

S12 CPU

Instruction Set

Reference Manual

38 pages (5M)

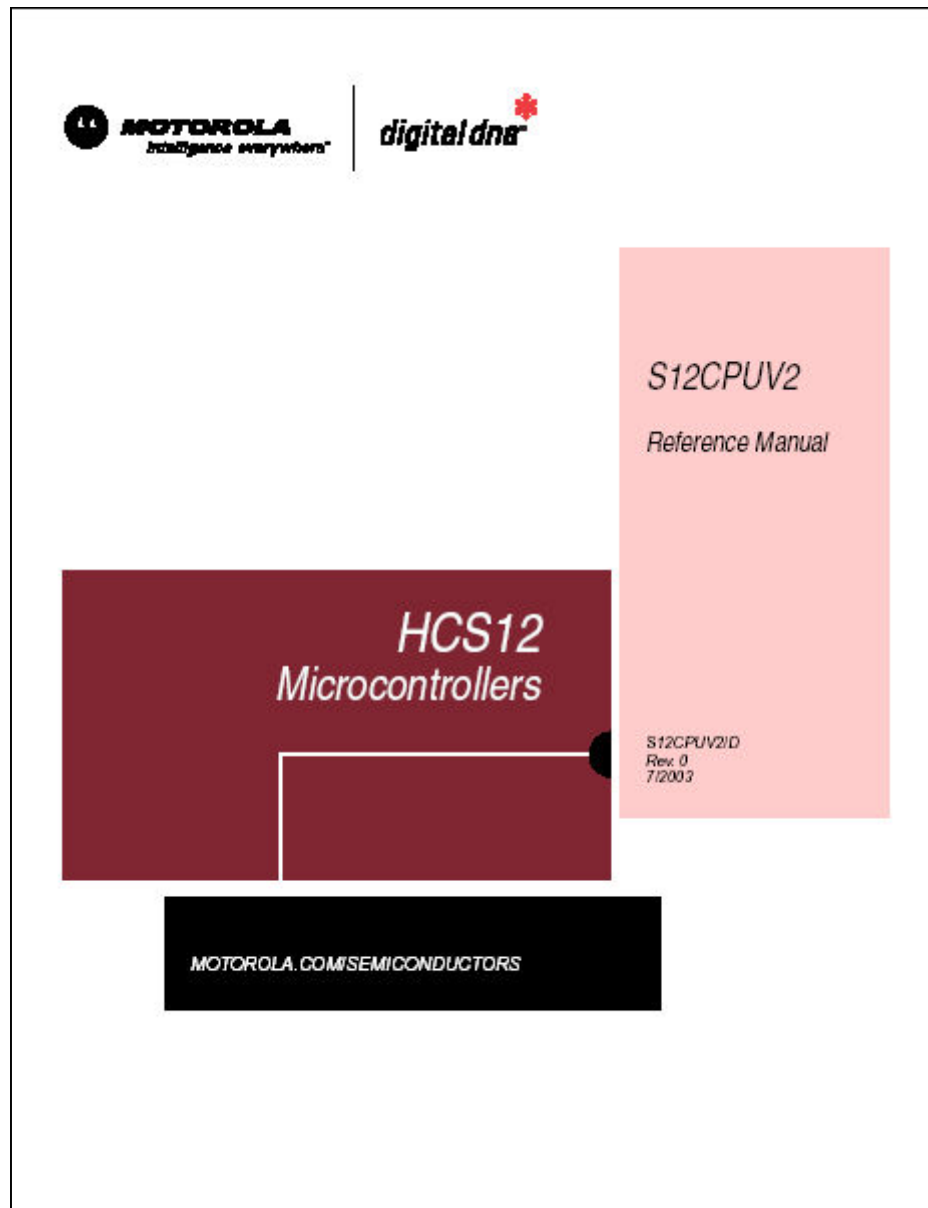
TP: 166.11.32.14



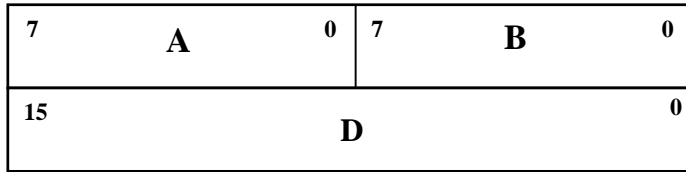
嵌入式实时系统与微控制器应用（研）



Data book

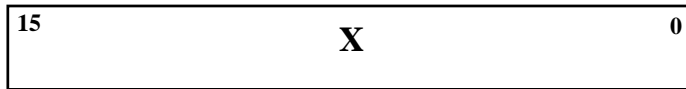


Programming Model

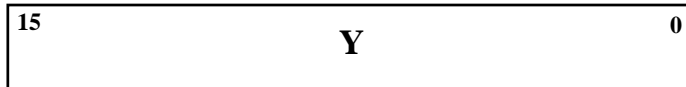


8-BIT ACCUMULATORS A AND B

OR 16-BIT DOUBLE ACCUMULATOR



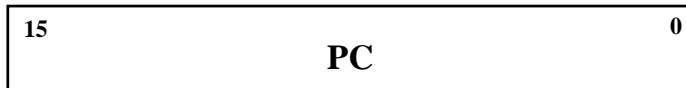
INDEX REGISTER X



INDEX REGISTER Y



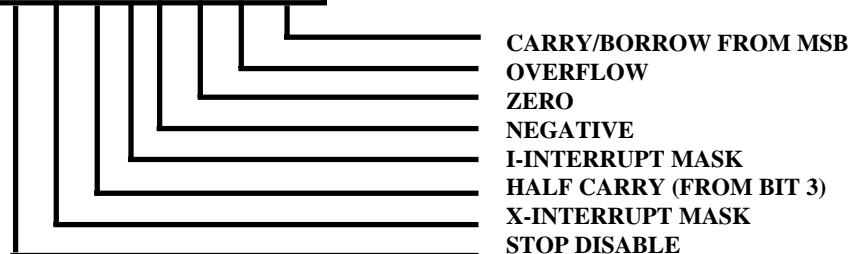
STACK POINTER



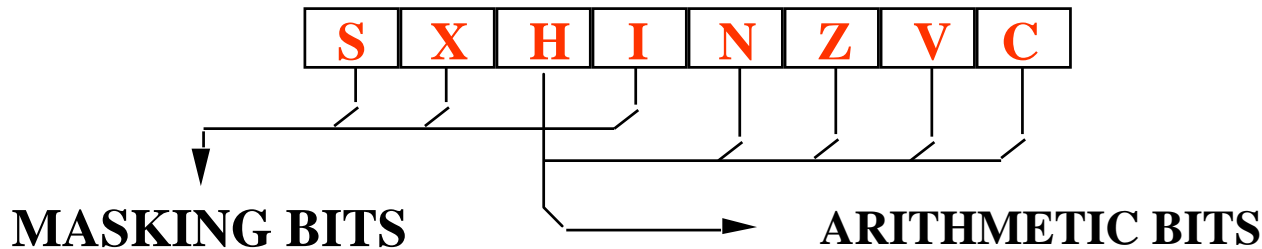
PROGRAM COUNTER



CONDITION CODE REGISTER



Condition Code Register



• **Reflect results of instruction execution.**

S - Disables **STOP** instruction when set.

X - Masks $\overline{\text{XIRQ}}$ request when set.
- set by hardware reset, cleared by software.
- set by unmaskable $\overline{\text{XIRQ}}$

I - Masks interrupt request from all $\overline{\text{IRQ}}$ level sources (both external and internal) when set.

- set by unmasked $\underline{\text{I}}$ level request
or unmasked XIRQ

C - Carry/Borrow from MSB
unsigned arithmetic

V - 2's complement overflow indicator
signed arithmetic

Z - Zero result

N - Negative (follows MS Bit of result)

H - Half Carry from bit 3 to bit 4
ADD operations only

HCS12

Addressing Modes

Addressing Modes

INHERENT

CLRB

IMMEDIATE

LDAA

#\$12

EXTENDED

LDAA

\$4000

DIRECT

LDAA

\$50

INDEXED

LDAB

30000,X

LDAA

\$8,X+

.....

