Tektronix 4052/4054 opcode Decoding table (gray cells 6800 unused, red on 4052/4054&A, 4052A/4054A ONLY=blue + green 6800 16-bit ext)

MSB \ LSB	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	TEST (INH)	NOP (INH)	NOP (INH)	SFA (INH)	LDAG D (DIR)	LDAG X (DIR)	TAP (INH)	TPA (INH)	INX (INH)	DEX (INH)	CLV (INH)	SEV (INH)	CLC (INH)	SEC (INH)	CLI (INH)	SEI (INH)
1_	SBA (INH)	CBA (INH)	TAPX (INH)	TPAX (INH)	ADXI I (IMM)	ASPI I (IMM)	TAB (INH)	TBA (INH)	SDA (INH)	DAA (INH)	LDXX (INH)	ABA (ACC)	LDAX (INH)	LDBX (INH)	STAX (INH)	JMPAX (INH)
2_	BRA (REL)	SDB (INH)	BHI (REL)	BLS (REL)	BCC (REL)	BCS (REL)	BNE (REL)	BEQ (REL)	BVC (REL)	BVS (REL)	BPL (REL)	BMI (REL)	BGE (REL)	BLT (REL)	BGT (REL)	BLE (REL)
3_	TSX (INH)	INS (INH)	PUL A (ACC)	PUL B (ACC)	DES (INH)	TXS (INH)	PSH A (ACC)	PSH B (ACC)	JMPIN (EXT)	RTS (INH)	FPSH D (DIR)	RTI (INH)	FPSH X (IDX)	FPSH (EXT)	WAI (INH)	SWI (INH)
4_	NEG A (ACC)	FPSH I (IMM*)	FPUL D (DIR)	COM A (ACC)	LSR A (ACC)	FPUL X (IDX)	ROR A (ACC)	ASR A (ACC)	ASL A (ACC)	ROL A (ACC)	DEC A (ACC)	FPUL (EXT)	INC A (ACC)	TST A (ACC)	FDUP (INH)	CLR A (ACC)
5_	NEG B (ACC)	FSWAP (INH)	FADD (INH)	COM B (ACC)	LSR B (ACC)	FSUB (INH)	ROR B (ACC)	ASR B (ACC)	ASL B (ACC)	ROL B (ACC)	DEC B (ACC)	FMUL (INH)	INC B (ACC)	TST B (ACC)	FDIV (INH)	CLR B (ACC)
6_	NEG X (IDX)	FNRM (INH)	PSHRET (DIR)	COM (IDX)	LSR (IDX)	RTRN (DIR)	ROR (IDX)	ASR (IDX)	ASL (IDX)	ROL (IDX)	DEC (IDX)	PSHX (INH)	INC (IDX)	TST (IDX)	JMP (IDX)	CLR (IDX)
7_	NEG (EXT)	STRK (INH)	VECT (INH)	COM (EXT)	LSR (EXT)	PULX (INH)	ROR (EXT)	ASR (EXT)	ASL (EXT)	ROL (EXT)	DEC (EXT)	STAG D (DIR)	INC (EXT)	TST (EXT)	JMP (EXT)	CLR (EXT)
8_	SUB A (IMM)	CMP A (IMM)	SBC A (IMM)	STAG X (IDX)	AND A (IMM)	BIT A (IMM)	LDA A (IMM)	ADDG D (DIR)	EOR A (IMM)	ADC A (IMM)	ORA A (IMM)	ADD A (IMM)	CPX A (IMM)	BSR (REL)	LDS (IMM)	ADDG X (IDX)
9_	SUB A (DIR)	CMP A (DIR)	SBC A (DIR)	SUBD G (DIR)	AND A (DIR)	BIT A (DIR)	LDA A (DIR)	STA A (DIR)	EOR A (DIR)	ADC A (DIR)	ORA A (DIR)	ADD A (DIR)	CPX A (DIR)	SUBD X (IDX)	LDS (DIR)	STS (DIR)
A_	SUB A (IDX)	CMP A (IDX)	SBC A (IDX)	INXSTX (DIR)	AND A (IDX)	BIT A (IDX)	LDA A (IDX)	STA A (IDX)	EOR A (IDX)	ADC A (IDX)	ORA A (IDX)	ADD A (IDX)	CPX A (IDX)	JSR (IDX)	LDS (IDX)	STS (IDX)
B_	SUB A (EXT)	CMP A (EXT)	SBC A (EXT)	LDAG (EXT)	AND A (EXT)	BIT A (EXT)	LDA A (EXT)	STA A (EXT)	EOR A (EXT)	ADC A (EXT)	ORA A (EXT)	ADD A (EXT)	CPX A (EXT)	JSR (EXT)	LDS (EXT)	STS (EXT)
C_	SUB B (IMM)	CMP B (IMM)	SBC B (IMM)	STAG (EXT)	AND B (IMM)	BIT B (IMM)	LDA B (IMM)	C7-ESC	EOR B (IMM)	ADC B (IMM)	ORA B (IMM)	ADD B (IMM)	ADAX (INH)	WAGDX (INH)	LDX (IMM)	
D_	SUB B (DIR)	CMP B (DIR)	SBC B (DIR)	LDAG I (EXT)	AND B (DIR)	BIT B (DIR)	LDA B (DIR)	STA B (DIR)	EOR B (DIR)	ADC B (DIR)	ORA B (DIR)	ADD B (DIR)	SBUG (INH)	CBUG (INH)	LDX (DIR)	STX (DIR)
E_	SUB B (IDX)	CMP B (IDX)	SBC B (IDX)	MOVLR (INH)	AND B (IDX)	BIT B (IDX)	LDA B (IDX)	STA B (IDX)	EOR B (IDX)	ADC B (IDX)	ORA B (IDX)	ADD B (IDX)	MOVRL (INH)	WADX (EXTI)	LDX (IDX)	STX (IDX)
F_	SUB B (EXT)	CMP B (EXT)	SBC B (EXT)	CPCH (INH)	AND B (EXT)	BIT B (EXT)	LDA B (EXT)	STA B (EXT)	EOR B (EXT)	ADC B (EXT)	ORA B (EXT)	ADD B (EXT)	FC-ESC	PCH (IMM)	LDX (EXT)	STX (EXT)
FC_	PSHG (INH)	PULG (INH)	ADDG I (EXTI)	ADDG (EXT)	SUBG I (EXTI)	SUBG (EXT)	CMPGX (INH)	CMPSYM (INH)	LDAGX (INH)	STAGX (INH)						
C7_	TGX (INH)	TXG (INH)	CLRGH (INH)	IFLOAT (INH)	FIXRND (INH)	TMULT (INH)	BUFIN (INH)	BUFOUT (INH)	SEABNK (INH)	DEVIN (INH)	DEVOUT (INH)					

