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

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 (IMM)	ASPI (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 (DIR)	RTI (INH)	FPSH X (IDX)	FPSH (EXT)	WAI (INH)	SWI (INH)
4_	NEG A (ACC)	FPSH (IMM*)	FPUL (DIR)	COM A (ACC)	LSR A (ACC)	FPUL (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 (DIR)	INC (EXT)	TST (EXT)	JMP (EXT)	CLR (EXT)
8_	SUB A (IMM)	CMP A (IMM)	SBC A (IMM)	STAG (IDX)	AND A (IMM)	BIT A (IMM)	LDA A (IMM)	ADDG (DIR)	EOR A (IMM)	ADC A (IMM)	ORA A (IMM)	ADD A (IMM)	CPX A (IMM)	BSR (REL)	LDS (IMM)	ADDG (IDX)
9_	SUB A (DIR)	CMP A (DIR)	SBC A (DIR)	SUBD (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 (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 (prefix)	EOR B (IMM)	ADC B (IMM)	ORA B (IMM)	ADD B (IMM)	ADAX (INH)	WADGX (INH)	LDX (IMM)	
D_	SUB B (DIR)	CMP B (DIR)	SBC B (DIR)	LDAG (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 (EXT)	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 (prefix)	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)					

Yellow highlighted red <u>6800 opcodes</u> have slightly different behavior in 4052/4054 than 6800

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 = AL (6800 compatible)
  - **Extended to 16-bits AE = AH and AL**
- **ACCG is 16-bit extension of A where A is low order 8-bits**
- **ACCB** Accumulator B=BL (6800 compatible)
  - **Extended to 16-bits BE = BH and BL**
- **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

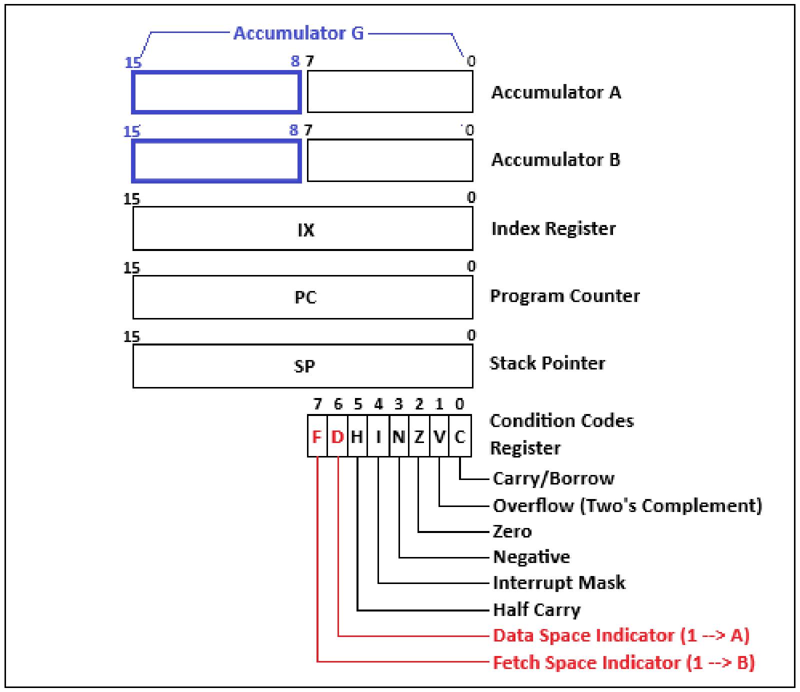

Figure 1- 4052/4054 and 4052A/4054A Registers

Opcodes added for the 4052/4054 and 4052A/4054A

ADAX	Add A to Index Register
ADXI	Add to Index Register Immediate
ASPI	Add to Stack Pointer Immediate
CBUG	Clear Debug Interrupt Vectors
CPCH	Call Code in Patch Space
CPX	Compare Index Register
FADD	Floating Point Add
FDIV	Floating Point Divide
FDUP	Duplicate Floating Point
FMUL	Floating Point Multiply
FNRM	Normalize Floating Point
FPSH	Push Floating Point
FPUL	Pull Floating Point
FSUB	Floating Point Subtract
FSWP	Swap Floating Point
JMPAX	Jump Double-Indexed
JMPIN	Jump Indirect
LDAX	Load A Register Double-Indexed
LDBX	Load B Register. Double-Indexed
LDXX	Load X Register Double-Indexed
MOVLR	Block Move Low to High
MOVRL	Block Move Low to High
NEG	Negate (2's complement)
PCH	Jump to Code in Patch Space
PSHRET	Push Return Address on Special Stack
PSHX	Push X on the Stack
PULX	Pull X from the Stack
RTRN	Return Via the Special Stack
SBUG	Set Debug Interrupt Vectors
SDA	Set Data Space to A
SDB	Set Data Space to B
SFA	Set Fetch Space to A
STAX	Store B Register Double-Indexed
STRK	Compute Stroke (4054 & 4054A ONLY)
TAP	A --> CC Not including Space Bits
TAPX	A --> CC Including Space Bits
TEST	Microcode Restart
TPA	CC --> A Not including Space Bits
TPAX	CC --> A Including Space Bits
VECT	Compute Vector (4054 & 4054A ONLY)
WADX	Add Memory to Index

