			*	*
JRC, JRNC						X				*	*
JRZ, JRNZ						X				*	*
JRR, JRS									X	*	Δ
LD, LDI					X					Δ	*
NOP						X				*	*
RES, SET								X		*	*
RET	X									*	*
RETI	X									Δ	Δ
RLC	X									Δ	Δ
SLA	X									Δ	Δ
STOP, WAIT	X									*	*
SUB		X		X	X					Δ	Δ

Notes:

INH. Inherent, DIR: Direct, Sh.DIR: Short Direct,

IND. Indirect, IMM: Immediate, PCR: Program Counter Relative

EXT. Extended, BIT DIR: Bit Direct, BIT TEST.: Bit Test

Δ . Affected

* . Not Affected

ST62 & ST63 INSTRUCTION SET (Continued)

Table 9. Opcode Map

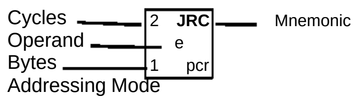
LOW / HI	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	LOW / HI
0 0000	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b0,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **LD** a,(x) 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b0,rr 2 b.d	2 **JRZ** e 1 pcr	4 **LDI** rr,nn 3 imm	2 **JRC** e 1 pcr	4 **LD** a,(y) 1 ind	**0 0000**
1 0001	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b0,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **INC** x 1 sd	2 **JRC** e 1 prc	4 **LDI** a,nn 2 imm	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b0,rr 2 b.d	2 **JRZ** e 1 pcr	4 **DEC** x 1 sd	2 **JRC** e 1 pcr	4 **LD** a,rr 2 dir	**1 0001**
2 0010	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b4,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **CP** a,(x) 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b4,rr 2 b.d	2 **JRZ** e 1 pcr	4 **COM** a 1 inh	2 **JRC** e 1 pcr	4 **CP** a,(y) 1 ind	**2 0010**
3 0011	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b4,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **LD** a,x 1 sd	2 **JRC** e 1 prc	4 **CPI** a,nn 2 imm	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b4,rr 2 b.d.	2 **JRZ** e 1 pcr	4 **LD** x,a 1 sd	2 **JRC** e 1 pcr	4 **CP** a,rr 2 dir	**3 0011**
4 0100	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b2,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **ADD** a,(x) 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b2,rr 2 b.d	2 **JRZ** e 1 pcr	2 **RETI** 1 inh	2 **JRC** e 1 pcr	4 **ADD** a,(y) 1 ind	**4 0100**
5 0101	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b2,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **INC** y 1 sd	2 **JRC** e 1 prc	4 **ADDI** a,nn 2 imm	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b2,rr 2 b.d	2 **JRZ** e 1 pcr	4 **DEC** y 1 sd	2 **JRC** e 1 pcr	4 **ADD** a,rr 2 dir	**5 0101**
6 0110	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b6,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **INC** (x) 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b6,rr 2 b.d	2 **JRZ** e 1 pcr	2 **STOP** 1 inh	2 **JRC** e 1 pcr	4 **INC** (y) 1 ind	**6 0110**
7 0111	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b6,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **LD** a,y 1 sd	2 **JRC** e 1 prc	#	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b6,rr 2 b.d	2 **JRZ** e 1 pcr	4 **LD** y,a 1 sd	2 **JRC** e 1 pcr	4 **INC** rr 2 dir	**7 0111**
8 1000	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b1,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **LD** (x),a 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b1,rr 2 b.d	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 pcr	4 **LD** (y),a 1 ind	**8 1000**
9 1001	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b1,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **INC** v 1 sd	2 **JRC** e 1 prc	#	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b1,rr 2 b.d	2 **JRZ** e 1 pcr	4 **DEC** v 1 sd	2 **JRC** e 1 pcr	4 **LD** rr,a 2 dir	**9 1001**
A 1010	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b5,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **AND** a,(x) 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b5,rr 2 b.d	2 **JRZ** e 1 pcr	4 **RLC** a 1 inh	2 **JRC** e 1 pcr	4 **AND** a,(y) 1 ind	**A 1010**
B 1011	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b5,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **LD** a,v 1 sd	2 **JRC** e 1 prc	4 **ANDI** a,nn 2 imm	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b5,rr 2 b.d	2 **JRZ** e 1 pcr	4 **LD** v,a 1 sd	2 **JRC** e 1 pcr	4 **AND** a,rr 2 dir	**B 1011**
C 1100	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b3,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **SUB** a,(x) 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b3,rr 2 b.d	2 **JRZ** e 1 pcr	2 **RET** 1 inh	2 **JRC** e 1 pcr	4 **SUB** a,(y) 1 ind	**C 1100**
D 1101	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b3,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **INC** w 1 sd	2 **JRC** e 1 prc	4 **SUBI** a,nn 2 imm	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b3,rr 2 b.d	2 **JRZ** e 1 pcr	4 **DEC** w 1 sd	2 **JRC** e 1 pcr	4 **SUB** a,rr 2 dir	**D 1101**
E 1110	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRR** b7,rr,ee 3 bt	2 **JRZ** e 1 pcr	#	2 **JRC** e 1 prc	4 **DEC** (x) 1 ind	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **RES** b7,rr 2 b.d	2 **JRZ** e 1 pcr	2 **WAIT** 1 inh	2 **JRC** e 1 pcr	4 **DEC** (y) 1 ind	**E 1110**
F 1111	2 **JRNZ** e 1 pcr	4 **CALL** abc 2 ext	2 **JRNC** e 1 pcr	5 **JRS** b7,rr,ee 3 bt	2 **JRZ** e 1 pcr	4 **LD** a,w 1 sd	2 **JRC** e 1 prc	#	2 **JRNZ** e 1 pcr	4 **JP** abc 2 ext	2 **JRNC** e 1 pcr	4 **SET** b7,rr 2 b.d	2 **JRZ** e 1 pcr	4 **LD** w,a 1 sd	2 **JRC** e 1 pcr	4 **DEC** rr 2 dir	**F 1111**