Abbreviations:

4052/4054 and 4052A/4054A Addressing modes (same as 6800):

ACC - Accumulator

In accumulator addressing, either accumulator A or accumulator B is specified. These are 1- byte instructions.

Ex: ABA adds the contents of accumulators and stores the result in accumulator A

IMM - Immediate

In immediate addressing, operand is located immediately after the opcode in the second byte of the instruction in program memory (except LDS and LDX where the operand is in the second and third bytes of the instruction). These are 2-byte or 3-byte instructions.

Ex: LDAA #$25 loads the number $(25)_H$ into accumulator A

DIR - Direct

In direct addressing, the address of the operand is contained in the second byte of the instruction. Direct addressing allows the user to directly address the lowest 256 bytes of the memory, i.e, locations 0 through 255. Enhanced execution times are achieved by storing data in these locations. These are 2-byte instructions.

Ex: LDAA $25 loads the contents of the memory address $(25)_H$ into accumulator A

EXT - Extended

In extended addressing, the address contained in the second byte of the instruction is used as the higher eight bits of the address of the operand. The third byte of the instruction is used as the lower eight bits of the address for the operand. This is an absolute address in the memory. These are 3-byte instructions.

Ex: LDAA $1000 loads the contents of the memory address $(1000)_H$ into accumulator A

IDX - Indexed

In indexed addressing, the address contained in the second byte of the instruction is added to the index register’s lowest eight bits. The carry is then added to the higher order eight bits of the index register. This result is then used to address memory. The modified address is held in a temporary address register so there is no change to the index register. These are 2-byte instructions.

Ex: LDX #$1000 or **LDAA $10,X**

Initially, LDX #$1000 instruction loads 1000_H to the index register (X) using immediate addressing. Then LDAA $10,X instruction, using indexed addressing, loads the contents of memory address $(10)_H$ + X = 1010_H into accumulator A.

INH - Implied (Inherent)

In the implied addressing mode, the instruction gives the address inherently (i.e., stack pointer, index register, etc.). Inherent instructions are used when no operands need to be fetched. These are 1-byte instructions.

Ex: INX increases the contents of the Index register by one. The address information is "inherent" in the instruction itself.

INCA increases the contents of the accumulator A by one.

DECB decreases the contents of the accumulator B by one.

REL - Relative

The relative addressing mode is used with most of the branching instructions on the 6802 microprocessor. The first byte of the instruction is the opcode. The second byte of the instruction is called the *offset*. The offset is interpreted as a *signed 7-bit number*. If the MSB (most significant bit) of the offset is 0, the number is positive, which indicates a forward branch. If the MSB of the offset is 1, the number is negative, which indicates a backward branch. This allows the user to address data in a range of -126 to +129 bytes of the present instruction. These are 2-byte instructions.