6800 instructions

Opcodes added to 4052A/4054A ONLY

ADDG	Add to G Accumulator
BUFIN	Read a buffer from the GPIB
BUFOUT	Write a buffer to the GPIB
CLRGH	Clear High Byte of G
CMPGX	Compare G and X
CMPSYM	Compare Name in a Symbol Table Record
DEVIN	Read a buffer from an I/O Device
DEVOUT	Write a buffer to an I/O Device
FIXRND	Round a Float to an Integer
IFLOAT	Convert an Integer to a Float
INXSTX	Increment Index Register and Store It
LDAG	Load G Accumulator
LDAGX	Load G Accumulator Double-Indexed
PSHG	Push G on the Stack
PULG	Pull G from the Stack
SEABNK	Search for a CALL name in a ROM bank
STAG	Store G Accumulator
STAGX	Store G Accumulator Double-Indexed
SUBG	Subtract from G Accumulator
TGX	Transfer G to the Index Register
TMULT	Multiply a 6-byte Integer by 10
TXG	Transfer the Index Register to G
WADGX	Add G to Index Extended

16-bit Register AE and BE Extensions to 6800 instructions for 4052A/4054A ONLY

ABA	Add 16-bit BE to 16-bit AE
ADD	Add 8-bit value to AE or BE
ASL	Arithmetic Shift Left AE or BE
CLR	Clear AE or BE
COM	Complement AE or BE
DEC	Decrement AE or BE
INC	Increment AE or BE
LDA	Load AE or BE from Memory
PUL	Pull Data from Stack to AE | BE
RTI	Return from Interrupt
SBA	Subtract BE from AE
SUB	Subtract Memory from AE | BE
SWI	Software Interrupt
TAB	Transfer AE to BE
TBA	Transfer BE to AE
WAI	Wait for Interrupt

ABA	ADD B to A
ADC	ADD Memory contents + Carry to Accumulator
ADD	ADD Memory contents to Accumulator
AND	Memory contents AND the Accumulator to the Accumulator
ASL	Arithmetic Shift Left. Bit 0 set 0 (multiplying by two)
ASR	Arithmetic Shift Right. Bit 7 stays the same
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal to zero
BGE	Branch if Greater or Equal to zero
BGT	Branch if Greater than zero
BHI	Branch if Accumulator contents higher than comparand
BIT	Memory contents AND the Accumulator, only Status is affected
BLE	Branch if Less than or Equal zero
BLS	Branch if Accumulator contents less than or same as comparand
BLT	Branch if Less Than zero
BMI	Branch if Minus
BNE	Branch if Not Equal zero
BPL	Branch if Plus
BRA	Unconditional branch relative to present Program Counter contents
BSR	Unconditional branch to Subroutine located relative to PC contents
BVC	Branch if overflow clear
BVS	Branch if overflow set
CBA	Compare A AND B. Only status is affected
CLC	Clear the Carry flag
CLI	Clear the Interrupt flag to enable Interrupts
CLR	Clear ACC, Memory or Overflow
CLV	Clear overflow flag
CMP	Compare Memory contents AND Accumulator. Only Status affected
COM	Complement ACC or Memory
CPX	Compare Memory contents to X
DAA	Decimal Adjust Accumulator A
DEC	Decrement Accumulator or Memory
DES	Decrement Stack Pointer
DEX	Decrement Index register X
EOR	Memory Exclusive OR Accumulator
INC	Increment Accumulator or Memory
INS	Increment the Stack Pointer
INX	Increment the Index Register X
JMP	Jump
JSR	Jump to Subroutine
LDA	Load Accumulator from Memory
LDS	Load the Stack Pointer
LDX	Load the Index Register X
LSR	Logical Shift Right - Bit7 set to zero.(dividing by two)
NEG	NEGATE the Accumulator or Memory
NOP	No operation
ORA	OR the Accumulator
PSH	Push Accumulator onto the Stack
PUL	Pull Data from Stack to Accumulator
ROL	Rotate Left through Carry
ROR	Rotate Right through Carry
RTI	Return from Interrupt
RTS	Return from Subroutine
SBA	Subtract B from A
SBC	Subtract Memory and Carry flag from Accumulator
SEC	Set the Carry flag
SEI	Set the Interrupt flag
SEV	Set the Overflow flag
STA	Store Accumulator in Memory
STS	Store Stack Pointer
STX	Store Index Register X
SUB	SUBTRACT Memory contents from Accumulator
SWI	Software Interrupt
TAB	Transfer A to B
TAP	Transfer A to Status Register
TBA	Transfer B to A
TPA	Transfer Status Register to A
TST	Test the Accumulator
TSX	Move Stack Pointer to X and INC
TXS	Move X to Stack Pointer and DEC
WAI	Wait for Interrupt