RELATIVE

BNE

LOOP

IDX Indexed Address

Indexed (no extension bytes):

- **5-bit constant offset from X, Y, SP or PC**
 - LDAA 20,X
 - LDAB ,X
- **Pre/post increment/decrement by 1–8, no offset**
 - LDAA 8,X+
 - LDAB 2,-X
- **Accumulator A, B, or D offset**
 - LDAB A,X
 - LDAA [D,X] (Index Indirect)

Effective Address

Example:

LEAS	-10,S	;Allocate space for 5 x 16-bit integers
LEAS	10,S	;Deallocate space for 5 x 16-bit ints

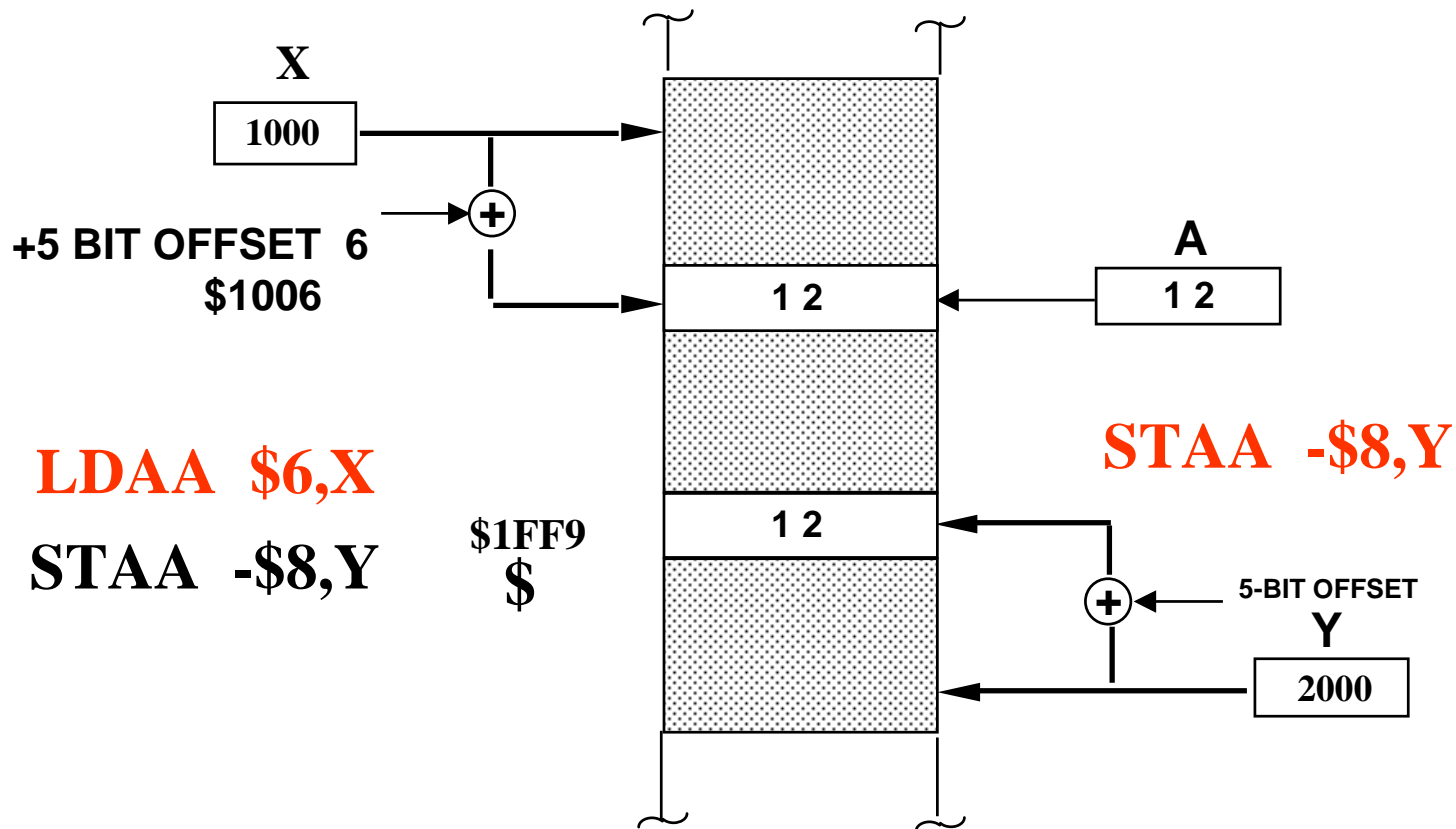
The (de)allocation can even be combined with a register push or pull as in the following example:

LDX	8,S+	;Load return value and deallocate
------------	-------------	--

HCS12 New Indexed Addressing Modes

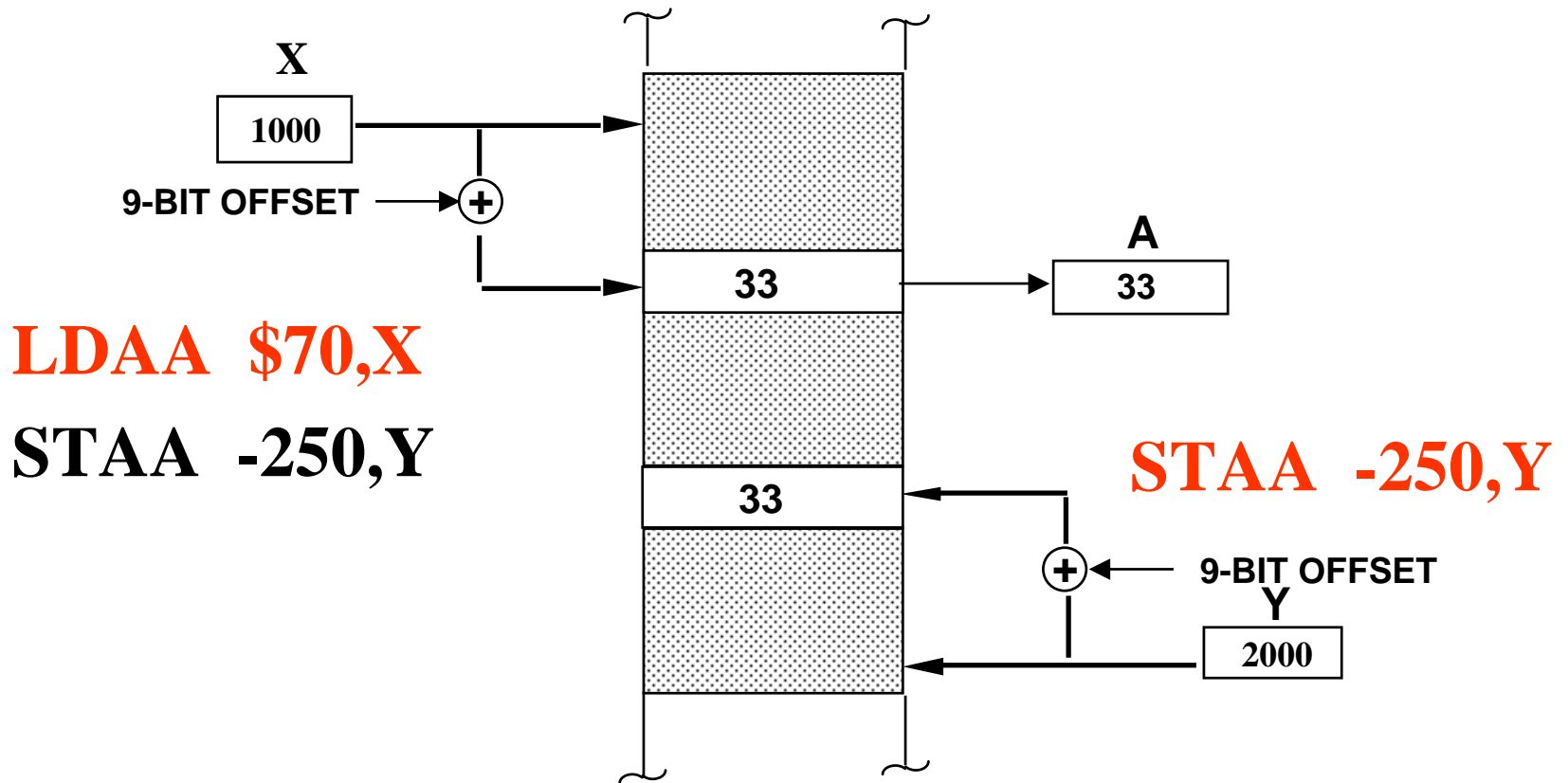
Index 5-bit signed offset	LDAA	-\$10,X
Indexed 9-bit signed offset	LDAA	-\$50,X
Indexed 16-bit signed offset	LDAA	-\$500,X
Indexed Indirect	JMP	[D,X]
	LDAA	[D,X]