Abbreviations for Addressing Modes:

dir	Direct
sd	Short Direct
imm	Immediate
inh	Inherent
ext	Extended
b.d	Bit Direct
bt	Bit Test
pcr	Program Counter Relative
ind	Indirect

Legend:

#	Indicates Illegal Instructions
e	5 Bit Displacement
b	3 Bit Address
rr	1byte dataspace address
nn	1 byte immediate data
abc	12 bit address
ee	8 bit Displacement



ST62 & ST63 INSTRUCTION SET (Continued)

Table 10. Instruction Set Cycle-by-Cycle Summary

Instruction	Cycles	Cycles(#)	Address Bus	Data Bus	CPU Activity	Notes
Indirect Addressing Mode						
ADD, AND, CP, DEC, INC, LD, SUB	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 Opcode Address +1	Opcode (*) Next Instruction Next Instruction Next Instruction	Decode Opcode Read Operand Address Read Operand Execute Instruction	ROM Data Space not Addressed
ADD, AND, CP, DEC, INC, LD, SUB	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 Data Space Rom Add	Opcode (*) Next Instruction Next Instruction Rom Data (#)	Decode Opcode Read Operand Address Read Operand Execute Instruction	ROM Data Space Addressed
Direct Addressing Mode						
ADD, AND, CP, DEC, INC, LD, RES, SET, LSA, SUB, CLR	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 (*) Opcode Address +2	Opcode (*) Operand Address Operand Address(*) Next Instruction	Decode Opcode Address Data Space Read Operand Execute Instruction	ROM Data Space not Addressed
ADD, AND, CP, DEC, INC, LD, RES, SET, LSA, SUB, CLR	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 (*) Data Space Rom Add. (*)	Opcode (*) Operand Address Operand Address(#) Rom Data (#)	Decode Opcode Address Data Space Read Operand Execute Instruction	ROM Data Space Addressed
Immediate Addressing Mode						
ADDI, ANDI, CPI, LDI, SUBI	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1(*) Opcode Address +2(*)	Opcode (*) Immediate Operand Immediate Operand Next Instruction	Decode Opcode Idle Read Operand Execute Instruction	
LDI rr	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +2 Opcode AdDress +3	Opcode (*) Register Address Immediate Operand Next Opcode	Decode Opcode Read Register Address Read Immediate Operand Write Operand To Reg.	ROM Data Space not Addressed
LDI rr	4	1 2 3 4	Opcode Address(*) Opcode Address +1 (*) Opcode Address +2 (#) Data Space Rom Add.	Opcode (*) Register Address Immediate Operand Rom Operand (#)	Decode Opcode Read Register Address Read Immediate Operand Write Operand To Reg.	ROM Data Space Addressed
Short Direct Addressing Mode						
DEC, INC, LD	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 Opcode Address +1	Opcode (*) Next Opcode Next Opcode Next Opcode	Decode Opcode Define Data Space Add. Read Operand Execute Instruction	
Other Instructions						

Notes:*. Valid only at the beginning of the cycle

#. Valid only until t18 of the cycle

ST62 & ST63 INSTRUCTION SET (Continued)

Table 10. Instruction Set Cycle-by-Cycle Summary (Continued)