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] For 4052A & 4054A: [AE] LDAXL [AE] + [BE]	Add B to A Condition Codes based on low byte of A-same as 6800
ADAX	ADAX	INH	1	$CC	?							For 4052/4054 & A: [X] ← [A] + [X] `Condition Codes:` `H I N Z V C` `. . 1 0 x x New X<0` `. . 0 1 x x New X=0` `. . x x 1 x 2's comp overflow in addition` `. . x x x 1 bit15 carry out in addition`	Add A to Index Register Unsigned value in A (eight assumed bits of 0) is added to the index register
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 A #data8	IMM	2	$8B	2							[A] ← [A] + data8 For 4052A & 4054A: [AL] ← [AL] + data8 [AH] ← [AH] + C	
	ADD A addr8	DIR	2	$9B	3							[A] ← [A] + [addr8] For 4052A & 4054A: [AL] ← [AL] + [addr8] [AH] ← [AH] + C	
ADD	ADD A data8,X	IDX	2	$AB	5	x	x	x	x	x	-	[A] ← [A] + [data8 + [X]] For 4052A & 4054A: [AL] ← [AL] + [X] [AH] ← Trash bits	Add Memory contents to the Accumulator Condition Codes based on low byte of A - same as 6800
	ADD A addr16	EXT	3	$BB	4							[A] ← [A] + [addr16] For 4052A & 4054A: [AL] ← [AL] + [addr16] [AH] ← [AH] + C	
	ADD B #data8	IMM	2	$CB	2							[B] ← [B] + data8 For 4052A & 4054A: [BL] ← [BL] + [data8] [BH] ← [BH] + C	

	ADD B addr8	DIR	2	$DB	3							[B] ← [B] + [addr8] For 4052A & 4054A: [BL] ← [BL] + [addr8] [BH] ← [BH] + C	
	ADD B data8,X	IDX	2	$EB	5							[B] ← [B] + [data8 + [X]] For 4052A & 4054A: [BL] ← [BL] + [data8 + [X]} [BH] ← Trashed bits	
	ADD B addr16	EXT	3	$FB	4							[B] ← [B] + [addr16] For 4052A & 4054A: [BL] ← [BL] + [addr16] [BH] ← [BH] + C	
ADXI	ADD X data8	IMM	2	$14	?							For 4052/4054 & A: [X] ← [X]+data8 CC unaffected	Add to Index Register IMM (signed two's complement operand)
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	-	-	C ← 76543210 ← 0 For 4052A & 4054A: C ← 16-bit ACCX ← 0	Arithmetic Shift Left. Bit 0 is set to 0. (multiplying by two) Condition Codes based on low byte - same as 6800
	ASL B	ACC	1	$58	2								
	ASL data8,X	IDX	2	$68	7								
	ASL addr16	EXT	3	$78	6								
ASPI	ASPI data8	IMM	2	$15	?							For 4052/4054 & A: [SP] ← [SP]+data8 CC unaffected	Add to Stack Pointer Immediate The signed 2's complement operand is added to the value currently in the stack pointer
ASR	ASR A	ACC	1	$47	2	x	x	x	x	-	-		

	ASR B	ACC	1	$57	2							$\boxed{76543210}$ → C	Arithmetic Shift Right. Bit 7 stays the same.
	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.
CBUG	CBUG	INH	1	$DD	?							For 4052/4054 & A: CC unaffected	Clear debug interrupt vectors This is the complement of the SBUG instruction. Interrupt vectors are restored to their normal area in A-Space
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 For 4052A & 4054A: [AE]← 0	Clear the Accumulator Condition Codes based on low byte - same as 6800
	CLR B	ACC	1	$5F	2							[B] ← 0 For 4052A & 4054A: [BE] ← 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] For 4052A & 4054A: [AE] ← $FFFF - [AE]	Complement the Accumulator Condition Codes based on low byte - same as 6800
	COM B	ACC	1	$53	2							[B] ← $FF - [B] For 4052A & 4054A: [BE] ← $FFFF - [BE]	
	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]	
CPCH	CPCH data8	IMM	2	$F3	?							For 4052/4054 & A: [PC] ← [PC]+2 V [PCL] [SP] ←[SP}-1 [PC] ←Rel*4+4400 CC unaffected	Call code in Patch space Second byte specifies offset (times 4) into patch area to be executed instead of original code. Return address following the patch is pushed on stack before patch code is executed
CPX	CPX addr8	DIR	2	$9C	?	-	x	x	x	-	-	4052/4054 & A: [X]-[M,M+1] 16-bit in all modes `Condition Codes:` `H I N Z V C` `. . 1 0 . .` `X arithmetically < M` `. . 0 1 . .  X = M` `. . 0 0 . .` `X arithmetically > M` `. . . . 1 . 2's Complement overflow in compare` `. . x 0 . 1` `X logically < M` `. . x 0 . 0` `X logically > M` [X(HI)] - [addr8], [X(LO)] - [addr8 + 1]	Compare Index Register [X]-[M,M+1] full 16-bit Current contents of X are compared to 16-bit operand. 2's complement subtract is used to set [CC], reflecting a valid 16-bit compare. Makes 6800 instruction more useful by correctly setting C bit (like 6800 does for CMP)

	CPX data8,X	IDX	2	$AC	?							[X(HI)] - [data8 + [X]], [X(LO)] - [data8 + [X] + 1]	
	CPX #data16	IMM	3	$8C	?							[X(HI)] - data16(HI), [X(LO)] - data16(LO)	
	CPX addr16	EXT	3	$BC	?							[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 For 4052A & 4054A: [AE] ← [AE] – 1	Decrement the Accumulator Condition Codes based on low byte - same as 6800
	DEC B	ACC	1	$5A	2							[B] ← [B] – 1 For 4052A & 4054A: [BE] ← [BE] – 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]	