Indexed 5-Bit Signed Offset



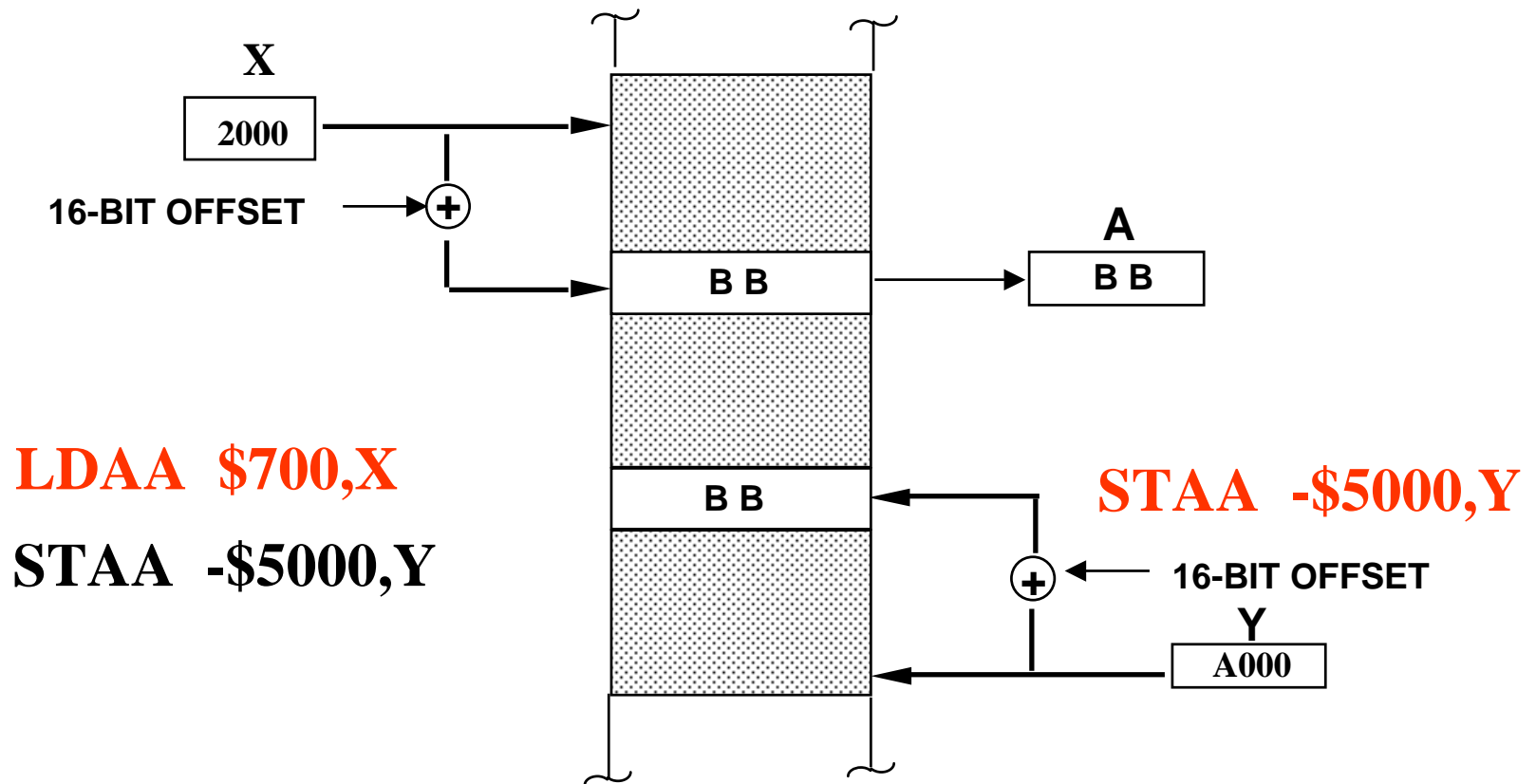
- 5 Bit offset is signed included in instruction post byte.
- X, Y, SP or PC register can be used for indexing.
- Offset range from -16 to +15 from base register.

Indexed 9-Bit Signed Offset (IDX1)



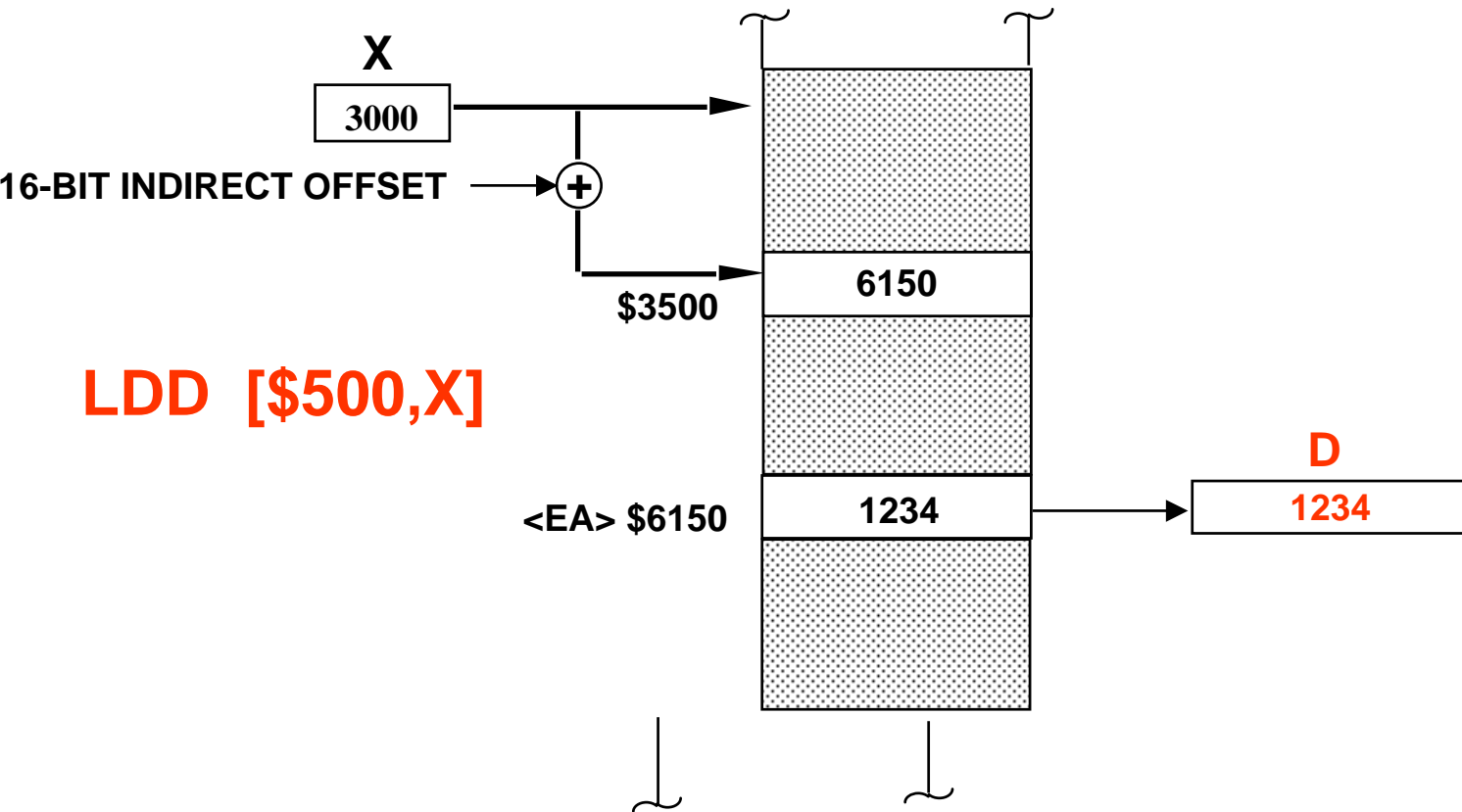
- 9 BIT OFFSET IS SIGNED INCLUDED IN EXTENSION BYTE FOLLOWING INSTRUCTION.
- X, Y, SP OR PC REGISTER CAN BE USED FOR INDEXING.
- OFFSET RANGE FROM -256 TO +255 FROM BASE REGISTER.