Instruction	Cycles	Cycles(#)	Address Bus	Data Bus	CPU Activity	Notes
CALL	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 Opcode Address +2(*)	Opcode (*) Subroutine Address Subroutine Address Next Instruction	Decode Opcode Increment Stack Pointer Push Return Address Calculate Subroutine Add.	
COM	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 Opcode Address +1	Opcode (*) Next Opcode Next Opcode Next Opcode	Decode Opcode Calculate Acc. Address Read Accumulator Complement Accumulator	
INTERRUPT	1	1	Next opcode address	Next Opcode (*)	Calculate Interrupt Add. Push Return Address Switch Flag Set	Note 1
JP	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 Opcode Address +2	Opcode (*) Jump Address Following Instr. Following Instr. (*)	Decode Opcode Idle Read Jump Address Calculate Jump Address	
JRC, JRNC, JRZ, JRNZ	2	1 2	Opcode Address(*) Opcode Address +1	Opcode (*) Following Instr.	Decode Opcode Calculate Offset	
JRR, JRS	5	1 2 3 4 5	Opcode Address(*) Opcode Address +1(*) Opcode Address +2(*) Opcode Address +2(*) Opcode Address +3(*)	Opcode (*) Operand Address (*) Branch Value Branch Value (*) Following Instr.	Decode Opcode Read Operand Test Operand Fetch Branch Value Calculate New Address	ROM Data Space not Addressed
JRR, JRS	5	1 2 3 4 5	Opcode Address(*) Opcode Address +1(*) Data Space Rom Add.(#) Opcode Address +2(*) Data Space Rom Add.(#)	Opcode (*) Operand Address (*) Rom Data (#) Branch Value (*) Rom Data (#)	Decode Opcode Read Operand Test Operand Fetch Branch Value Calculate New Address	ROM Data Space Addressed
RET	2	1 2	Opcode Address(*) Return Address	Opcode (*) Next Opcode	Decode Opcode Pop Return Address	
RETI	2	1 2	Opcode Address(*) Return Address	Opcode (*) Next Opcode	Decode Opcode Pop Return Address Switch Flag Set	
RLC	4	1 2 3 4	Opcode Address(*) Opcode Address +1 Opcode Address +1 Opcode Address +1	Opcode (*) Next Opcode Next Opcode Next Opcode	Decode Opcode Calculate Acc. Address Read Accumulator Shifted	
STOP, WAIT	2	1 2	Opcode Address(*) Opcode Address +1	Opcode (*) Next Opcode	Decode Opcode Stop/Wait the Oscillator	

Notes: *. Valid only at the beginning of the cycle
#. Valid only until t18 of the cycle

1. Add oscillator build up time plus 16 oscillator clocks if a stop instruction has been executed before the interrupt occured

ADD
Addition

Mnemonic: ADD

Function: Addition

Description: The contents of the source byte is added to the accumulator leaving the result in the accumulator. The source register remains unaltered.

Operation: dst $\leftarrow$ dst + src
The destination must be the accumulator.

Instruction Format	Opcode (Hex)	Bytes	Cycles	Flags	
ADD dst,src				Z	C
ADD A,A	5F FF	2	4	Δ	Δ
ADD A,X	5F 80	2	4	Δ	Δ
ADD A,Y	5F 81	2	4	Δ	Δ
ADD A,V	5F 82	2	4	Δ	Δ
ADD A,W	5F 83	2	4	Δ	Δ
ADD A,(X)	47	1	4	Δ	Δ
ADD A,(Y)	4F	1	4	Δ	Δ
ADD A,rr	5F rr	2	4	Δ	Δ

Notes:
rr.1 Byte dataspace address.
Δ:Z is set if the result is zero. Cleared otherwise.
C is cleared before the operation and than set if there is an overflow from the 8-bit result.

Example: If data space register 22h contains the value 33h and the accumulator holds the value 20h then the instruction,

ADD A,22h

will cause the accumulator to hold 53h (i.e. 33+20).

Addressing Modes: Source: Direct, Indirect
Destination: Accumulator

ADDI

Addition Immediate

Mnemonic: ADDI

Function: Addition Immediate

Description: The immediately addressed data (source) is added to the accumulator leaving the result in the accumulator.

Operation: dst ⟵ dst + src

The destination must be the accumulator.

Instruction Format	Opcode (Hex)	Bytes	Cycles	Flags	
ADDI dst,src				Z	C
ADDI A,nn	57 nn	2	4	Δ	Δ

Notes:
nn.1 Byte immediate data
Δ: Z is set if result is zero. Cleared otherwise
C is cleared before the operation and than set if there is an overflow from the 8-bit result

Example: If the accumulator holds the value 20h then the instruction,

ADDI A,22h

will cause the accumulator to hold 42h (i.e. 22+20).

Addressing Modes:		
	Source:	Immediate
	Destination:	Accumulator

AND
Logical AND

Mnemonic: AND

Function: Logical AND

Description: This instruction logically ANDs the source register and the accumulator. The result is left in the destination register and the source is unaltered.

Operation: dst ⟵src AND dst
The destination must be the accumulator.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
AND dst,src				Z	C
AND A,A	BF FF	2	4	Δ	*
AND A,X	BF 80	2	4	Δ	*
AND A,Y	BF 81	2	4	Δ	*
AND A,V	BF 82	2	4	Δ	*
AND A,W	BF 83	2	4	Δ	*
AND A,(X)	A7	1	4	Δ	*
AND A,(Y)	AF	1	4	Δ	*
AND A,rr	BF rr	2	4	Δ	*

Notes:
rr.1 Byte dataspace address
*.C is unaffected
Δ.Z is set if the result is zero. Cleared otherwise.

Example: If data space register 54h contains the binary value 11110000 and the accumulator contains the binary value 11001100 then the instruction,

AND A,54h

will cause the accumulator to be altered to 11000000.

Addressing Modes:

	Source:	Direct, Indirect.
	Destination:	Accumulator

ANDI

Logical AND Immediate

Mnemonic: ANDI

Function: Logical AND Immediate

Description: This instruction logically ANDs the immediate data byte and the accumulator. The result is left in the accumulator.