FADD	FADD	INH	1	$52	?							For 4052/4054 & A: [SP] ← [SP]+9 [M([SP]+2..[SP]+9)] + [M([SP]-7..[SP]) ← M([SP]+2..[SP+9) **Condition Codes:** `H I N Z V C` `. . 1 0 x x Result Negativ` `. . 0 1 x 0 Result Zero` `. . 0 0 x x Result Positiv` `. . 0 1 1 0 Underflow – zero result` `. . x 0 1 0 Overflow – plus/minus infinity result`	Floating Point Add Floating point add on two top 8-byte FP numbers on the stack. Result is left on stack Programming notes: Acc A crashed Acc B is number of shifts used to normalize result Example: “6.0+7.0”, first push 6.0, then 7.0, then do the Add
FDIV	FDIV	INH	1	$5E	?							For 4052/4054 & A: [SP] ← [SP]+9 [M([SP]+2..[SP]+9)] / [M([SP]-7..[SP]) ← M([SP]+2..[SP+9) **Condition Codes:** `H I N Z V C` `. . 1 0 x x Result Negativ` `. . 0 1 x 0 Result Zero` `. . 0 0 x x Result Positiv` `. . 0 1 1 0 Underflow – zero result` `. . x 0 1 0 Overflow – plus/minus infinity result` `. . x 0 1 1 Divide by zero (causes interrupt)`	Floating Point Divide Floating point divide on two top 8-byte FP numbers on the stack. Result is left on stack Programming notes: Acc A crashed Acc B is number of shifts used to normalize result Example: “6.0+7.0”, first push 6.0, then 7.0, then do the Divide

FDUP	FDUP	INH	1	$4E	?							For 4052/4054 & A: [M([SP]..[SP]+9)] ← [M([SP]-9..[SP]) [SP] ← [SP]-9 [CC] unaffected	Duplicate Floating Point Floating point number on the top of stack is replicated on the stack Programming notes: To save microcode time, the byte below that pointed to by the SP is duplicated also, making a total of 10 bytes duplicated
FMUL	FMUL	INH	1	$5B	?							For 4052/4054 & A: [SP] ← [SP]+9 [M([SP]+2..[SP]+9)] * [M([SP]-7..[SP]) ← M([SP]+2..[SP+9) `Condition Codes:` `H I N Z V C` `. . 1 0 x x Result Negativ` `. . 0 1 x 0 Result Zero` `. . 0 0 x x Result Positiv` `. . 0 1 1 0 Underflow – zero result` `. . x 0 1 0 Overflow – plus/minus infinity result`	Floating Point Multiply Floating point multiply on two top 8-byte FP numbers on the stack. Result is left on stack Programming notes: Acc A crashed Acc B is number of shifts used to normalize result Example: “6.0+7.0”, first push 6.0, then 7.0, then do the Multiply
FNRM	FNRM	INH	1	$61	?							For 4052/4054 & A: Normalize [M([SP]+2..[SP]+9)] ← M([SP]+2..[SP]+9) `Condition Codes:`	Normalize Floating Point Floating point number on the top of the stack is normalized. Programming notes: Acc A crashed

												H I N Z V C . . 1 0 0 0 Result Negativ . . 0 1 x 0 Result Zero . . 0 0 0 0 Result Positiv . . 0 1 1 0 Underflow – zero result, FP Interrupt	Acc B is number of shifts used to normalize result
FPSH	FPSH addr8	DIR	2	$3A	?							For 4052A & 4054A: Operation: V M[EA+7] V M[EA+6] V M[EA+5] V M[EA+4] V M[EA+3] V M[EA+2] V M[EA+1] V M[EA+0] V Valtg [CC] unaffected	Push Floating Point Floating point number specified by the operand is pushed on the stack along with a floating point tag. Programming note: A deferred fetch-space change will occur after an FPSH Immediate instruction.
	FPSH data8,X	IDX	2	$3C	?								
	FPSH addr16	EXT	3	$3D	?								
	FPSH #^H<16-digit Hex value>	IMM	2	$41	?								
FPUL	FPUL addr8	DIR	2	$42	?							For 4052/4054 & A: [SP] ← [SP]+9 M..M+7← [M([SP]-7..[SP]) [CC] unaffected	Pull Floating Point Floating point number specified by operand is pulled from the stack. The floating point tag is discarded
	FPUL data8,X	IDX	2	$45	?								
	FPUL addr16	EXT	3	$4B	?								