Indexed 16-Bit Offset (IDX2)



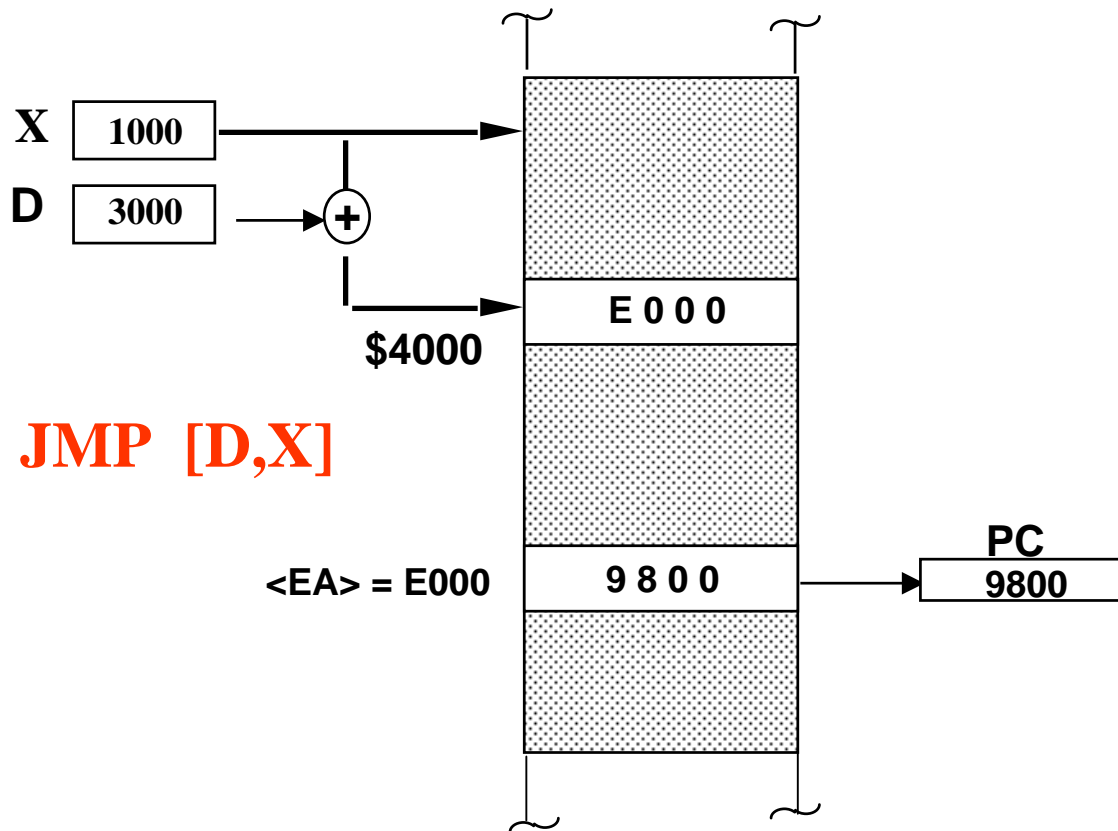
- 16- Bit offset provided in two extension post bytes.
- X, Y, SP OR PC register can be used for indexing.
- Offset Range + or - 32KBytes from base register.

Indexed 16-Bit Indirect Offset ([IDX2])



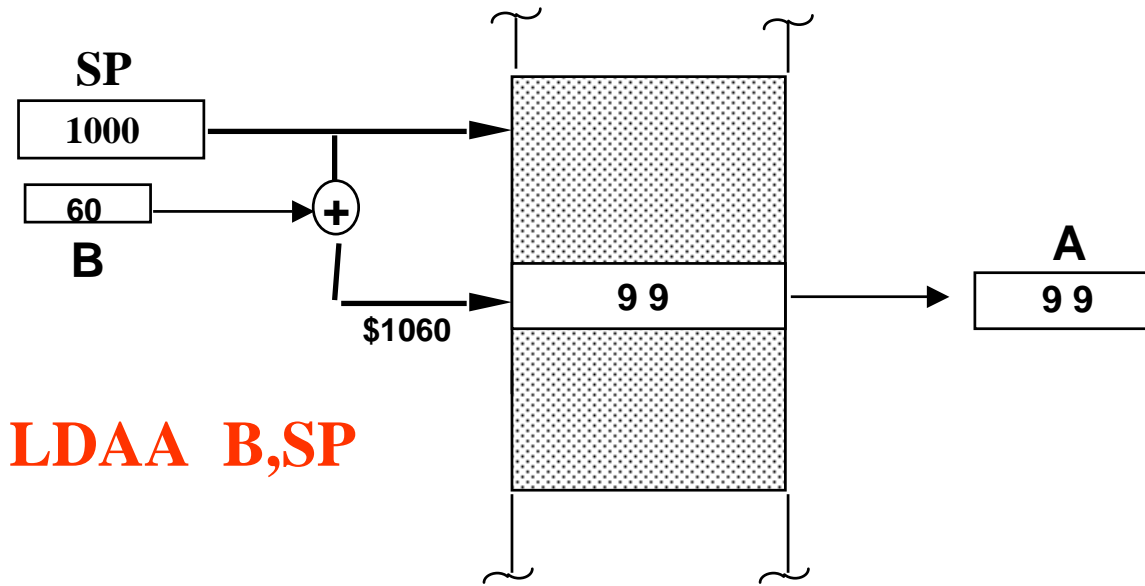
- 16- Bit indirect offset added to base register to form <ea> of the operand.
- X, Y, SP or PC register can be used for indexing.
- Offset range + or - 32k bytes from base register.

Indexed - D-Indirect ([D,IDX])



- **D**- Accumulator added to base register to form <ea> of indirect address.
- **X**, **Y**, **SP** or **PC** register can be used for indexing.
- Offset range + or - 32KBytes from base register.

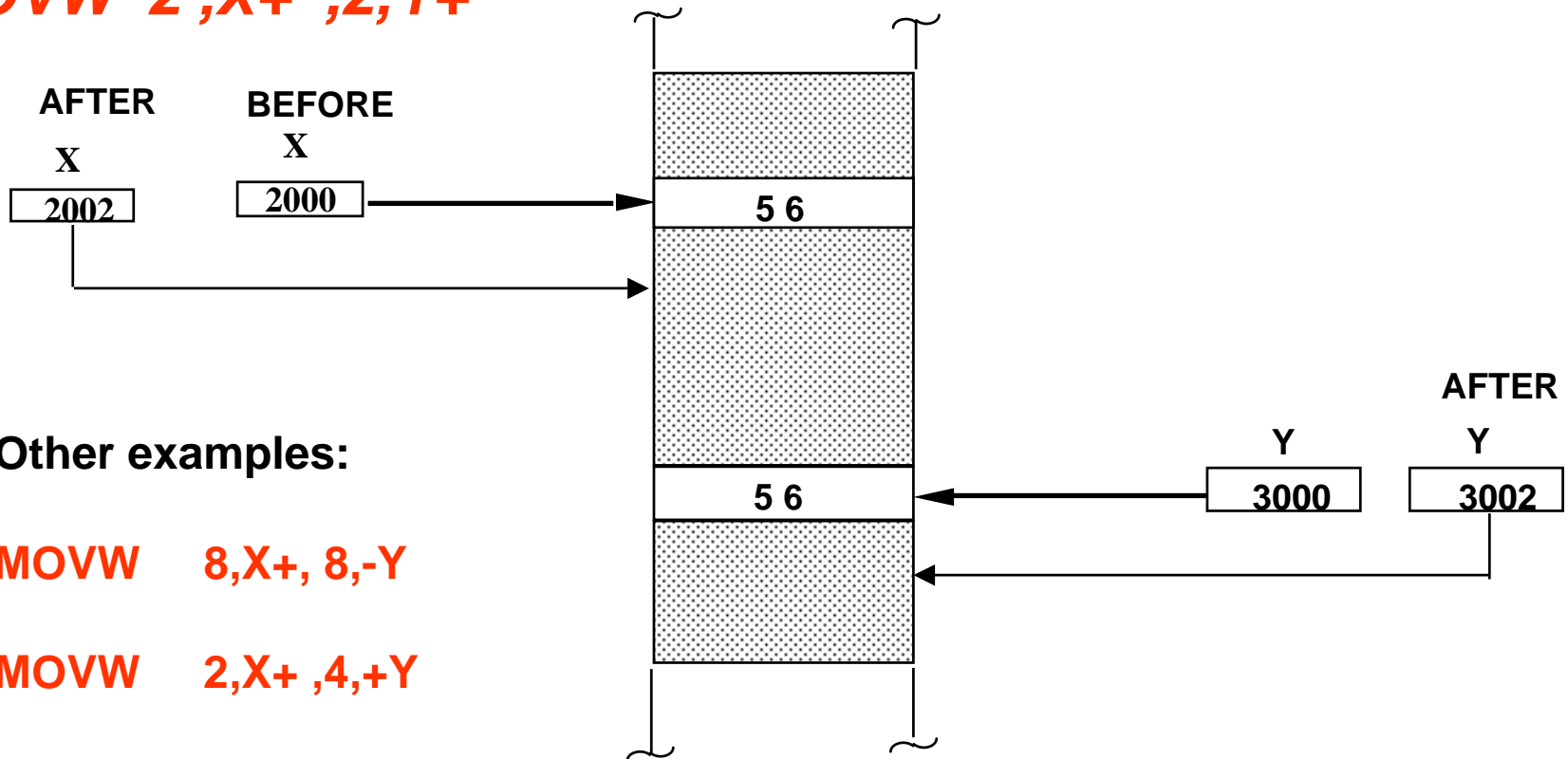
Indexed- Accumulator Offset



- The Accumulator maybe A, B or D .
- X, Y, SP or PC can be used for indexing.
- Offset range up to $\pm 32K$ Bytes from base register.