Ex:

```
PC    Hex Label  Instruction
0009  2004       BRA 0FH
```

Data Space   - A 0x0000-FFFF 56KB of DRAM + 8KB of DATA ROM

Fetch Space - B 0x0000-FFFF 48KB of BASIC ROM at 0x4000-0xFFFF plus 16KB of bank switched BASIC or option ROM Pack at 0x0000

6800, 4052/4054 &A and 4052A/4054A only registers:

- **ACCA** Accumulator A
  - **Extended to 16-bits**
  - **ACCA is low order 8 bits**
- **ACCG is 16-bit extension of A where A is low order 8-bits**
- **ACCB** Accumulator B
  - **Extended to 16-bits**
  - **ACCB is low order 8 bits**
- **ACCX is Accumulator ACCA or ACCB**
- **X** Index register (XH and XL)
- **PC** Program Counter (PCH and PCL)
- **SP** Stack Pointer (SPH and SPL)
- **CC** Status register

CC status register:

Bit 0	**C** Carry/Borrow status
Bit 1	**V** Two's complement / overflow indicator
Bit 2	**Z** Zero status
Bit 3	**N** Sign/Negative status
Bit 4	**I** Interrupt Mask status
Bit 5	**H** Half carry
Bit 6	**D** Data Space Indicator (1 → A)
Bit 7	**F** Fetch Space Indicator (1 → B)

Symbols in the STATUSES column:

- **(blank)** operation does not affect status
- **x** operation affects status
- **0** flag is cleared by the operation
- **1** flag is set by the operation

data8 8-bit immediate data
data16 16-bit immediate data
addr8 8-bit direct address
addr16 16-bit extended address
disp 8-bit signed address displacement

(HI) bits 15-8 from 16bit value
(LO) bits 7-0 from 16bit value
[...] content of ...
[[...]] implied addressing (content of [content of ...])
∧ Logical AND
∨ Logical OR
⊻ Logical Exclusive-OR
← Data is transferred in the direction of the arrow

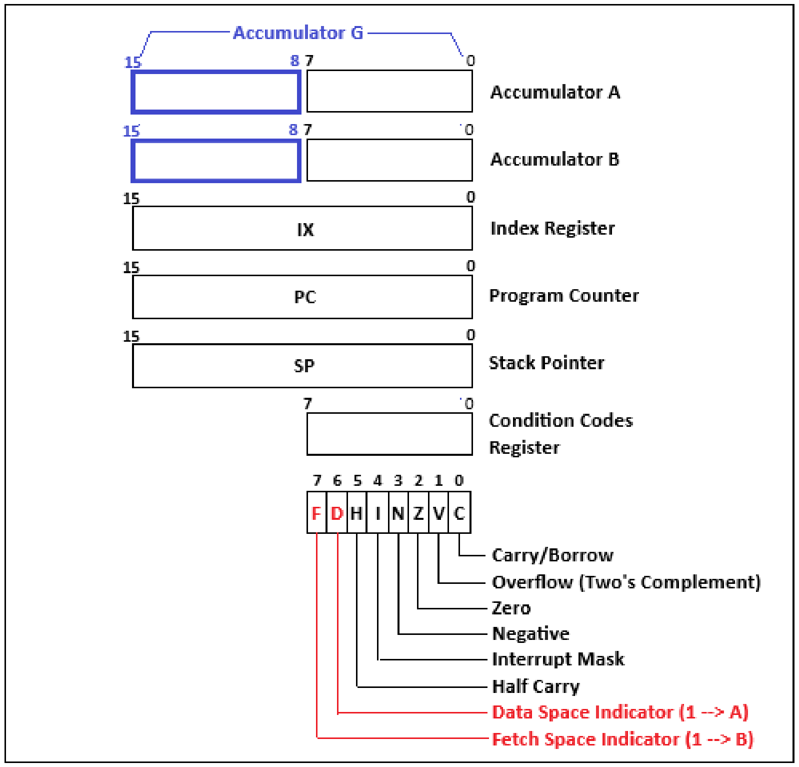

Figure 1- 4052/4054 and 4052A/4054A Registers

6800 OPCODE DETAILS

MNEMO	SYNTAX	MODE	BYTES	CODE	CYCLES	C	Z	S	O	A_c	I	SYMBOLIC OPERATION	DESCRIPTION
ABA	ABA	ACC	1	$1B	2	x	x	x	x	x	-	[A] ← [A] + [B]	Add B to A
ADC	ADC A #data8	IMM	2	$89	2	x	x	x	x	x	-	[A] ← [A] + data8 + C	Add contents of Memory + Carry Flag to Accumulator
	ADC A addr8	DIR	2	$99	3							[A] ← [A] + [addr8] + C	
	ADC A data8,X	IDX	2	$A9	5							[A] ← [A] + [data8 + [X]] + C	
	ADC A addr16	EXT	3	$B9	4							[A] ← [A] + [addr16] + C	
	ADC B #data8	IMM	2	$C9	2							[B] ← [B] + data8 + C	
	ADC B addr8	DIR	2	$D9	3							[B] ← [B] + [addr8] + C	
	ADC B data8,X	IDX	2	$E9	5							[B] ← [B] + [data8 + [X]] + C	
	ADC B addr16	EXT	3	$F9	4							[B] ← [B] + [addr16] + C	
ADD	ADD A #data8	IMM	2	$8B	2	x	x	x	x	x	-	[A] ← [A] + data8	Add Memory contents to the Accumulator
	ADD A addr8	DIR	2	$9B	3							[A] ← [A] + [addr8]	
	ADD A data8,X	IDX	2	$AB	5							[A] ← [A] + [data8 + [X]]	