FSUB	FSUB	INH	1	$55	?							For 4052/4054 & A: [SP] ← [SP]+9 [M([SP]+2..[SP]+9)] - [M([SP]-7..[SP]) ← M([SP]+2..[SP+9) Condition Codes: H I N Z V C . . 1 0 x x Result Negativ . . 0 1 x 0 Result Zero . . 0 0 x x Result Positiv . . 0 1 1 0 Underflow – zero result. . x 0 1 0 Overflow – plus/minus infinity result	Floating Point Subtract Floating point subtract on two top 8-byte FP numbers on the stack. Result is left on stack Programming notes: Acc A crashed Acc B is number of shifts used to normalize result Example: “6.0+7.0”, first push 6.0, then 7.0, then do the subtract
FSWP	FSWP	INH	1	$51	?							For 4052/4054 & A: [M([SP]..[SP]+9)] swapped with [M([SP]-9..[SP]) [CC] unaffected	Swap Floating Point The top two floating point numbers on the stack are interchanged Programming notes: Only the eight-byte FP actual values are swapped. The tags are not swapped.
INC	INC A	ACC	1	$4C	2	-	x	x	x	-	-	[A] ← [A] + 1 For 4052A & 4054A: [AE] ← [AE] + 1	Increment the Accumulator Condition Codes based on low byte - same as 6800

	INC B	ACC	1	$5C	2							[B] ← [B] + 1 For 4052A & 4054A: [BE] ← [BE] + 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	
JMPAX	JMPAX	INH	1	$1F	?							For 4052/4054 & A: [PC] ← [A]+[X] [CC] unaffected	Jump Double-Indexed The next instruction to be executed is to be found at X+A
JMPIN	JMPIN addr16	EXT	3	$38	?							For 4052/4054 & A: [PC] ← [M,M+1] [CC] unaffected	Jump Indirect The next instruction to be executed is to be found at the 16-bit address POINTED to by the 16-bit Operand

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 For 4052A & 4054A: [AL] ← data8 [AH] ←0	Load Accumulator from Memory Condition Codes based on low byte - same as 6800
	LDA A addr8	DIR	2	$96	3							[A] ← [addr8] For 4052A & 4054A: [AL] ← [addr8] [AH] ←0	
	LDA A data8,X	IDX	2	$A6	5							[A] ← [data8 + [X]] For 4052A & 4054A: [AL] ← data8 [AH] ←Trash bits	
	LDA A addr16	EXT	3	$B6	4							[A] ← [addr16] For 4052A & 4054A: [AL] ← [addr16] [AH] ←0	

	LDA B #data8	IMM	2	$C6	2							[B] ← data8 For 4052A & 4054A: [BL] ← data8 [BH] ←0	
	LDA B addr8	DIR	2	$D6	3							[B] ← [addr8] For 4052A & 4054A: [BL] ← [addr8] [BH] ←0	
	LDA B data8,X	IDX	2	$E6	5							[B] ← [data8 + [X]] For 4052A & 4054A: [BL] ← data8 [BH] ←Trash bits	
	LDA B addr16	EXT	3	$F6	4							[B] ← [addr16] For 4052A & 4054A: [BL] ← [addr16] [BH] ←0	
LDAG	LDAG addr8	DIR	1	$04	?							For 4052A/4054A only: [G] ← [addr8]	Load A Register

												Condition Codes: H I N Z V C . . 1 0 0 . New A Negative . . 0 1 0 . New A Zero . . 0 0 0 . New A Positive	The word at M is loaded into register G (16-bit A)
	LDAG data8,X	IDX	1	$05	?							For 4052A/4054A only: [G] ←[data8 + [X]] Condition Codes: H I N Z V C . . 1 0 0 . New A Negative . . 0 1 0 . New A Zero . . 0 0 0 . New A Positive	
	LDAG addr16	EXT	1	$B3	?							For 4052A/4054A only: [G] ← [addr16] Condition Codes: H I N Z V C . . 1 0 0 . New A Negative . . 0 1 0 . New A Zero . . 0 0 0 . New A Positive	

	LDAG #data16	IMM	1	$D3	?							For 4052A/4054A only: [G] ← [#data16] Condition Codes: H I N Z V C . . 1 0 0 . New A Negative . . 0 1 0 . New A Zero . . 0 0 0 . New A Positive	
LDAGX	LDAGX	INH	1	$FC08	?							For 4052A/4054A only: [G] ← [X+B,X+B+1] Condition Codes: H I N Z V C . . 1 0 0 . New A Negative . . 0 1 0 . New A Zero . . 0 0 0 . New A Positive	Load G Register Double-Indexed The word at X+B is loaded into register G (16-bit A)
LDAX	LDAX	INH	1	$1C	?							For 4052/4054 & A: [A] ← [X]+[A] Condition Codes: H I N Z V C . . 1 0 0 . New A Negative . . 0 1 0 . New A Zero . . 0 0 0 . New A Positive	Load A Register Double-Indexed The byte at X+A is loaded into register A

LDBX	LDBX	INH	1	$1D	?							For 4052/4054 & A: [B] ← [X]+[A] Condition Codes: H I N Z V C . . 1 0 0 . New B Negative . . 0 1 0 . New B Zero . . 0 0 0 . New B Positive	Load B Register Double-Indexed The byte at X+A is loaded into register B
LDXX	LDXX	INH	1	$1A	?							For 4052/4054 & A: [X] ← [X+A,X+A+1] Condition Codes: H I N Z V C . . 1 0 0 . New X Negative . . 0 1 0 . New X Zero . . 0 0 0 . New X Positive	Load X Register Double-Indexed The 16-bit value at X+A is loaded into register A
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 → 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								
MOVLR	MOVLR	INH	1	$E3	?							For 4052/4054 & A: M[SP+1],M[SP+2] is lowest source address M[SP+3],M[SP+4] is lowest destination address M[SP+5],M[SP+6] is byte count (may be zero) Data moved: [SP] ←[SP]+6	Block Move Low to High A block of data in memory is moved, incrementing the pointers, until the byte count is zero