Operation: dst ← src AND dst

The source is immediate data and the destination must be the accumulator.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
ANDI dst,src				Z	C
ANDI A,nn	B7 nn	2	4	Δ	*

Notes:

nn.1 Byte immediate data

*.C is unaffected

Δ. Z is set if the result is zero. Cleared otherwise.

Example: If the accumulator contains the binary value 00001111 then the instruction,

ANDI A,33h

will cause the accumulator to hold the value 00000011.

Addressing Modes: Source: Immediate

Destination: Accumulator

CALL

Call Subroutine

Mnemonic: CALL

Function: Call Subroutine

Description: The CALL instruction is used to call a subroutine. It “pushes” the current contents of the program counter (PC) onto the top of the stack. The specified destination address is then loaded into the PC and points to the first instruction of a procedure. At the end of the procedure a RETurn instruction can be used to return to the original program flow. RET pops the top of the stack back into the PC.
Because the ST6 stack is 4 levels deep (ST60) and 6 levels deep (ST62,ST63), a maximum of four/six calls or interrupts may be nested. If more calls are nested, the PC values stacked latest will be lost. In this case returns will return to the PC values stacked first.

Operation: PC ⟵ dst; Top of stack ⟵ PC

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
CALL dst				Z	C
CALL abc	c0001 ab	2	4	*	*

Notes:
abc.the three half bytes of a twelve bit address, the start location of the subroutine.

*. C,Z not affected

Example: If the current PC is 345h then the instruction,

CALL 8DCh

The current PC 345h is pushed onto the top of the stack and the PC will be loaded with the value 8DCh. The next instruction to be executed will be the instruction at 8DCh, the first instruction of the called subroutine.

Addressing Modes: Extended

CLR
Clear

Mnemonic: CLR

Function: Clear

Description: The destination register is cleared to 00h.

Operation: dst ← 0

Inst. Format	OPCDE (Hex)	Bytes	Cycles	Flags	
CLR dst				Z	C
CLR A	DF FF	2	4	Δ	Δ
CLR X	0D 80 00	3	4	*	*
CLR Y	0D 81 00	3	4	*	*
CLR V	0D 82 00	3	4	*	*
CLR W	0D 83 00	3	4	*	*
CLR rr	0D rr 00	3	4	*	*

Notes:
rr. 1 Byte dataspace address
Δ. Z set, Δ. C reset
*. C,Z unaffected

Example: If data space register 22h contains the value 33h,

CLR 22h

will cause register 22h to hold 00h.

Addressing Modes: Direct

COM
Complement

Mnemonic: COM

Function: Complement

Description: This instruction complements each bit of the accumulator; all bits which are set to 1 are cleared to 0 and vice-versa.

Operation: dst ← NOT dst

The destination must be the accumulator.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
COM dst				Z	C
COM A	2D	1	4	Δ	Δ

Note :

Δ:Z is set if the result is zero. Cleared otherwise.

C will contain the value of the MSB before the operation.

Example: If the accumulator contains the binary value 10111001 then the instruction

COM A

will cause the accumulator to be changed to 01000110 and the carry flag to be set (since the original MSB was 1).

Addressing Modes: Inherent

CP
Compare

Mnemonic: CP

Function: Compare

Description: This instruction compares the source byte (subtracted from) with the destination byte, which must be the accumulator. The carry and zero flags record the result of this comparison.

Operation: dst - src

The destination must be the accumulator, but it will not be changed.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
CP dst,src				Z	C
CP A,A	3F FF	2	4	Δ	Δ
CP A,X	3F 80	2	4	Δ	Δ
CP A,Y	3F 81	2	4	Δ	Δ
CP A,V	3F 82	2	4	Δ	Δ
CP A,W	3F 83	2	4	Δ	Δ
CP A,(X)	27	1	4	Δ	Δ
CP A,(Y)	2F	1	4	Δ	Δ
CP A,rr	3F rr	2	4	Δ	Δ

Note: rr. 1 Byte dataspace address

ST60 Δ: Z is set if the result is zero. Cleared otherwise.

C is set if Acc ≥ src, cleared if Acc < src.

ST62/63 Δ: Z is set if the result is zero. Cleared otehrwise.

C is set if Acc < src, cleared if Acc ≥ src.

Example: If the accumulator contains the value 11111000 and the register 34h contains the value 00011100 then the instruction,

CP A,34h

will clear the Zero flag Z and set the Carry flag C, indicating that Acc≥ src (on ST60)

CPI
Compare Immediate

Mnemonic: CPI

Function: Compare Immediate

Description: This instruction compares the immediately addressed source byte (subtracted from) with the destination byte, which must be the accumulator. The carry and zero flags record the result of this comparison.

Operation: dst-src

The source must be the immediately addressed data and the destination must be the accumulator, that will not be changed.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
CPI dst,src				Z	C
CPI A,nn	37 nn	2	4	Δ	Δ

Note: nn.1 Byte immediate data.

ST60 Δ: Z is set if the result is zero. Cleared otherwise.