	ADD A addr16	EXT	3	$BB	4							[A] ← [A] + [addr16]	
	ADD B #data8	IMM	2	$CB	2							[B] ← [B] + data8	
	ADD B addr8	DIR	2	$DB	3							[B] ← [B] + [addr8]	
	ADD B data8,X	IDX	2	$EB	5							[B] ← [B] + [data8 + [X]]	
	ADD B addr16	EXT	3	$FB	4							[B] ← [B] + [addr16]	
AND	AND A #data8	IMM	2	$84	2	-	x	x	0	-	-	[A] ← [A] ∧ data8	Memory contents AND the Accumulator to the Accumulator
	AND A addr8	DIR	2	$94	3							[A] ← [A] ∧ [addr8]	
	AND A data8,X	IDX	2	$A4	5							[A] ← [A] ∧ [data8 + [X]]	
	AND A addr16	EXT	3	$B4	4							[A] ← [A] ∧ [addr16]	
	AND B #data8	IMM	2	$C4	2							[B] ← [B] ∧ data8	
	AND B addr8	DIR	2	$D4	3							[B] ← [B] ∧ [addr8]	
	AND B data8,X	IDX	2	$E4	5							[B] ← [B] ∧ [data8 + [X]]	
	AND B addr16	EXT	3	$F4	4							[B] ← [B] ∧ [addr16]	
ASL	ASL A	ACC	1	$48	2	x	x	x	x	-	-		

	ASL B	ACC	1	$58	2							C ← $\boxed{76543210}$ ← 0	Arithmetic Shift Left. Bit 0 is set to 0. (multiplying by two)
	ASL data8,X	IDX	2	$68	7								
	ASL addr16	EXT	3	$78	6								
ASR	ASR A	ACC	1	$47	2	x	x	x	x	-	-	$\boxed{76543210}$ → C	Arithmetic Shift Right. Bit 7 stays the same.
	ASR B	ACC	1	$57	2								
	ASR data8,X	IDX	2	$67	7								
	ASR addr16	EXT	3	$77	6								
BCC	BCC disp	REL	2	$24	4	-	-	-	-	-	-	(C == 0) ? {[PC] ← [PC] + disp + 2}	Branch if carry clear
BCS	BCS disp	REL	2	$25	4	-	-	-	-	-	-	(C == 1) ? {[PC] ← [PC] + disp + 2}	Branch if carry set
BEQ	BEQ disp	REL	2	$27	4	-	-	-	-	-	-	(Z == 1) ? {[PC] ← [PC] + disp + 2}	Branch if equal to zero
BGE	BGE disp	REL	2	$2C	4	-	-	-	-	-	-	(S ⊻ O == 0) ? {[PC] ← [PC] + disp + 2}	Branch if greater than or equal to zero
BGT	BGT disp	REL	2	$2E	4	-	-	-	-	-	-	(Z ⊻ (S ⊻ O) == 0) ? {[PC] ← [PC] + disp + 2}	Branch if greater than zero

BHI	BHI disp	REL	2	$22	4	-	-	-	-	-	-	(C ∨ Z == 0) ? {[PC] ← [PC] + disp + 2}	Branch if Accumulator contents higher than comparand
BIT	BIT A #data8	IMM	2	$85	2	-	x	x	0	-	-	[A] ∧ data8	Memory contents AND the Accumulator, but only Status register is affected.
	BIT A addr8	DIR	2	$95	3							[A] ∧ [addr8]	
	BIT A data8,X	IDX	2	$A5	5							[A] ∧ [data8 + [X]]	
	BIT A addr16	EXT	3	$B5	4							[A] ∧ [addr16]	
	BIT B #data8	IMM	2	$C5	2							[B] ∧ data8	
	BIT B addr8	DIR	2	$D5	3							[B] ∧ [addr8]	
	BIT B data8,X	IDX	2	$E5	5							[B] ∧ [data8 + [X]]	
	BIT B addr16	EXT	3	$F5	4							[B] ∧ [addr16]	
BLE	BLE disp	REL	2	$2F	4	-	-	-	-	-	-	(Z ∨ (S ⊻ O) == 1) ? {[PC] ← [PC] + disp + 2}	Branch if less than or equal to zero
BLS	BLS disp	REL	2	$23	4	-	-	-	-	-	-	(C ∨ Z == 1) ? {[PC] ← [PC] + disp + 2}	Branch if Accumulator contents less than or same as comparand
BLT	BLT disp	REL	2	$2D	4	-	-	-	-	-	-	(S ⊻ O == 1) ? {[PC] ← [PC] + disp + 2}	Branch if less than zero