												Condition codes - crashed	
MOVLR	MOVLR	INH	1	$EC	?							For 4052/4054 & A: M[SP+1],M[SP+2] is highest source address M[SP+3],M[SP+4] is highest destination address M[SP+5],M[SP+6] is byte count (may be zero) Data moved: [SP] ←[SP]+6 Condition codes - crashed	Block Move High to Low A block of data in memory is moved, decrementing the pointers, until the byte count is zero
NEG	NEG A	ACC	1	$40	2	x	x	x	x	-	-	[A] ← 0 - [A]	Negate Accumulator (2's complement) ///// WARNING \\\\\\ The 4052/4054 set the CARRY bit exactly opposite of how it is set in the 6800 and 4052A/4054A!! See 6800 manual
	NEG B	ACC	1	$50	2							[B] ← 0 - [B]	
	NEG data8,X	IDX	2	$60	7							[data8 + [X]] ← 0 - [data8 + [X]]	Negate Memory Location (2's complement) ///// WARNING \\\\\\
	NEG addr16	EXT	3	$70	6							[addr16] ← 0 - [addr16]	

													The 4052/4054 set the CARRY bit exactly opposite of how it is set in the 6800 and 4052A/4054A!! See 6800 manual
NOP	NOP	INH	1	$01	2	-	-	-	-	-	-		No Operation
ORA	ORA A #data8	IMM	2	$8A	2	-	x	x	0	-	-	[A] ← [A] v data8	OR the Accumulator
	ORA A addr8	DIR	2	$9A	3							[A] ← [A] v [addr8]	
	ORA A data8,X	IDX	2	$AA	5							[A] ← [A] v [data8 + [X]]	
	ORA A addr16	EXT	3	$BA	4							[A] ← [A] v [addr16]	
	ORA B #data8	IMM	2	$CA	2							[B] ← [B] v data8	
	ORA B addr8	DIR	2	$DA	3							[B] ← [B] v [addr8]	
	ORA B data8,X	IDX	2	$EA	5							[B] ← [B] v [data8 + [X]]	
	ORA B addr16	EXT	3	$FA	4							[B] ← [B] v [addr16]	
PCH	PCH data8	IMM	2	$FD	?							For 4052/4054 & A: [PC] ←Rel*4+4400	Jump to code in Patch space This instruction forces a JUMP to code in the patch

												CC unaffected	space. The code begins at the second byte times 4 plus 4400 hex.
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	
PSHRET	PSHRET data8	DIR	2	$62	?							For 4052/4054 & A: M([SP]+1),M([SP]+2) ← M([Psp]),M([Psp]+1) [SP] ←[SP]+2 [Psp]←[Psp]+2 [X] ←[SP]+1 CC unaffected	Push return address on special stack The return address on the regular stack is transferred to the pseudo stack referenced by the specified page zero pointer. Programming Note: Notice the implicit TSX at the end of the instruction!!
PSHX	PSHX data8	INH	2	$6B	?							For 4052/4054 & A: [SP]←[SP]-2 [X] ←M([SP]+1),M([SP]+2) CC unaffected	Push X on the stack The index register is pushed on the stack
PUL	PUL A	ACC	1	$32	4	-	-	-	-	-	-	[SP] ← [SP] + 1, [A] ← [[SP]] For 4052A & 4054A: [AL] ← [[SP+1]], [SP] ← [SP] + 1	Pull Data from Stack to Accumulator

												[AH] ← Trash bits	Condition Codes based on low byte - same as 6800
	PUL B	ACC	1	$33	4							[SP] ← [SP] + 1, [B] ← [[SP]] For 4052A & 4054A: [BL] ← [[SP+1]], [SP] ← [SP] + 1 [BH] ← Trash bits	
PULX	PULX data8	INH	2	$75	?							For 4052/4054 & A: [X] ←M([SP]+1),M([SP]+2) [SP]←[SP]+2 CC unaffected	Pull X from the stack The index register is pulled from the stack
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 For 4052A & 4054A: [SR] ← [[SP] + 1], [BL] ← [[SP] + 2], [AL] ← [[SP] + 3], [X(HI)] ← [[SP] + 4], [X(LO)] ← [[SP] + 5], [PC(HI)] ← [[SP] + 6], [PC(LO)] ← [[SP] + 7], [BH] ← [[SP] + 8], [BL] ← [[SP] + 9], (ignored) [AH] ← [[SP] + 10], (G reg) [AL] ← [[SP] + 11], (ignored) [SP] ← [SP] + 11	Return from interrupt. Put registers from Stack and increment Stack Pointer. For 4052A & 4054A: RTI pops 11 bytes (6800 popped only 7) to restore the hardware registers to the state they were before an interrupt occurred (or SWI [ODT only] or WAI [not used in 4052 or 4054]). When the interrupt occurred, Status Register CC was pushed onto the stack and then the D and F bits in CC were set to 1 (1 --> Fetch B and Data A).
RTRN	RTRN	DIR	2	$65	?							For 4052/4054 & A: [Psp]←[Psp]-2 [B]←[A] [PC]←M([Psp]),M([Psp]+1) CC See TAB instruction	Return via the special stack Fetch the return address from the pseudo stack with stack pointer on page zero. Programming Note: An implicit TAB instruction is done!!!