C is set if Acc ≥ src, cleared if Acc < src.

ST62/63 Δ: Z is set if the result is zero. Cleared otherwise.

C is set if Acc < src, cleared if Acc ≥ src.

Example: If the accumulator contains the value 11111000 then the instruction,

CPI A,00011100B

will clear the Zero flag Z and set the Carry flag C indicating that Acc≥ src (on ST60).

Addressing Modes: Source: Immediate

Destination: Accumulator

DEC

Decrement

Mnemonic: DEC

Function: Decrement

Description: The destination register's contents are decremented by one.

Operation: dst ← dst-1

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
DEC dst				Z	C
DEC A	FF FF	2	4	Δ	*
DEC X	1D	1	4	Δ	*
DEC Y	5D	1	4	Δ	*
DEC V	9D	1	4	Δ	*
DEC W	DD	1	4	Δ	*
DEC (X)	E7	1	4	Δ	*
DEC (Y)	EF	1	4	Δ	*
DEC rr	FF rr	2	4	Δ	*

Notes:
rr.1 Byte dataspace address
*.C is unaffected
Δ.Z is set if the result is zero. Cleared otherwise.

Example: If the X register contains the value 45h and the data space register 45h contains the value 16h then the instruction,

DEC (X)

will cause data space register 45h to contain the value 15h.

Addressing Modes: Short direct, Direct, Indirect.

INC
Increment

Mnemonic: INC

Function: Increment

Description: The destination register's contents are incremented by one.

Operation: dst $\leftarrow$ dst+1

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
INC dst				Z	C
INC A	7F FF	2	4	Δ	*
INC X	15	1	4	Δ	*
INC Y	55	1	4	Δ	*
INC V	95	1	4	Δ	*
INC W	D5	1	4	Δ	*
INC (X)	67	1	4	Δ	*
INC (Y)	6F	1	4	Δ	*
INC rr	7F rr	2	4	Δ	*

Notes:

rr.1 Byte dataspace address

*. C is unaffected

Δ.Z is set if the result is zero. Cleared otherwise.

Example: If the X register contains the value 45h and the data space register 45h contains the value 16h then the instruction

INC (X)

will cause data space register 45h to contain the value 17h.

Addressing Modes: Short direct, Direct, Indirect.

JP
Jump

Mnemonic: JP

Function: Jump (Unconditional)

Description: The JP instruction replaces the PC value with a twelve bit value thus causing a simple jump to another location in the program memory. The previous PC value is lost, not stacked.

Operation: PC ← dst

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
JP dst				Z	C
JP abc	c1001 ab	2	4	*	*

Notes:

abc.the three half bytes of a twelve bit address.

*. C,Z not affected

Example: The instruction,

JP 5CDh

will cause the PC to be loaded with 5CDh and the program will continue from that location.

Addressing Modes: Extended

JRC
Jump Relative on Carry Flag

Mnemonic: JRC

Function: Jump Relative on Carry Flag

Description: This instruction causes the carry (C) flag to be tested and if this flag is set then a jump is performed within the program memory. This jump is in the range -15 to +16 and is relative to the PC value. The displacemente is of five bits. If C=0 than the next instruction is executed.

Operation: If C=1, PC ⟵ PC + e
where e= 5 bit displacement

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
JRC e	e110	1	2	*	*

Notes:

e.5 bit displacement in the range –15 to + 16

*.C,Z not affected

Example: If the carry flag is set then the instruction,

JRC + 8

will cause a branch forward to PC+8. The user can use labels asindentifiers and the assembler will automatically allow the jump if it is in the range -15 to +16.

Addressing Modes: Program Counter Relative

JRNC

Jump Relative on Non Carry Flag

Mnemonic: JRNC

Function: Jump Relative on Non Carry Flag

Description: This instruction causes the carry (C) flag to be tested and if this flag is cleared to zero then a jump is performed within the program memory. This jump is in the range -15 to +16 and is relative to the PC value. The dispacement is of five bits. If C=1 then the next instruction is executed.

Operation: If C=0, PC $\leftarrow$ PC + e
where e= 5 bit displacement

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
JRNC e	e010	1	2	*	*

Notes:

e:5 bit displacement in the range -15 to +16

*:C,Z not affected

Example: If the carry flag is cleared then the instruction,

JRNC -5

will cause a branch backward to PC-5. The user can use labels as identifiers and the assembler will automatically allow the jump if it is in the range -15 to +16.

Addressing Modes: Program Counter Relative

JRNZ
Jump Relative on Non Zero Flag

Mnemonic: JRNZ

Function: Jump Relative on Non Zero Flag

Description: This instruction causes the zero (Z) flag to be tested and if this flag is cleared to zero then a jump is performed within the program memory. This jump is in the range -15 to +16 and is relative to the PC value. The displacement is of five bits. If Z=1 then the next instruction is executed.

Operation: If Z=0, PC $\longleftarrow$ PC + e
where e= 5 bit displacement

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
JRNZ e	e000	1	2	*	*

Notes:

e.5 bit displacement in the range -15 to +16.