Indexed - Pre/Post Decrement/Increment

MOVW 2,X+ ,2,Y+



Other examples:

MOVW 8,X+ ,8,-Y

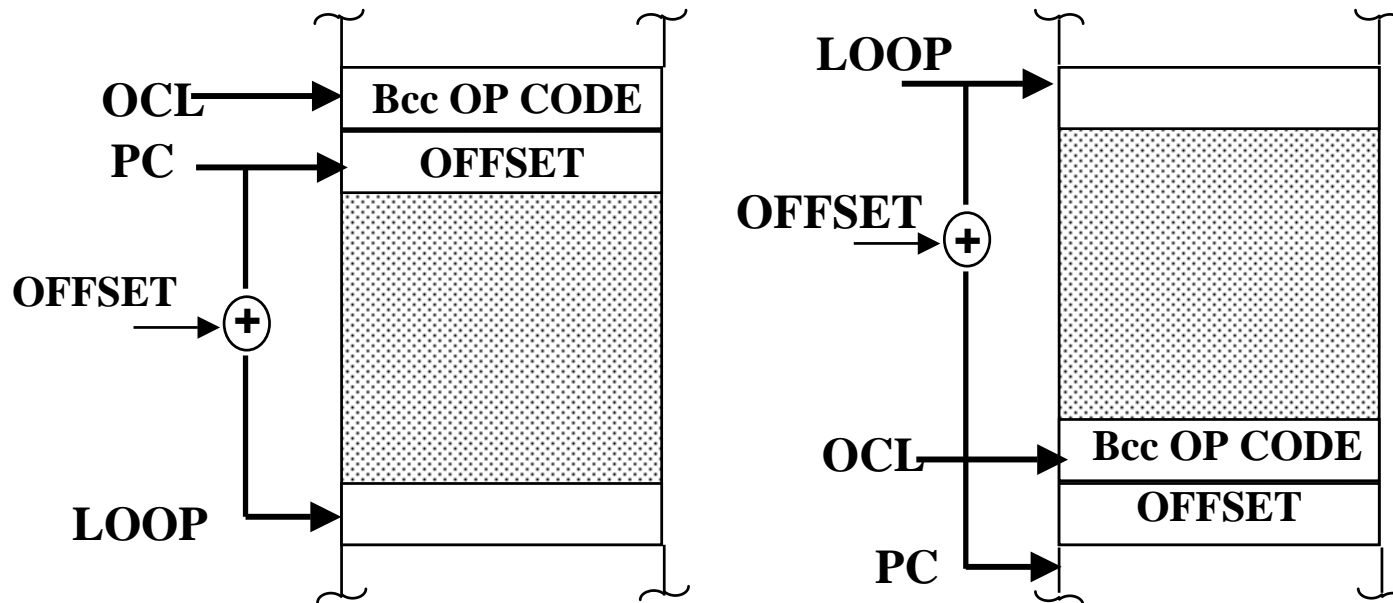
MOVW 2,X+ ,4,+Y

STAA 1,-SP

STAA 4,SP+

Relative Addressing

- Only for **Branch Instructions**. **(L)BEQ LOOP**



- Branch Instructions are 2 or 4 bytes in length.
- All Branches are taken from the next instruction address
(Destination of branch is calculated by adding signed offset to OCL +2 OR +4)

HCS12

Instruction set

INSTRUCTION SET

Data Handling

Arithmetic

Logic

Data Test

Branch

Jump & Subroutine Calls

Source Form	Operation	Mode	Coding (Hex)	Access Detail	S X H T N Z V C								
ADCB #opr8i ADCB opr8a ADCB opr16a ADCB oprx0_xysppc ADCB oprx9_xysppc ADCB oprx16_xysppc ADCB [D,xysppc] ADCB [opr16_xysppc]	Add with carry to B; (B)+(M)+C⇒B or (B)+imm+C⇒B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C9 ii D9 dd F9 hh 11 E9 xb E9 xb ff E9 xb ee ff E9 xb E9 xb ee ff	P rPf rPO rPf rPO frPP fIf rPf fIPrPf	<table><tr><td>-</td><td>-</td><td>Δ</td><td>-</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td></tr></table>	-	-	Δ	-	Δ	Δ	Δ	Δ
-	-	Δ	-	Δ	Δ	Δ	Δ						
ADDA #opr8i ADDA opr8a ADDA opr16a ADDA oprx0_xysppc ADDA oprx9_xysppc ADDA oprx16_xysppc ADDA [D,xysppc] ADDA [opr16_xysppc]	Add to A; (A)+(M)⇒A or (A)+imm⇒A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8B ii 9B dd BB hh 11 AB xb AB xb ff AB xb ee ff AB xb AB xb ee ff	P rPf rPO rPf rPO frPP fIf rPf fIPrPf	<table><tr><td>-</td><td>-</td><td>Δ</td><td>-</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td></tr></table>	-	-	Δ	-	Δ	Δ	Δ	Δ
-	-	Δ	-	Δ	Δ	Δ	Δ						
ADDB #opr8i ADDB opr8a ADDB opr16a ADDB oprx0_xysppc ADDB oprx9_xysppc ADDB oprx16_xysppc ADDB [D,xysppc] ADDB [opr16_xysppc]	Add to B; (B)+(M)⇒B or (B)+imm⇒B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CB ii DB dd FB hh 11 EB xb EB xb ff EB xb ee ff EB xb EB xb ee ff	P rPf rPO rPf rPO frPP fIf rPf fIPrPf	<table><tr><td>-</td><td>-</td><td>Δ</td><td>-</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td></tr></table>	-	-	Δ	-	Δ	Δ	Δ	Δ
-	-	Δ	-	Δ	Δ	Δ	Δ						
ADDD #opr16i ADDD opr8a ADDD opr16a ADDD oprx0_xysppc ADDD oprx9_xysppc ADDD oprx16_xysppc ADDD [D,xysppc] ADDD [opr16_xysppc]	Add to D; (A:B)+(M:M+1)⇒A:B or (A:B)+imm⇒A:B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C3 jj kk D3 dd F3 hh 11 E3 xb E3 xb ff E3 xb ee ff E3 xb E3 xb ee ff	PO rPf rPO rPf rPO frPP fIf rPf fIPrPf	<table><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td></tr></table>	-	-	-	-	Δ	Δ	Δ	Δ
-	-	-	-	Δ	Δ	Δ	Δ						
ANDA #opr8i ANDA opr8a ANDA opr16a ANDA oprx0_xysppc ANDA oprx9_xysppc ANDA oprx16_xysppc ANDA [D,xysppc] ANDA [opr16_xysppc]	AND with A; (A)•(M)⇒A or (A)•imm⇒A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	84 ii 94 dd B4 hh 11 A4 xb A4 xb ff A4 xb ee ff A4 xb A4 xb ee ff	P rPf rPO rPf rPO frPP fIf rPf fIPrPf	<table><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>Δ</td><td>Δ</td><td>0</td><td>-</td></tr></table>	-	-	-	-	Δ	Δ	0	-
-	-	-	-	Δ	Δ	0	-						
ANDB #opr8i ANDB opr8a ANDB opr16a ANDB oprx0_xysppc ANDB oprx9_xysppc ANDB oprx16_xysppc ANDB [D,xysppc] ANDB [opr16_xysppc]	AND with B; (B)•(M)⇒B or (B)•imm⇒B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C4 ii D4 dd F4 hh 11 E4 xb E4 xb ff E4 xb ee ff E4 xb E4 xb ee ff	P rPf rPO rPf rPO frPP fIf rPf fIPrPf	<table><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>Δ</td><td>Δ</td><td>0</td><td>-</td></tr></table>	-	-	-	-	Δ	Δ	0	-
-	-	-	-	Δ	Δ	0	-						

Load and Store

Mnemonic	Function	Operation
Load Instructions		
LDA	Load A	$(M) \Rightarrow A$
LDAB	Load B	$(M) \Rightarrow B$
LDD	Load D	$(M : M + 1) \Rightarrow (A:B)$
LDS	Load SP	$(M : M + 1) \Rightarrow SP_H:SP_L$
LDX	Load index register X	$(M : M + 1) \Rightarrow X_H:X_L$
LDY	Load index register Y	$(M : M + 1) \Rightarrow Y_H:Y_L$
LEAS	Load effective address into SP	Effective address \Rightarrow SP
LEAX	Load effective address into X	Effective address \Rightarrow X
LEAY	Load effective address into Y	Effective address \Rightarrow Y
Store Instructions		
STAA	Store A	$(A) \Rightarrow M$
STAB	Store B	$(B) \Rightarrow M$
STD	Store D	$(A) \Rightarrow M, (B) \Rightarrow M + 1$
STS	Store SP	$(SP_H:SP_L) \Rightarrow M : M + 1$
STX	Store X	$(X_H:X_L) \Rightarrow M : M + 1$
STY	Store Y	$(Y_H:Y_L) \Rightarrow M : M + 1$