BMI	BMI disp	REL	2	$2B	4	-	-	-	-	-	-	(S == 1) ? {[PC] ← [PC] + disp + 2}	Branch if minus
BNE	BNE disp	REL	2	$26	4	-	-	-	-	-	-	(Z == 0) ? {[PC] ← [PC] + disp + 2}	Branch if not equal to zero
BPL	BPL disp	REL	2	$2A	4	-	-	-	-	-	-	(S == 0) ? {[PC] ← [PC] + disp + 2}	Branch if plus
BRA	BRA disp	REL	2	$20	4	-	-	-	-	-	-	[PC] ← [PC] + disp + 2	Unconditional branch relative to present Program Counter contents.
BSR	BSR disp	REL	2	$8D	8	-	-	-	-	-	-	[[SP]] ← [PC(LO)], [[SP] - 1] ← [PC(HI)], [SP] ← [SP] - 2, [PC] ← [PC] + disp + 2	Unconditional branch to subroutine located relative to present Program Counter contents.
BVC	BVC disp	REL	2	$28	4	-	-	-	-	-	-	(O == 0) ? {[PC] ← [PC] + disp + 2}	Branch if overflow clear
BVS	BVS disp	REL	2	$29	4	-	-	-	-	-	-	(O == 1) ? {[PC] ← [PC] + disp + 2}	Branch if overflow set
CBA	CBA	INH	1	$11	2	x	x	x	x	-	-	[A] - [B]	Compare contents of Accumulators A and B. Only the Status register is affected.
CLC	CLC	INH	1	$0C	2	0	-	-	-	-	-	C ← 0	Clear the Carry Flag

CLI	CLI	INH	1	$0E	2	-	-	-	-	-	0	I ← 0	Clear the Interrupt flag to enable interrupts
CLR	CLR A	ACC	1	$4F	2	0	1	0	0	-	-	[A] ← 0	Clear the Accumulator
	CLR B	ACC	1	$5F	2							[B] ← 0	
	CLR data8,X	IDX	2	$6F	7							[data8 + [X]] ← 0	Clear the Memory location
	CLR addr16	EXT	3	$7F	6							[addr16] ← 0	
CLV	CLV	INH	1	$0A	2	-	-	-	0	-	-	O ← 0	Clear the Overflow flag
CMP	CMP A #data8	IMM	2	$81	2	x	x	x	x	-	-	[A] - data8	Compare the contents of Memory and Accumulator. Only the Status register is affected.
	CMP A addr8	DIR	2	$91	3							[A] - [addr8]	
	CMP A data8,X	IDX	2	$A1	5							[A] - [data8 + [X]]	
	CMP A addr16	EXT	3	$B1	4							[A] - [addr16]	
	CMP B #data8	IMM	2	$C1	2							[B] - data8	
	CMP B addr8	DIR	2	$D1	3							[B] - [addr8]	
	CMP B data8,X	IDX	2	$E1	5							[B] - [data8 + [X]]	
	CMP B addr16	EXT	3	$F1	4							[B] - [addr16]	

COM	COM A	ACC	1	$43	2	1	x	x	0	-	-	[A] ← $FF - [A]	Complement the Accumulator
	COM B	ACC	1	$53	2							[B] ← $FF - [B]	
	COM data8,X	IDX	2	$63	7							[data8 + [X]] ← $FF - [data8 + [X]]	Complement the Memory Location
	COM addr16	EXT	3	$73	6							[addr16] ← $FF - [addr16]	
CPX	CPX addr8	DIR	2	$9C	4	-	x	x	x	-	-	[X(HI)] - [addr8], [X(LO)] - [addr8 + 1]	Compare the contents of Memory to the Index Register X
	CPX data8,X	IDX	2	$AC	6							[X(HI)] - [data8 + [X]], [X(LO)] - [data8 + [X] + 1]	
	CPX #data16	IMM	3	$8C	3							[X(HI)] - data16(HI), [X(LO)] - data16(LO)	
	CPX addr16	EXT	3	$BC	5							[X(HI)] - [addr16(HI)], [X(LO)] - [addr16(LO)]	
DAA	DAA	INH	1	$19	2	x	x	x	x	-	-		Decimal Adjust Accumulator A
DEC	DEC A	ACC	1	$4A	2	-	x	x	x	-	-	[A] ← [A] - 1	Decrement the Accumulator
	DEC B	ACC	1	$5A	2							[B] ← [B] - 1	