*.C,Z not affected

Example: If the zero flag is cleared then the instruction,

JRNZ -5

will cause a branch backward to PC-5. The user can use labels as identifiers and the assembler will automatically allow the jump if it is in the range -15 to +16.

Addressing Modes: Program Counter Relative

JRR
Jump Relative if Reset

Mnemonic: JRR

Function: Jump Relative if RESET

Description: This instruction causes a specified bit in a given dataspace register to be tested. If this bit is reset (=0) then the PC value will be changed and a relative jump will be performed within the program. The relative jump range is -126 to +129. If the tested bit is not reset then the next instruction is executed.

Operation: If bit=0, PC $\longleftarrow$ PC + ee
where ee= 8 bit displacement

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
JRR b,rr,ee	b00011 rr ee	3	5	*	Δ

Notes:

b.3 bit-address

rr.1 Byte dataspace address

ee.8 bit displacement in the range -126 to +129

*.Z is not affected

Δ.The tested bit is shifted into carry.

Example: If bit 4 of dataspace register 70h is reset and the PC=110 then the instruction,

JRR 4, 70h, -20

will cause the PC to be changed to 90 (110-20) and the instruction starting at that address in the program memory to be the next instruction executed.

The user is advised to use labels for conditional jumps. The relative jump will be calculated by the assembler. The jump must be in the range -126 to +129.

Addressing Modes: Bit Test

JRS
Jump Relative if Set

Mnemonic: JRS

Function: Jump Relative if set

Description: This instruction causes a specified bit in a given dataspace register to be tested. If this bit is set (=1) then the PC value will be changed and a relative jump will be performed within the program. The relative jump range is -126 to +129. If the tested bit is not set then the next instruction is executed.

Operation: If bit=1, PC ⟵ PC + ee
where ee= 8 bit displacement

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
JRS b,rr,ee	b10011 rr ee	3	5	*	∆

Notes:

b.3 bit-address

rr.1 Byte dataspace address

ee.8 bit displacement in the range -126 to +129

*.Z is not affected

∆.The tested bit is shifted into carry.

Example: If bit 7 of dataspace register AFh is set and the PC=123 then the instruction,

JRS 7,AFh,+25

will cause the PC to be changed to 148 (123+25) and the instruction starting at that address in the program memory to be the next instruction executed.

The user is advised to use labels for conditional jumps. The relative jump will be calculated by the assembler. The jump must be in the range -126 to +129.

Addressing Modes: Bit Test

JRZ

Jump Relative on Zero Flag

Mnemonic: JRZ

Function: Jump Relative on Zero Flag

Description: This instruction causes the zero (Z) flag to be tested and if this flag is set to one then a jump is performed within the program memory. This jump is in the range -15 to +16 and is relative to the PC value. The displacement is of five bits.
If Z=0 then next instruction is executed.

Operation: If Z=1, PC $\leftarrow$ PC + e
where e= 5 bit displacement

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
JRZ e	e100	1	2	*	*

Notes:

e.5 bit displacement in the range -15 to +16.

*.C,Z not affected

Example: If the zero flag is set then the instruction,

JRZ +8

will cause a branch forward to PC+8. The user can use labels as identifiers and the assembler will automatically allow the jump if it is in the range -15 to +16.

Addressing Modes: Program Counter Relative

LD
Load

Mnemonic: LD

Function: Load

Description: The contents of the source register are loaded into the destination register. The source register remains unaltered and the previous contents of the destination register are lost.

Operation: dst $\leftarrow$ src

Either the source or the destination must be the accumulator.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
LD dst,src				Z	C
LD A,X	35	1	4	Δ	*
LD A,Y	75	1	4	Δ	*
LD A,V	B5	1	4	Δ	*
LD A,W	F5	1	4	Δ	*
LD X,A	3D	1	4	Δ	*
LD Y,A	7D	1	4	Δ	*
LD V,A	BD	1	4	Δ	*
LD W,A	FD	1	4	Δ	*
LD A,(X)	07	1	4	Δ	*
LD (X), A	87	1	4	Δ	*
LD A,(Y)	0F	1	4	Δ	*
LD (Y),A	8F	1	4	Δ	*
LD A,rr	1F rr	2	4	Δ	*
LD rr,A	9F rr	2	4	Δ	*

Notes:

rr.1 Byte dataspace address

*.C not affected

Δ.Z is set if the result is zero. Cleared otherwise.

Example: If data space register 34h contains the value 45h then the instruction;

LD A,34h

will cause the accumulator to be loaded with the value 45h. Register 34h will keep the value 45h.

LDI
Load Immediate

Mnemonic: LDI

Function: Load Immediate

Description: The immediately addressed data (source) is loaded into the destination data space register.

Operation: dst ← src

The source is always an immediate data while the destination can be the accumulator, one of the X,Y,V,W registers or one of the available data space registers.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
LDI dst,src				Z	C
LDI A,nn	17 nn	2	4	Δ	*
LDI X,nn	0D 80 nn	3	4	*	*
LDI Y,nn	0D 81 nn	3	4	*	*
LDI V,nn	0D 82 nn	3	4	*	*
LDI W,nn	0D 83 nn	3	4	*	*
LDI rr,nn	0D rr nn	3	4	*	*

Notes:

rr.1 Byte dataspace address

nn.1 Byte immediate value

*.Z, C not affected

Δ.Z is set if the result is zero. Cleared otherwise.