DATA HANDLING INSTRUCTIONS

(DATA MOVEMENT)

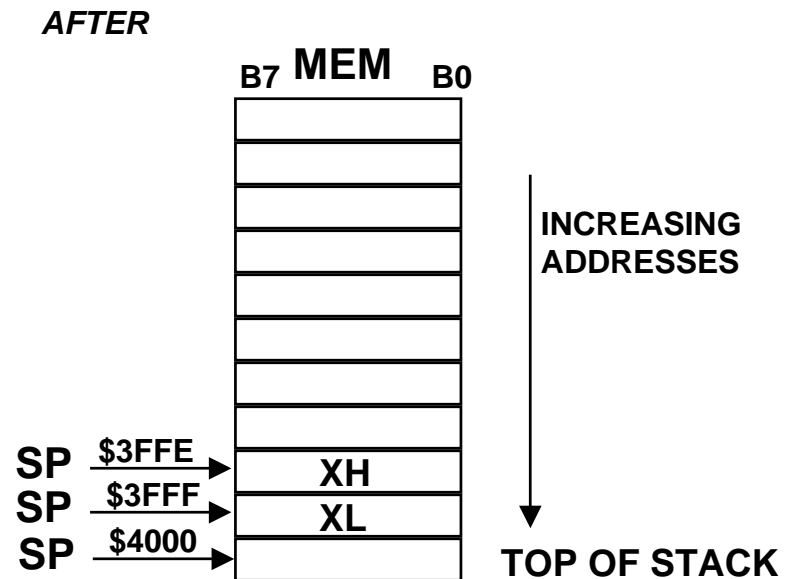
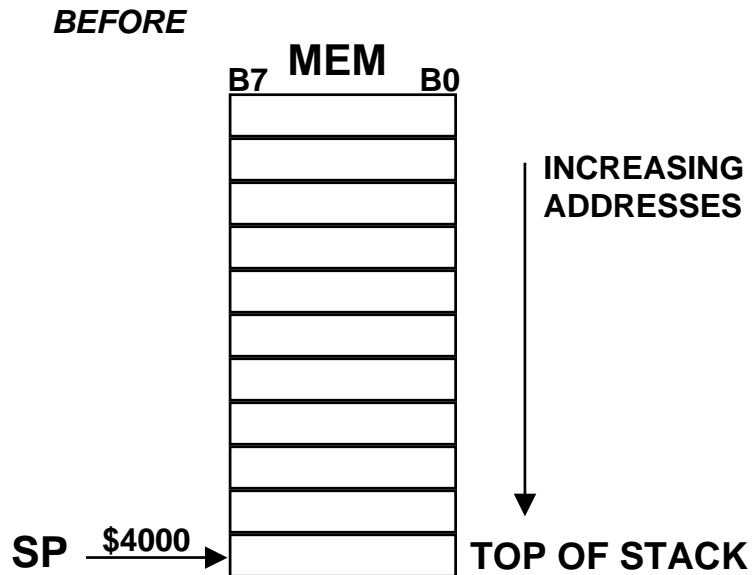
Mnemonic	Function	Operation
PSHA	Push A	$(SP) - 1 \Rightarrow SP; (A) \Rightarrow M_{(SP)}$
PSHB	Push B	$(SP) - 1 \Rightarrow SP; (B) \Rightarrow M_{(SP)}$
PSHC	Push CCR	$(SP) - 1 \Rightarrow SP; (A) \Rightarrow M_{(SP)}$
PSHD	Push D	$(SP) - 2 \Rightarrow SP; (A : B) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PSHX	Push X	$(SP) - 2 \Rightarrow SP; (X) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PSHY	Push Y	$(SP) - 2 \Rightarrow SP; (Y) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PULA	Pull A	$(M_{(SP)}) \Rightarrow A; (SP) + 1 \Rightarrow SP$
PULB	Pull B	$(M_{(SP)}) \Rightarrow B; (SP) + 1 \Rightarrow SP$
PULC	Pull CCR	$(M_{(SP)}) \Rightarrow CCR; (SP) + 1 \Rightarrow SP$
PULD	Pull D	$(M_{(SP)} : M_{(SP+1)}) \Rightarrow A : B; (SP) + 2 \Rightarrow SP$
PULX	Pull X	$(M_{(SP)} : M_{(SP+1)}) \Rightarrow X; (SP) + 2 \Rightarrow SP$
PULY	Pull Y	$(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y; (SP) + 2 \Rightarrow SP$

EXAMPLE: **MOVW 2,X+ , 2,-Y**

STACK OPERATION

EXAMPLE: ***PSHX***

PSHX



DATA HANDLING INSTRUCTIONS

(TRANSFER AND EXCHANGE)

FUNCTION	MNEMONIC	OPERATION
TRANSFER DATA	TBA TAB	$B \rightarrow A$ $A \rightarrow B$
	TXS TYS	$R \rightarrow SP$
	TSY TSX	$SP \rightarrow R$
TRANSFER REG TO REG	TFR	A, B, CCR, D, X, Y, SP $\rightarrow A, B, CCR, D, X, Y, SP$
EXCHANGE	EXG	A, B, CCR, D, X, Y, SP \longleftrightarrow A, B, CCR, D, X, Y, SP
EXCHANGE DATA	XGDX XGDY	$D \longleftrightarrow X$ $D \longleftrightarrow Y$

EXAMPLE1: **TFR X,A**

EXAMPLE2: **EXG Y,B**

DATA HANDLING INSTRUCTIONS

(ALTER DATA)

FUNCTION	MNEMONIC	OPERATION
DECREMENT	DEC DECA DECB	$(M)-1 \rightarrow (M)$ $A-1 \rightarrow A$ $B-1 \rightarrow B$
	DEX DEY DES	$X-1 \rightarrow X$ $Y-1 \rightarrow Y$ $S-1 \rightarrow S$
INCREMENT	INC INCA INCB	$(M)+1 \rightarrow (M)$ $A+1 \rightarrow A$ $B+1 \rightarrow B$
	INX INY INS	$X+1 \rightarrow X$ $Y+1 \rightarrow Y$ $S+1 \rightarrow S$

DATA HANDLING INSTRUCTIONS

(ALTER DATA)

FUNCTION	MNEMONIC	OPERATION
COMPLEMENT, 2'S (NEGATE)	NEG NEGA NEGB	$0-(M) \rightarrow (M)$ $0-A \rightarrow A$ $0-B \rightarrow B$
COMPLEMENT, 1'S	COM COMA COMB	$(\overline{M}) \rightarrow (M)$ $\overline{A} \rightarrow A$ $\overline{B} \rightarrow B$
CLEAR	CLR CLRA CLRB	$0 \rightarrow (M)$ $0 \rightarrow A$ $0 \rightarrow B$
BIT(S) CLEAR	BCLR	$(M) \cdot \overline{MASK} \rightarrow (M)$
BIT(S) SET	BSET	$(M) + MASK \rightarrow (M)$

- Bit Manipulation Example: **BSET OFFSET,X, #MASK**

DATA HANDLING INSTRUCTIONS

Mnemonic	Function	Operation
Minimum Instructions		
EMIND	MIN of two unsigned 16-bit values result to accumulator	$\text{MIN}((D), (M : M + 1)) \Rightarrow D$
EMINM	MIN of two unsigned 16-bit values result to memory	$\text{MIN}((D), (M : M + 1)) \Rightarrow M : M + 1$
MINA	MIN of two unsigned 8-bit values result to accumulator	$\text{MIN}((A), (M)) \Rightarrow A$
MINM	MIN of two unsigned 8-bit values result to memory	$\text{MIN}((A), (M)) \Rightarrow M$
Maximum Instructions		
EMAXD	MAX of two unsigned 16-bit values result to accumulator	$\text{MAX}((D), (M : M + 1)) \Rightarrow D$
EMAXM	MAX of two unsigned 16-bit values result to memory	$\text{MAX}((D), (M : M + 1)) \Rightarrow M : M + 1$
MAXA	MAX of two unsigned 8-bit values result to accumulator	$\text{MAX}((A), (M)) \Rightarrow A$
MAXM	MAX of two unsigned 8-bit values result to memory	$\text{MAX}((A), (M)) \Rightarrow M$

DATA HANDLING INSTRUCTIONS

(SHIFT AND ROTATE)