	DEC data8,X	IDX	2	$6A	7							[data8 + [X]] ⟵ [data8 + [X]] - 1	Decrement the Memory Location
	DEC addr16	EXT	3	$7A	6							[addr16] ⟵ [addr16] - 1	
DES	DES	INH	1	$34	4	-	-	-	-	-	-	[SP] ⟵ [SP] - 1	Decrement the Stack Pointer
DEX	DEX	INH	1	$09	4	-	x	-	-	-	-	[X] ⟵ [X] - 1	Decrement the Index Register X
EOR	EOR A #data8	IMM	2	$88	2	-	x	x	0	-	-	[A] ⟵ [A] ⊻ data8	Memory contents EXLCLUSIVE OR the Accumulator
	EOR A addr8	DIR	2	$98	3							[A] ⟵ [A] ⊻ [addr8]	
	EOR A data8,X	IDX	2	$A8	5							[A] ⟵ [A] ⊻ [data8 + [X]]	
	EOR A addr16	EXT	3	$B8	4							[A] ⟵ [A] ⊻ [addr16]	
	EOR B #data8	IMM	2	$C8	2							[B] ⟵ [B] ⊻ data8	
	EOR B addr8	DIR	2	$D8	3							[B] ⟵ [B] ⊻ [addr8]	
	EOR B data8,X	IDX	2	$E8	5							[B] ⟵ [B] ⊻ [data8 + [X]]	
	EOR B addr16	EXT	3	$F8	4							[B] ⟵ [B] ⊻ [addr16]	
INC	INC A	ACC	1	$4C	2	-	x	x	x	-	-	[A] ⟵ [A] + 1	Increment the Accumulator

	INC B	ACC	1	$5C	2							[B] ← [B] + 1	
	INC data8,X	IDX	2	$6C	7							[data8 + [X]] ← [data8 + [X]] + 1	Increment the Memory Location
	INC addr16	EXT	3	$7C	6							[addr16] ← [addr16] + 1	
INS	INS	INH	1	$31	4	-	-	-	-	-	-	[SP] ← [SP] + 1	Increment the Stack Pointer
INX	INX	INH	1	$08	4	-	x	-	-	-	-	[X] ← [X] + 1	Increment the Index Register X
JMP	JMP data8,X	IDX	2	$6E	4	-	-	-	-	-	-	[PC] ← data8 + [X]	Jump
	JMP addr16	EXT	3	$7E	3							[PC] ← addr16	
JSR	JSR data8,X	IDX	2	$AD	8	-	-	-	-	-	-	[[SP]] ← [PC(LO)], [[SP] - 1] ← [PC(HI)], [SP] ← [SP] - 2, [PC] ← data8 + [X]	Jump to Subroutine
	JSR addr16	EXT	3	$BD	9							[[SP]] ← [PC(LO)], [[SP] - 1] ← [PC(HI)], [SP] ← [SP] - 2, [PC] ← addr16	
LDA	LDA A #data8	IMM	2	$86	2	-	x	x	0	-	-	[A] ← data8	Load Accumulator from Memory
	LDA A addr8	DIR	2	$96	3							[A] ← [addr8]	

	LDA A data8,X	IDX	2	$A6	5							[A] ← [data8 + [X]]	
	LDA A addr16	EXT	3	$B6	4							[A] ← [addr16]	
	LDA B #data8	IMM	2	$C6	2							[B] ← data8	
	LDA B addr8	DIR	2	$D6	3							[B] ← [addr8]	
	LDA B data8,X	IDX	2	$E6	5							[B] ← [data8 + [X]]	
	LDA B addr16	EXT	3	$F6	4							[B] ← [addr16]	
LDS	LDS addr8	DIR	2	$9E	4	-	x	x	0	-	-	[SP(HI)] ← [addr8], [SP(LO)] ← [addr8 + 1]	Load the Stack Pointer
	LDS data8,X	IDX	2	$AE	6							[SP(HI)] ← [data8 + [X]], [SP(LO)] ← [data8 + [X] + 1]	
	LDS #data16	IMM	3	$8E	3							[SP(HI)] ← data16(HI), [SP(LO)] ← data16(LO)	
	LDS addr16	EXT	3	$BE	5							[SP(HI)] ← [addr16(HI)], [SP(LO)] ← [addr16(LO)]	
LDX	LDX addr8	DIR	2	$DE	4	-	x	x	0	-	-	[X(HI)] ← [addr8], [X(LO)] ← [addr8 + 1]	Load the Index Register
	LDX data8,X	IDX	2	$EE	6							[X(HI)] ← [data8 + [X]], [X(LO)] ← [data8 + [X] + 1]	