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] For 4052A & 4054A: [AE] ← [AE] - [BE]	Subtract contents of Accumulator B from those of Accumulator A. Condition Codes based on low byte of A only - same as 6800
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	
SBUG	SBUG	INH	1	$DC	?							For 4052/4054 & A:	Set debug interrupt vectors

												CC unaffected	Swap in debug interrupt vectors. Programming Note: Subsequent interrupts will be serviced via vectors in B-Space from locations 2 to F in the bank with address 20. This supports the 4052 Diagnostic ROM Pack.
SDA	SDA	INH	1	$18	?							For 4052/4054 & A: [CC] ← [CC] ! D [CC] D set	Set Data Space to A Subsequent memory accesses for data will access A space
SDB	SDB	INH	1	$21	?							For 4052/4054 & A: [CC] ← [CC] & NOT D [CC] D reset	Set Data Space to B Subsequent memory accesses for data will access B space
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

SFA	SFA	INH	1	$03	?							For 4052/4054 & A: [CC] ← [CC] & NOT F [CC] F reset	Set Fetch Space to A Instructions subsequent to next JSR, RTS, BSR, JMP, BRA, relative branch, RTRN, JMPAX, RTI or FPSH immediate instruction will come from DATA space
STA	STA A addr8	DIR	2	$97	4	-	x	x	0	-	-	[addr8] ← [A]	Store Accumulator in Memory
	STA A data8,X	IDX	2	$A7	6							[data8 + [X]] ← [A]	
	STA A addr16	EXT	3	$B7	5							[addr16] ← [A]	
	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]	
STAX	STAX	INH	1	$1E	?							For 4052/4054 & A: [X]+[A] ← [B] `Condition Codes:` `H I N Z V C` `. . 1 0 0 . New A Negative` `. . 0 1 0 . New A Zero` `. . 0 0 0 . New A Positive`	Store B Register Double-Indexed The byte at X+A is loaded from B register

STRK	STRK	INH	1	$71	?							For 4054 & 4054A ONLY: See 4052 Assembler pg 44 for the pseudo code. **Condition Codes:** Set to the state of the A register	Compute Stroke Given a stroke from the character stroke table in A and scale code in B, this instruction computes desired X and Y for the stroke. If the stroke has the negative bit set, the vector-drawing information needed by the 4054 display is pushed onto the stack as in VECT.
STS	STS addr8	DIR	2	$9F	5	-	x	x	0	-	-	[addr8] ← [SP(HI)], [addr8 + 1] ← [SP(LO)]	Store the Stack Pointer
	STS data8,X	IDX	2	$AF	7							[data8 + [X]] ← [SP(HI)], [data8 + [X] + 1] ← [SP(LO)]	
	STS addr16	EXT	3	$BF	6							[addr16(HI)] ← [SP(HI)], [addr16(LO)] ← [SP(LO)]	
STX	STX addr8	DIR	2	$DF	5	-	x	x	0	-	-	[addr8] ← [X(HI)], [addr8 + 1] ← [X(LO)]	Store the Index Register X
	STX data8,X	IDX	2	$EF	7							[data8 + [X]] ← [X(HI)], [data8 + [X] + 1] ← [X(LO)]	
	STX addr16	EXT	3	$FF	6							[addr16(HI)] ← [X(HI)], [addr16(LO)] ← [X(LO)]	

SUB	SUB A #data8	IMM	2	$80	2	x	x	x	x	-	-	[A] ← [A] - data8 For 4052A & 4054A: [AE] ← [AE] - data8	Subtract Memory contents from Accumulator Condition Codes based on low byte of A or B only - same as 6800
	SUB A addr8	DIR	2	$90	3							[A] ← [A] - [addr8] For 4052A & 4054A: [AE] ← [AE] – [addr8]	
	SUB A data8,X	IDX	2	$A0	5							[A] ← [A] - [data8 + [X]] For 4052A & 4054A: [AL] ← [AL] – [data8 + [X]] [AH] ← Trash bits	
	SUB A addr16	EXT	3	$B0	4							[A] ← [A] - [addr16] For 4052A & 4054A: [AE] ← [AE] – [addr16]	
	SUB B #data8	IMM	2	$C0	2							[B] ← [B] - data8 For 4052A & 4054A: [BE] ← [BE] - data8	
	SUB B addr8	DIR	2	$D0	3							[B] ← [B] - [addr8] For 4052A & 4054A: [BE] ← [BE] – [addr8]	
	SUB B data8,X	IDX	2	$E0	5							[B] ← [B] - [data8 + [X]]	

												For 4052A & 4054A: [BL] ← [BL] – [data8 + [X]] [BH] ← Trash bits	
	SUB B addr16	EXT	3	$F0	4							[B] ← [B] - [addr16] For 4052A & 4054A: [BE] ← [BE] – [addr16]	