Example: The instruction

LDI 34h,45h

will cause the value 45h to be loaded into data register at location 34h.

Addressing Modes: Source: Immediate

Destination: Direct

NOP
No Operation

Mnemonic: NOP

Function: No Operation

Description: No action is performed by this instruction. It is typically used for timing delay.

Operation: No Operation

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
NOP	04	1	2	*	*

Note: *. C,Z not affected

Addressing Modes: Program Counter Relative

RES
Reset Bit

Mnemonic: RES

Function: Reset Bit

Description: The RESET instruction is used to reset a specified bit in a given register in the data space.

Operation: dst (n) $\leftarrow$ 0, 0 $\leq$ n $\leq$ 7

Inst. Format	**OPCODE (Hex)**	**Bytes**	**Cycles**	**Flags**	
RES bit,dst				**Z**	**C**
RES b,A	b01011 FF	2	4	*	*
RES b,rr	b01011 rr	2	4	*	*

Notes:

b.3 bit-address

rr.1 Byte dataspace address

*.C,Z not affected

Example: If register 23h of the dataspace contains 11111111 then the instruction,

RES 4,23h

will cause register 23h to hold 11101111.

Addressing Modes: Bit Direct

RET
Return from Subroutine

Mnemonic: RET

Function: Return From Subroutine

Description: This instruction is normally used at the end of a subroutine to return to the previously executed procedure. The previously stacked program counter (stacked during CALL) is popped back from the stack. The next statement executed is that addressed by the new contents of the PC. If the stack had already reached its highest level (no more PC stacked) before the RET is executed, program execution will be continued at the next instruction after the RET.

Operation: PC $\longleftarrow$ Stacked PC

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
RET	CD	1	2	*	*

Note: *. C,Z not affected

Example: If the current PC value is 456h and the PC value at the top of the stack is 3DFh then the instruction,

RET

will cause the PC value 456h to be lost and the current PC value to be 3DFh.

Addressing Modes: Inherent

RETI

Return from Interrupt

Mnemonic: RETI

Function: Return from Interrupt

Description: This instruction marks the end of the interrupt service routine and returns the ST60/62/63 to the state it was in before the interrupt. It "pops" the top (last in) PC value from the stack into the current PC. This instruction also causes the ST60/62/63 to switch from the interrupt flags to the normal flags. The RETI instruction also applies to the end of NMI routine for ST62/63 devices; in this case the instruction causes the switch from NMI flags to normal flags (if NMI was acknowledged inside a normal routine) or to standard interrupt flags (if NMI was acknowledged inside a standard interrupt service routine).

In addition the RETI instruction also clears the interrupt mask (also NMI mask for ST62/63) which was set when the interrupt occurred. If the stack had already reached its highest level (no more PC stacked) before the RETI is executed, program execution will be continued with the next instruction after the RETI. Because the ST60 is in interrupt mode after reset (NMI mode for ST62/63), RETI has to be executed to switch to normal flags and enable interrupts at the end of the starting routine. If no call was executed during the starting routine, program execution will continue with the instruction after the RETI (supposed no interrupt is active).

Operation: Actual Flags $\leftarrow$ Normal Flags (1)

PC $\leftarrow$ Stacked PC

IM $\leftarrow$ 0

(1) Standard Interrupt flags if NMI was acknowledged inside a standard interrupt service (ST62/63 only).

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
RETI	4D	1	2	Δ	Δ

Note: Δ C,Z normal flag will be used from now on.

Example: If the current PC value is 456h and the PC value at the top of the stack is 3DFh then the instruction

RETI

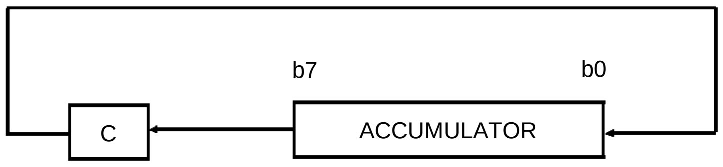
RLC
Rotate Left Through Carry

Mnemonic: RLC

Function: Rotate Left through Carry

Description: This instruction moves each bit in the accumulator one place to the left (i.e. towards the MSBit. The MSBit (bit 7) is moved into the carry flag and the carry flag is moved into the LSBit (bit0) of the accumulator.

Operation:



dst(0) $\leftarrow$ C
C $\leftarrow$dst(7)
dst(n+1) $\leftarrow$ dst(n), $0 \leq n \leq 6$
This instruction can only be performed on the accumulator.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
RLC A	AD	1	4	Δ	Δ

Note : Δ: Z is set if the result is zero. Cleared otherwise.
C will contain the value of the MSB before the operation.

Example: If the accumulator contains the binary value 10001001 and the carry flag is set to 0 then the instruction,

RLC A

will cause the accumulator to have the binary value 00010010 and the carry flag to be set to 1.