	LDX #data16	IMM	3	$CE	3							[X(HI)] ← data16(HI), [X(LO)] ← data16(LO)	
	LDX addr16	EXT	3	$FE	5							[X(HI)] ← [addr16(HI)], [X(LO)] ← [addr16(LO)]	
LSR	LSR A	ACC	1	$44	2	x	x	0	x	-	-	0 → $\boxed{76543210}$ → C	Logical Shift Right. Bit 7 is set to 0. (dividing by two)
	LSR B	ACC	1	$54	2								
	LSR data8,X	IDX	2	$64	7								
	LSR addr16	EXT	3	$74	6								
NEG	NEG A	ACC	1	$40	2	x	x	x	x	-	-	[A] ← 0 - [A]	Negate the Accumulator
	NEG B	ACC	1	$50	2							[B] ← 0 - [B]	
	NEG data8,X	IDX	2	$60	7							[data8 + [X]] ← 0 - [data8 + [X]]	Negate the Memory Location
	NEG addr16	EXT	3	$70	6							[addr16] ← 0 - [addr16]	
NOP	NOP	INH	1	$01	2	-	-	-	-	-	-		No Operation
ORA	ORA A #data8	IMM	2	$8A	2	-	x	x	0	-	-	[A] ← [A] ∨ data8	OR the Accumulator
	ORA A addr8	DIR	2	$9A	3							[A] ← [A] ∨ [addr8]	

	ORA A data8,X	IDX	2	$AA	5							[A] ← [A] ∨ [data8 + [X]]	
	ORA A addr16	EXT	3	$BA	4							[A] ← [A] ∨ [addr16]	
	ORA B #data8	IMM	2	$CA	2							[B] ← [B] ∨ data8	
	ORA B addr8	DIR	2	$DA	3							[B] ← [B] ∨ [addr8]	
	ORA B data8,X	IDX	2	$EA	5							[B] ← [B] ∨ [data8 + [X]]	
	ORA B addr16	EXT	3	$FA	4							[B] ← [B] ∨ [addr16]	
PSH	PSH A	ACC	1	$36	4	-	-	-	-	-	-	[[SP]] ← [A], [SP] ← [SP] - 1	Push Accumulator onto the Stack
	PSH B	ACC	1	$37	4							[[SP]] ← [B], [SP] ← [SP] - 1	
PUL	PUL A	ACC	1	$32	4	-	-	-	-	-	-	[SP] ← [SP] + 1, [A] ← [[SP]]	Pull Data from Stack to Accumulator
	PUL B	ACC	1	$33	4							[SP] ← [SP] + 1, [B] ← [[SP]]	
ROL	ROL A	ACC	1	$49	2	x	x	x	x	-	-	C ← 76543210 ← C	Rotate left through Carry.
	ROL B	ACC	1	$59	2								
	ROL data8,X	IDX	2	$69	7								

	ROL addr16	EXT	3	$79	6								
ROR	ROR A	ACC	1	$46	2	x	x	x	x	-	-	C → 76543210 → C	Rotate right through Carry.
	ROR B	ACC	1	$56	2								
	ROR data8,X	IDX	2	$66	7								
	ROR addr16	EXT	3	$76	6								
RTI	RTI	INH	1	$3B	10	x	x	x	x	x	x	[SR] ← [[SP] + 1], [B] ← [[SP] + 2], [A] ← [[SP] + 3], [X(HI)] ← [[SP] + 4], [X(LO)] ← [[SP] + 5], [PC(HI)] ← [[SP] + 6], [PC(LO)] ← [[SP] + 7], [SP] ← [SP] + 7	Return from interrupt. Put registers from Stack and increment Stack Pointer.
RTS	RTS	INH	1	$39	5	-	-	-	-	-	-	[PC(HI)] ← [[SP] + 1], [PC(LO)] ← [[SP] + 2], [SP] ← [SP] + 2	Return from subroutine. Pull PC from top of Stack and increment Stack Pointer.
SBA	SBA	INH	1	$10	2	x	x	x	x	-	-	[A] ← [A] - [B]	Subtract contents of Accumulator B from those of Accumulator A.
SBC	SBC A #data8	IMM	2	$82	2	x	x	x	x	-	-	[A] ← [A] - data8 - C	Subtract Mem and Carry Flag from Accumulator
	SBC A addr8	DIR	2	$92	3							[A] ← [A] - [addr8] - C	

	SBC A data8,X	IDX	2	$A2	5							[A] ← [A] - [data8 + [X]] - C	
	SBC A addr16	EXT	3	$B2	4							[A] ← [A] - [addr16] - C	
	SBC B #data8	IMM	2	$C2	2							[B] ← [B] - data8 - C	
	SBC B addr8	DIR	2	$D2	3							[B] ← [B] - [addr8] - C	
	SBC B data8,X	IDX	2	$E2	5							[B] ← [B] - [data8 + [X]] - C	
	SBC B addr16	EXT	3	$F2	4							[B] ← [B] - [addr16] - C	
SEC	SEC	INH	1	$0D	2	1	-	-	-	-	-	C ← 1	Set the Carry Flag
SEI	SEI	INH	1	$0F	2	-	-	-	-	-	1	I ← 1	Set the Interrupt Flag to disable interrupts
SEV	SEV	INH	1	$0B	2	-	-	-	1	-	-	O ← 1	Set the Overflow Flag
	STA A addr8	DIR	2	$97	4							[addr8] ← [A]	
	STA A data8,X	IDX	2	$A7	6							[data8 + [X]] ← [A]	
STA	STA A addr16	EXT	3	$B7	5	-	x	x	0	-	-	[addr16] ← [A]	Store Accumulator in Memory
	STA B addr8	DIR	2	$D7	4							[addr8] ← [B]	
	STA B data8,X	IDX	2	$E7	6							[data8 + [X]] ← [B]	