SWI	SWI	INH	1	$3F	12	-	-	-	-	-	1	For 4052 & 4054 like 6800: [[SP]] ← [PC(LO)], [[SP] - 1] ← [PC(HI)], [[SP] - 2] ← [X(LO)], [[SP] - 3] ← [X(HI)], [[SP] - 4] ← [A], [[SP] - 5] ← [B], [[SP] - 6] ← [SR], [SP] ← [SP] - 7, [PC(HI)] ← [$FFFA], [PC(LO)] ← [$FFFB] For 4052A & 4054A: [[SP]] ←[AL], [[SP] - 1] ←[AH], (G register) [[SP] - 2]←[BL], [[SP] - 3]←[BH], [[SP] - 4]← [PC(LO)], [[SP] - 5] ← [PC(HI)], [[SP] - 6] ← [X(LO)], [[SP] - 7] ← [X(HI)], [[SP] - 8] ← [AL], [[SP] - 9] ← [BL], [[SP] - 10] ← [SR], [SP] ← [SP] – 11	Software Interrupt: push registers onto Stack, decrement Stack Pointer, and jump to interrupt subroutine. For 4052A & 4054A: RTI pops 11 bytes (6800 popped only 7) to restore the hardware registers to the state they were before an interrupt occurred (or SWI [ODT only] or WAI [not used in 4052 or 4054]). When the interrupt occurred, Status Register CC was pushed onto the stack and then the D and F bits in CC were set to 1 (1 --> Fetch B and Data A).

TAB	TAB	INH	1	$16	2	-	x	x	0	-	-	[B] ← [A] For 4052A & 4054A: [BE] ← [AE]	Transfer A to B Condition Codes based on low byte of B only - same as 6800
TAP	TAP	INH	1	$06	2	x	x	x	x	x	x	For 4052/4054 & A: [CC] ← [A] Low 6 bits	Transfer A to CC NOT including space bits Set the CC register to contents of A, ignoring top two bits of A and leaving top two bits of CC unchanged.
TAPX	TAPX	INH	1	$12	2	x	x	x	x	x	x	For 4052/4054 & A: [CC] ← [A] all 8 bits	Transfer A to CC including space bits Set the CC register to contents of A. All bits moved. Programming Note: If the F-bit changes, instructions subsequent to next JSR, RTS, BSR, JMP, BRA, relative branch, RTRN, JMPAX, RTI or FPSH immediate instruction will come from DATA space
TBA	TBA	INH	1	$17	2	-	x	x	0	-	-	[A] ← [B] For 4052A & 4054A:	Transfer B to A

												[AE] ← [BE]	Condition Codes based on low byte of A only - same as 6800
TEST	TEST	INH	1	$00	2	-	-	-	-	-	-	For 4052/4054 & A: [A] ← [CC] low 6-bits [A] to 2 high bits ←11 CC unaffected	Microcode Restart This instruction performs a microcode restart without disturbing the hardware.
TPA	TPA	INH	1	$07	2	-	-	-	-	-	-	For 4052/4054 & A: [A] ← [CC] low 6-bits [A] to 2 high bits ←11 CC unaffected	Transfer CC Register to A – Space bits set to 11 (1 --> Fetch B and Data A). Microcode sets 2 high bits to 11 regardless of CC contents
TPAX	TPAX	INH	1	$13	2	-	-	-	-	-	-	For 4052/4054 & A: [A] ← [CC] all 8 bits moved CC unaffected	Transfer CC Register to A – including Space bits
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.

VECT	VECT	INH	1	$72	?							For 4054 & 4054A ONLY: See 4052 Assembler pg 48 for the pseudo code. `Condition Codes:` Set to the state of the A register	Compute Vector This instruction pushes onto the stack the vector-drawing information needed by the display interface of the 4054.
WADG	WADG	INH	1	$CD	?							For 4052A & 4054A ONLY: See 4052 Assembler pg 44 for the pseudo code. `Condition Codes:` Set to the state of the A register	Add G to index extended Programming note: Cannot be used with the old 4052/4054 because interrupts (maskable and non-maskable) will screw up the high byte of the G register
WADX	WADX,addr16	EXT	1	$ED	?							For 4052/4054 & A: [X] ←M([PC]+1,[PC]+2) +[X] `Condition Codes:` `H I N Z V C` `. . 1 0 x x Result Negativ` `. . 0 1 x x Result Zero` `. . x x 1 x Overflow` `. . x x x 1 Carry of bit15`	Add memory to index The sixteen-bit value in memory is added to the index register.
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 For 4052A & 4054A: [[SP]] ←[AL], [[SP] - 1] ←[AH], (G register) [[SP] - 2]←[BL], [[SP] - 3]←[BH], [[SP] - 4]← [PC(LO)], [[SP] - 5] ← [PC(HI)], [[SP] - 6] ← [X(LO)], [[SP] - 7] ← [X(HI)], [[SP] - 8] ← [AL], [[SP] - 9] ← [BL], [[SP] - 10] ← [SR], [SP] ← [SP] – 11	Push registers onto Stack, decrement Stack Pointer, end wait 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. For 4052A & 4054A: RTI pops 11 bytes (6800 popped only 7) to restore the hardware registers to the state they were before an interrupt occurred (or SWI [ODT only] or WAI [not used in 4052 or 4054]). See pg 81 When the interrupt occurred, Status Register CC was pushed onto the stack and then the D and F bits in CC were set to 1 (1 --> Fetch B and Data A).