FUNCTION	MNEMONIC	OPERATION
ROTATE LEFT	ROL ROLA ROLB	<div> <div> M A B </div> </div>
ROTATE RIGHT	ROR RORA RORB	<div> <div> M A B </div> </div>
SHIFT LEFT, ARITHMETIC (LOGICAL)	ASL(LSL) ASLA(LSLA) ASLB(LSLB) ASLD(LSLD)	<div> <div> M A B D </div> </div>
SHIFT RIGHT, ARITHMETIC	ASR ASRA ASRB	<div> <div> M A B </div> </div>
SHIFT RIGHT, LOGICAL	LSR LSRA LSRB LSRD	<div> <div> M A B D </div> </div>

DATA TEST INSTRUCTIONS

FUNCTION	MNEMONIC	TEST
BIT TEST	BITA BITB	A • (M) B • (M)
COMPARE COMPARE STACK	CBA CMPA CMPB	A - B A - (M) B - (M)
	CPD CPX CPY CPS	R_L - (M+1) R_H - (M)-C SP - (M :M +1)
TEST, ZERO OR MINUS	TST TSTA TSTB	(M)-0 A-0 B-0

CONDITIONAL BRANCH INSTRUCTIONS (10F3)

MNEMONIC	CONDITION	CCR TEST	INDICATION
(L) BMI	MINUS	N=1	r=NEGATIVE
(L) BPL	PLUS	N=0	r=POSITIVE
*(L) BVS	OVERFLOW	V=1	r=SIGN ERROR
*(L) BVC	NO OVERFLOW	V=0	r=SIGN OK
*(L) BLT	LESS	$[N \oplus V]=1$	A < M
*(L) BGE	GREATER OR EQUAL	$[N \oplus V]=0$	A >= M
* (L) BLE	LESS OR EQUAL	$[Z + (N \oplus V)]=1$	A <= M
*(L) BGT	GREATER	$[Z + (N \oplus V)]=0$	A > M
(L) BEQ	EQUAL	Z=1	A=M
(L) BNE	NOT EQUAL	Z=0	A <> M
(L) BHI	HIGHER	$[C+Z]=0$	A > M
(L) BLS	LOWER OR SAME	$[C+Z]=1$	A <= M
(L) BCC (BHS)	CARRY CLEAR	C=0	A >= M
(L) BCS (BLO)	CARRY SET	C=1	A < M

Indication refers to the use of a CMPA M instruction immediately before the branch

Use for signed arithmetic only

CONDITIONAL BRANCH INSTRUCTIONS (2 OF 3)

FUNCTION	MNEMONIC	OPERATION
DECREMENT & BRANCH	DBEQ	COUNTER - \$01 → COUNTER IF COUNT. =0, THEN (PC)+\$0003 +REL → PC
	DBNE	COUNTER - \$01, → COUNTER IF COUNT.<>0, THEN (PC)+\$0003 +REL → PC
INCREMENT & BRANCH	IBEQ	COUNTER + \$01 → COUNTER IF COUNTER =0, THEN (PC)+\$0003 +REL → PC
	IBNE	COUNTER + \$01 → COUNTER IF COUNTER <>0, THEN (PC)+\$0003 +REL → PC
TEST & BRANCH	TBEQ	IF COUNTER = 0, THEN PC+\$0003 + REL → PC
	TBNE	IF COUNTER <>0, THEN PC+\$0003 + REL → PC

BRANCH IF BITS SET OR CLEAR (3 of 3)

- SINGLE INSTRUCTION TO LOGICALLY "AND" MASK WITH OPERAND AND BRANCH IF BITS ARE EITHER SET OR CLEARED.
- USEFUL FOR POLLING INTERRUPT STATUS FLAGS, AND FOR MAKING PROGRAM DECISIONS BASED ON BIT(S) VALUES.
- BRANCH IS TAKEN FROM NEXT INSTRUCTION ADDRESS (OCL+4, 5, OR 6)



- ADDRESSING MODES ALLOWED ARE: DIR, EXT, IDX, IDX1 & IDX2.

ARITHMETIC INSTRUCTIONS (1 of 4)

FUNCTION	MNEMONIC	OPERATION
ADD	ADDA ADDB ADDD	$A + (M) \rightarrow A$ $B + (M) \rightarrow B$ $D_L + (M+1) \rightarrow D_L ; D_H + M + C \rightarrow D_H$
ADD ACCUMULATORS	ABA ABX ABY	$A + B \rightarrow A$ $X + B \rightarrow X$ $Y + B \rightarrow Y$
ADD WITH CARRY	ADCA ADCB	$A + M + C \rightarrow A$ $B + M + C \rightarrow B$
DECIMAL ADJUST	DAA	CONVERTS BINARY ADDITION OF BCD CHARS INTO BCD FORMAT

ARITHMETIC INSTRUCTIONS (2 of 4)

FUNCTION	MNEMONIC	OPERATION
SUBTRACT	SUBA SUBB SUBD	$A - (M) \rightarrow A$ $B - (M) \rightarrow B$ $D - (M+1) \rightarrow D_L; D - (M) - C \rightarrow D_H$
SUBTRACT ACCUMULATORS	SBA	$A - B \rightarrow A$
SUBTRACT WITH CARRY	SBCA SBCB	$A - (M) - C \rightarrow A$ $B - (M) - C \rightarrow B$
MULTIPLY	MUL	$A * B \rightarrow D$
EXTENDED MULTIPLY	EMUL	$D * Y \rightarrow Y : D$
EXTENDED MULTIPLY SIGNED	EMULS	$D * Y \rightarrow Y : D$

ARITHMETIC INSTRUCTIONS (3 of 4)

DIVIDE INSTRUCTIONS

- OPERATION **D** REG / **X** REG
- RESULT QUOTIENT IS IN X
REMAINDER IS IN D

- INTEGER DIVIDE **IDIV** / **IDIVS**

RADIX POINT OF THE RESULT IS TO THE RIGHT OF THE LSB

- EXTENDED DIVIDE 32-BIT BY 16-BIT ([UN]SIGNED) EDIV/EDIVS

EDIV EXAMPLE: EDIV[S]

OPERATION $(Y:\overrightarrow{D}) / (X)$ $Y; \overrightarrow{\text{REMAINDER}}$ D

V = 1, IF RESULT > \$FFFF FOR UNSIGNED, UNDEFINED IF DIVISOR IS \$0000

V = 1, IF RESULT > \$7FFF FOR SIGNED, UNDEFINED IF DIVISOR IS \$0000

C = 1, IF DIVISOR WAS \$0000

ARITHMETIC INSTRUCTIONS (4 of 4)

FRACTIONAL DIVIDE INSTRUCTION

- FRACTIONAL DIVIDE **FDIV**

RADIX POINT OF THE RESULT IS TO THE LEFT OF THE MSB

**IF NUMERATOR IS GREATER THAN OR EQUAL TO THE DENOMINATOR,
THEN V FLAG IS SET.**

RESULT EXAMPLES:

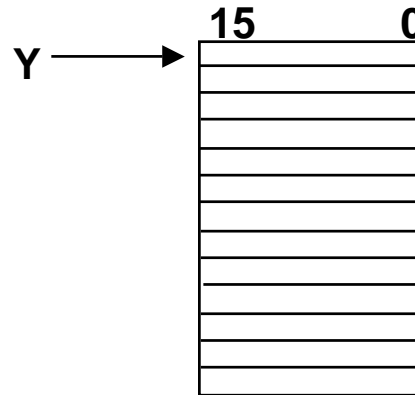
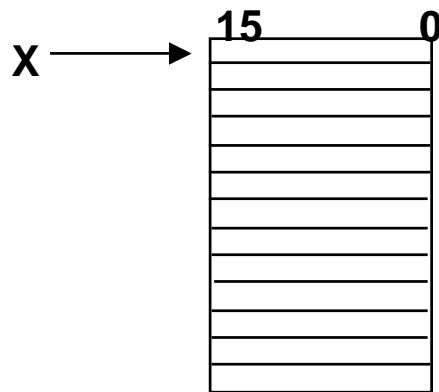
A RESULT OF 1 IS $1/\$10000$ WHICH IS .0001

A RESULT OF \$C000 IS $\$C000/\10000 WHICH IS .75

A RESULT OF \$FFFF IS $\$FFFF/\10000 WHICH IS .9999

EXTENDED MULTIPLY AND ACCUMULATE (EMACS)

OPERATION: $(M : M) * (M : M) + M \sim M+3 \longrightarrow M \sim M+3$
(X) (X+1) (Y) (Y+1)



EXAMPLE:

EMACS \$2500 (* 32-BIT RESULT *)

This is a DSP Instruction, very useful for FFT.....