	STA B addr16	EXT	3	$F7	5							[addr16] ← [B]	
	STS addr8	DIR	2	$9F	5							[addr8] ← [SP(HI)], [addr8 + 1] ← [SP(LO)]	
STS	STS data8,X	IDX	2	$AF	7	-	x	x	0	-	-	[data8 + [X]] ← [SP(HI)], [data8 + [X] + 1] ← [SP(LO)]	Store the Stack Pointer
	STS addr16	EXT	3	$BF	6							[addr16(HI)] ← [SP(HI)], [addr16(LO)] ← [SP(LO)]	
	STX addr8	DIR	2	$DF	5							[addr8] ← [X(HI)], [addr8 + 1] ← [X(LO)]	
STX	STX data8,X	IDX	2	$EF	7	-	x	x	0	-	-	[data8 + [X]] ← [X(HI)], [data8 + [X] + 1] ← [X(LO)]	Store the Index Register X
	STX addr16	EXT	3	$FF	6							[addr16(HI)] ← [X(HI)], [addr16(LO)] ← [X(LO)]	
	SUB A #data8	IMM	2	$80	2							[A] ← [A] - data8	
	SUB A addr8	DIR	2	$90	3							[A] ← [A] - [addr8]	
SUB	SUB A data8,X	IDX	2	$A0	5	x	x	x	x	-	-	[A] ← [A] - [data8 + [X]]	Subtract Memory contents from Accumulator
	SUB A addr16	EXT	3	$B0	4							[A] ← [A] - [addr16]	
	SUB B #data8	IMM	2	$C0	2							[B] ← [B] - data8	

	SUB B addr8	DIR	2	$D0	3							[B] ← [B] - [addr8]	
	SUB B data8,X	IDX	2	$E0	5							[B] ← [B] - [data8 + [X]]	
	SUB B addr16	EXT	3	$F0	4							[B] ← [B] - [addr16]	
SWI	SWI	INH	1	$3F	12	-	-	-	-	-	1	[[SP]] ← [PC(LO)], [[SP] - 1] ← [PC(HI)], [[SP] - 2] ← [X(LO)], [[SP] - 3] ← [X(HI)], [[SP] - 4] ← [A], [[SP] - 5] ← [B], [[SP] - 6] ← [SR], [SP] ← [SP] - 7, [PC(HI)] ← [$FFFA], [PC(LO)] ← [$FFFB]	Software Interrupt: push registers onto Stack, decrement Stack Pointer, and jump to interrupt subroutine.
TAB	TAB	INH	1	$16	2	-	x	x	0	-	-	[B] ← [A]	Transfer A to B
TAP	TAP	INH	1	$06	2	x	x	x	x	x	-	[SR] ← [A]	Transfer A to Status Register
TBA	TBA	INH	1	$17	2	-	x	x	0	-	-	[A] ← [B]	Transfer B to A
TPA	TPA	INH	1	$07	2	-	-	-	-	-	-	[A] ← [SR]	Transfer Status Register to A
TST	TST A	ACC	1	$4D	2	0	x	x	0	-	-	[A] - 0	Test the Accumulator
	TST B	ACC	1	$5D	2							[B] - 0	
	TST data8,X	IDX	2	$6D	7							[data8 + [X]] - 0	Test the Memory Location

	TST addr16	EXT	3	$7D	6							[addr16] - 0	
TSX	TSX	INH	1	$30	4	-	-	-	-	-	-	[X] ← [SP] + 1	Move Stack Pointer contents to Index register and increment.
TXS	TXS	INH	1	$35	4	-	-	-	-	-	-	[SP] ← [X] - 1	Move Index register contents to Stack Pointer and decrement.
WAI	WAI	INH	1	$3E	9	-	-	-	-	-	1	[[SP]] ← [PC(LO)], [[SP] - 1] ← [PC(HI)], [[SP] - 2] ← [X(LO)], [[SP] - 3] ← [X(HI)], [[SP] - 4] ← [A], [[SP] - 5] ← [B], [[SP] - 6] ← [SR], [SP] ← [SP] - 7	Push registers onto Stack, decrement Stack Pointer, end wiat for interrupt. If [I] = 1 when WAI is executed, a non-maskable interrupt is required to exit the Wait state. Otherwise, [I] ← 1 when the interrupt occurs.