Addressing Modes: Inherent

SET
Set Bit

Mnemonic: SET

Function: Set Bit

Description: The SET instruction is used to set a specified bit in a given register in the data space.

Operation: dst (n) $\leftarrow$ 1, 0 $\leq$ n $\leq$ 7

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
SET bit,dst				Z	C
SET b,A	b11011 FF	2	4	*	*
SET b,rr	b11011 rr	2	4	*	*

Notes:
b. 3 bit-address
rr. 1 Byte dataspace address
*. C,Z not affected

Example: If register 23h of the dataspace contains 00000000 then the instruction,

SET 4,23h

will cause register 23h to hold 00010000.

Addressing Modes: Bit Direct

SLA
Shift Left Accumulator

Mnemonic: SLA

Function: Shift Left Accumulator

Description: This instruction implements an addition of the accumulator to itself (i.e a doubling of the accumulator) causing an arithmetic left shift of the value in the register.

Operation: ADD A,FFh

This instruction can only be performed on the accumulator.

Inst. Format	OPCPDE (Hex)	Bytes	Cycles	Flags	
				Z	C
SLA A	5F FF	2	4	Δ	Δ

Note: Δ: Z is set if the result is zero. Cleared otherwise.

C will contain the value of the MSB before the operation.

Example: If the accumulator contains the binary value 11001101 then the instruction,

SLA A

will cause the accumulator to have the binary value 10011010 and the carry flag to be set to 1.

Addressing Modes: Inherent

STOP
Stop Operation

Mnemonic: STOP

Function: Stop operation

Description: This instruction is used for putting the ST60/62/63 into a stand-by mode in which the power consumption is reduced to a minimum. All the on-chip peripherals and oscillator are stopped (for some peripherals, A/D for example, it is necessary to individually turn-off the macrocell before entering the STOP instruction). To restart the processor an external interrupt or a reset is needed.

Operation: Stop Processor

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
STOP	6D	1	2	*	*

Note : *: C,Z not affected

Addressing Mode: Inherent

SUB
Subtraction

Mnemonic: SUB

Function: Subtraction

Description: This instruction subtracts the source value from the destination value.

Operation: dst $\longleftarrow$dst-src
The destination must be the accumulator.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
SUB dst,src				Z	C
SUB A,A	DF FF	2	4	Δ	Δ
SUB A,X	DF 80	2	4	Δ	Δ
SUB A,Y	DF 81	2	4	Δ	Δ
SUB A,V	DF 82	2	4	Δ	Δ
SUB A,W	DF 83	2	4	Δ	Δ
SUB A,(X)	C7	1	4	Δ	Δ
SUB A,(Y)	CF	1	4	Δ	Δ
SUB A,rr	DF rr	2	4	Δ	Δ

Note: rr.1 Byte dataspace address

ST60 Δ: Z is set if the result is zero. Cleared otherwise.

C is set if Acc ≥ src, cleared if Acc < src.

ST62/63 Δ: Z is set if the result is zero. Cleared otherwise.

C is set if Acc < src, cleared if Acc ≥ src.

Example: If the Y register contains the value 23h, dataspace register 23h contains the value 53h and the accumulator contains the value 78h then the instruction,

SUB A,(Y)

will cause the accumulator to hold the value 25h (i.e. 78-53). The zero flag is cleared and the carry flag is set (on ST60), indicating that result is > 0.

Addressing Modes: Source: Indirect,Direct

SUBI
Subtraction Immediate

Mnemonic: SUBI

Function: Subtraction Immediate

Description: This instruction causes the immediately addressed source data to be subtracted from the accumulator.

Operation: dst $\leftarrow$ dst - src
The destination must be the accumulator.

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
SUBI dst,src				Z	C
SUBI A,nn	D7 nn	2	4	Δ	Δ

Note: nn. 1 Byte of immediate data

ST60 Δ**:** Z is set if the result is zero. Cleared otherwise.

C is set if Acc $\geq$ src, cleared if Acc < src.

ST62/63 Δ: Z is set if the result is zero. Cleared otherwise.

C is set if Acc < src, cleared if Acc $\geq$ src.

Example: If the accumulator contains the value 56h then the instruction,

SUBI A,25

will cause the accumulator to contain the value 31h. The zero flag is cleared and the carry flag is set (on ST60), indicating that the result is > 0.

Addressing Modes: Source: Immediate

Destination: Accumulator

WAIT

Wait Processor

Mnemonic: WAIT

Function: Wait Processor

Description: This instruction is used for putting the ST60/62/63 into a stand-by mode in which the power consumption is reduced to a minimum. Instruction execution is stopped, but the oscillator and some on-chip peripherals continue to work. To restart the processor an interrupt from an active on-chip peripheral (eg. timer), an external interrupt or reset is needed. For on-chip peripherals active during wait, see ST60/62/63 data sheets.

Operation: Put ST6 in stand-by mode

Inst. Format	OPCODE (Hex)	Bytes	Cycles	Flags	
				Z	C
WAIT	ED	1	2	*	*

Note : *. C,Z not affected

Addressing Modes: Inherent