LOGIC INSTRUCTIONS

FUNCTION	MNEMONIC	OPERATION
AND	ANDA ANDB ANDCC	$A \bullet (M) \rightarrow A$ $B \bullet (M) \rightarrow B$ $CCR \bullet MASK \rightarrow CCR$
EXCLUSIVE OR	EORA EORB	$A \oplus (M) \rightarrow A$ $B \oplus (M) \rightarrow B$
INCLUSIVE OR	ORAA ORAB ORCC	$A + (M) \rightarrow A$ $B + (M) \rightarrow B$ $CCR + MASK \rightarrow CCR$

Jump and Subroutine Call Instructions

Mnemonic	Function	Operation
BSR	Branch to subroutine	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address $\Rightarrow PC$
CALL	Call subroutine in expanded memory	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP$ $(PPAGE) \Rightarrow M_{(SP)}$ Page $\Rightarrow PPAGE$ Subroutine address $\Rightarrow PC$
JMP	Jump	Address $\Rightarrow PC$
JSR	Jump to subroutine	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address $\Rightarrow PC$
RTC	Return from call	$M_{(SP)} \Rightarrow PPAGE$ $SP + 1 \Rightarrow SP$ $M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L$ $SP + 2 \Rightarrow SP$
RTS	Return from subroutine	$M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L$ $SP + 2 \Rightarrow SP$

CONDITION CODE REGISTER INSTRUCTIONS

FUNCTION	MNEMONIC	OPERATION
CLEAR CARRY	CLC	$0 \rightarrow C$
CLEAR INTERRUPT MASK	CLI	$0 \rightarrow I$
CLEAR OVERFLOW	CLV	$0 \rightarrow V$
SET CARRY	SEC	$1 \rightarrow C$
SET INTERRUPT MASK	SEI	$1 \rightarrow I$
SET OVERFLOW	SEV	$1 \rightarrow V$
ACCUMULATOR A \rightarrow CCR	TAP	$A \rightarrow CCR$
CCR \rightarrow ACCUMULATOR A	TPA	$CCR \rightarrow A$
OR CONDITION CODE	ORCC	$CCR + \text{OPERAND}$
AND CONDITION CODE	ANDCC	$CCR \wedge \text{OPERAND}$

Other Instructions

- Fuzzy Logical Control
- Wait, Stop, SWI, RTI
-

Program in Assembler Language

For S12

Assembler Format & Pseudo Instruction

LABEL	OPCODE	OPERAND	COMMENTS
PORTA	EQU	\$0001	; Hardware Address
EOT	EQU	\$04	;ASCII code for end of text
	ORG	\$4000	;Code original from here
GLOBLE	RMB	2	;Reserve2Memory Byte
STRING	FCC	'Hello World'	; = DC.B 'Hello World'
	FCB	EOT	; = DC.B EOT
	FDB	GLOBLE	; = DC.W GLOBLE

Assembler

&

C

```
PORTA    EQU    $0001
```

```
C        RMB    1        ;  
A        RMB    2        ;  
B        RMB    2
```

```
STRING   FCC    "Hello World"  
         FCB    0
```

```
XDEF     GET_CHAR ;  
XREF     main    ;
```

```
#define POATA 0x0001
```

```
char c;  
int a,b;
```

```
const STRING 'Hello World'
```

```
public main;  
extern GET_CHAR;
```

Assembler Example: INCH

```
INCHECK    LDAA    #$20        ; EDRF Bit
           BITA    SCI0SR1     ; Get status
           RTS
```

*

```
INCH       BSR     INCHECK     ; Check status
           BEQ     INCH
           LDAB    SCIDRL      ; Read the Char.
           RTS
```

Assembler Example: OUTCH

```
OUTCH    BRCLR    SCI0SR1, #0x80, *    ; Check if TBF Empty?
        STAA      SCI0DRL              ; Out char.
        RTS
```

* Out space(s) Routine:

```
OUT2S    BSR      OUT1S
OUT1S    LDAA      #0x20                ; ASCII Code for Space ' '
        BRA       OUTCH
```

* Print string PDATA:

```
PRINT    JSR      OUTCH
PDATA    LDAA      ,X+
        CMPA      #EOT
        BNE       PRINT
        RTS
```

Programming Skill in ASM (28Bytes)

* Getting month length

```
M_LENGTH    LDAA    #30                ; Normal month 30 days
              LDAB    #2                ; If Feb?
              CMPB    MONTH
              BEQ     FEBRARY
              LDAB    MONTH
              CMPB    #8                ; Before August
              BLT     ODDBIG            ; Odd Month 31 Days.
              INCB                     ; Even to Odd or Odd to Even
ODDBIG       ANDB     #1                ; Only Use Bit 0
              ABA                      ; Result in A
              RTS
FEBRARY      DECA                     ; If Leap Year?
              LDAB    YEAR
              ANDB    #3                ; If (Year%4) = 0?
              BEQ     FINI              ; Month Length = 29
              DECA                     ; Month Length = 28
FINI         RTS
              END
```

Same Program in C (83 bytes)

```
/* Module month-length */
```

```
extern char year, month;
```

```
char m-length()
```

```
{
```

```
if (month==2)
```

```
{ if (year % 4) return(28);
```

```
else return(29);
```

```
}
```

```
else {
```

```
if ((month==4)||((month==6)||((month==9)||((month==11))
```

```
return(30);
```

```
else return(31);
```

```
}
```

```
}
```


CLEAR RAM ROUTINE

Write a routine to clear the RAM, from \$2000 to \$2FFF.

CLR RAM_RTN:

```
..... ; Initialize X pointer to RAM start($2000)
LOOP: ..... ; Clear memory pointed by X ,
..... ; Increment X
CPX    #$3000 ; Compare pointer with $3000
..... ; If pointer not equal, Branch to LOOP
DONE   BRA    DONE ; End program here
```

CLEAR RAM ROUTINE

Write a routine to clear the RAM, from \$2000 to \$2FFF.

CLRRAM_RTN:

```
        LDX    #$2000    ; Initialize X to RAMstart($2000)
LOOP:    CLR    ,X        ; Clear memory pointed to by X,
        INX                    ; Increment X
        CPX    #$3000    ; Compare pointer with $3000
        BNE    LOOP      ; If not equal, Branch to LOOP
DONE     BRA    DONE      ; End program here
```

You can use: CLR 1,X+

to replace: CLR ,X
 INX

BLOCK MOVE ROUTINE

COPIE DATA FROM \$1000 TO \$1100. END, IF DATA=0.

WRITE YOUR PROGRAM HERE:

```
          ORG      $1000
SOURCE    FCC      'DATA TO MOVE'
          FCB      0
```

```
          ORG      $4000
BEGIN     .....
          .....
          .....
          .....
          BEQ      DONE
          .....
          .....
          .....
          .....
DONE      BRA      DONE
```

SUGGESTED STEPS

ORIGINATE DATA AT \$1000.
FORM DATA TO BE MOVED
FORM CONSTANT BYTE OF '0'.

PROGRAM BEGINS @\$4000.

1. INIT SOURCE POINTER TO \$1000.
2. DESTINATION POINTE TO \$1100.
3. GET DATA FROM SOURCE ADDRESS.
4. WRITE DATA TO DESTINATION,
5. IF DATA=0, GOTO 9, ELSE TO 6.
6. INCREMENT SOURCE POINTER.
7. INCREMENT DESTINATION POINTER
8. GO TO STEP 3.
9. STAY HERE.

BLOCK MOVE ROUTINE

COPIE DATA FROM \$1000 TO \$1100. END, IF DATA=0.

WRITE YOUR PROGRAM HERE

SOURCE **ORG** \$1000
 FCC 'DATA TO MOVE'
 FCB 0

BEGIN **ORG** \$4000
 LDX #\$1000

LDY #\$1100

LOOP **LDAA** ,X

STAA ,Y

BEQ DONE

INX

INY

BRA LOOP

DONE **BRA** DONE

SUGGESTED PROGRAM STEPS

ORIGINATE DATA AT ADDRESS \$1000.

FORM TABLE OF DATA TO BE MOVED

FORM CONSTANT BYTE OF '0'.

PROGRAM BEGINS @\$4000.

; 1. INIT SOURCE POINTER TO \$1000.

; 2. DESTINATION POINTE TO \$1100.

3. GET DATA FROM SOURCE ADDRESS.

4. WRITE DATA TO DESTINATION,

; 5. IF DATA=0, GOTO 9, ELSE TO 6.

6. INCREMENT SOURCE POINTER.

7. INCREMENT DESTINATION POINTER

; 8. GO TO STEP 3.

; 9. STAY HERE.

How about using: **LDAA 1,X+ ?**

Try to make it works at experiment course !