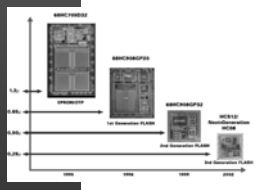
CPU12

The CPU12 Central Processing Unit

M68HC12 and HCS12 microcontrollers

Babak Kia
Adjunct Professor
Boston University
College of Engineering
Email: bkia -at- bu.edu

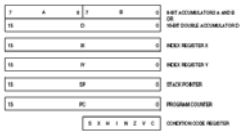
Introduction



- **The CPU12 is a high-speed, 16-bit processing unit, with an identical programming model to the industry standard M68HC11 CPU**
- **High performance with 40 ns minimum instruction cycles**
- **Flexible addressing modes, on-chip Flash, and a whole slew of on-chip peripherals**

CPU12 Programming Model

- **The CPU12 has two 8-bit general purpose accumulators (A and B)**
- **A and B can also be accessed as a single 16-bit register (D)**
- **Two index registers (X and Y) used for indexed addressing**
- **16-bit Stack Pointer (SP), locatable anywhere in the 64Kb address space**
- **16-bit Program Counter (PC)**
- **8-bit Condition Code Register (CCR)**



CPU12 Programming Model

- **The Condition Code Register consists of 5 status indicators, 2 interrupt masks, and a STOP instruction control bit**
  - ***S Control Bit*. Clearing the S bit enables the STOP instruction which stops the on-chip oscillator. If set, the CPU treats the STOP instruction as a NOP.**
  - ***X Mask Bit*. The XIRQ# signal poses as a modified version of the NMI# interrupt. Enabling a non-maskable interrupt before a system is fully operational can lead to spurious interrupts. The X bit allows for the enabling of the NMI once the system is operational**
  - ***H Status Bit*. The H bit indicates a half-carry on the accumulator. It is used by the DAA instruction on BCD operations**

CPU12 Programming Model

- ***I Mask Bit*. The I bit enables or disables all maskable interrupts. The I bit is set to 1 (masked) at reset. While I is set, interrupts can become pending and are remembered, but the system is not interrupted until the I mask is cleared.**
- ***N Status Bit*. The N bit is mostly used in two's complement arithmetic and shows the state of the MSB of the result.**
- ***Z Status Bit*. Z is set when the result of an operation is 0.**
- ***V Status Bit*. V is the overflow indicator.**
- ***C Status Bit*. C is set when carry occurs during an addition, or a borrow is needed during subtraction.**

Addressing Mode

- **The CPU12 employs an extensive Addressing Mode scheme:**
  - **Inherent**
  - **Immediate**
  - **Direct**
  - **Extended**
  - **Relative**
  - **Indexed (5-bit offset)**
  - **Indexed (pre-dec.)**
  - **Indexed (pre-inc.)**
  - **Indexed (post-dec.)**
  - **Indexed (post-inc.)**
  - **Indexed (accumulator offset)**
  - **Indexed (9-bit offset)**
  - **Indexed (16-bit offset)**
  - **Indexed-Indirect (16-bit offset)**
  - **Indexed-Indirect (D accumulator offset)**

Addressing Mode

- **Inherent Addressing Mode**
  - **Instructions that use this addressing mode either have no operands, or the operands are CPU registers**
  - **NOP ; Does not have an operand**
  - **INX ; Operand is a CPU register**
- **Immediate Addressing Mode**
  - **Operands for Immediate Addressing mode are present during normal program fetch cycle. The '#' symbol is used to identify the immediate operand**
  - **LDAA #$55 ; Load A with 8-bit value**
  - **LDX #$1234 ; Load X register with 16-bit value**

Addressing Mode

- **Direct Addressing Mode**
  - **Also called a zero-page addressing mode because it is used for operands in the $0000 - $00FF memory range. A system can be optimized by placing most commonly used operands in this range**
  - **LDAA $55 ; Load A with content at addr $55**
- **Extended Addressing Mode**
  - **In this addressing mode, the full 16-bit address of the memory location is employed**
  - **LDAA $F123**
- **Relative Addressing Mode**
  - **Used by the branch instruction**
  - **The offset provided is added to the address of the next memory location after the offset to form an effective address**

Addressing Mode

- **Indexed Addressing: 5-bit offset**
  - **The 5-bit offset is added to the base index register (X, Y, SP, or PC) to determine the effective address**
  - **Provides a range of -16 through +15**
  - **Majority of indexed instructions use offsets that fit in the 5-bit size indexing mode**
- **Indexed Addressing: 9-bit offset**
  - **Same as above, but with 9-bit offsetting.**
  - **Allows for a range of –256 through +256 from the value of the base index register**
  - **Example: "LDAB -20,Y", for Y = $2000, loads B with the value at address $1FEC**

Addressing Mode

- **Indexed Addressing: 16-bit offset**
  - **Allows access to addresses anywhere within the 64 Kbyte address space**
  - **Does not matter whether it is signed or unsigned, since $FFFF can be treated either as +65,535, or as a -1**
- **Indexed-Indirect Addressing: 16-bit offset**
  - **Unlike indexed addressing, this addressing mode is used to change the contents of the memory pointed to by an address**
  - **For example,**

```
LDAA [10,X]        ; X = $1000, *($1010) = $1234
Takes the address contained in X and adds 10 to it.
Loads the contents of memory location $1010
to make A = $1234
```

Addressing Mode

- **Auto Pre/Post Increment/Decrement Indexed Addressing**
  - **This addressing mode provides four ways of automatically incrementing or decrementing a pointer after processing the instruction**
  - **Pre inc/dec adjusts the value of the index register prior to using it**
  - **Post inc/dec adjusts the value of the index register after the operation is complete**
  - **Example**
    - **STAA 1,-SP ; Equivalent to PSHA**
    - **STX 2,-SP ; Equivalent to PSHX**
    - **LDX 2,SP+ ; Equivalent to PULX**
    - **LDAA 1,SP+ ; Equivalent to PULA**

Addressing Mode

- **Accumulator Offset Indexed Addressing**
  - **In this addressing mode, the effective offset is the sum of the value of the index register, and the unsigned offset in the accumulator**
  - **Example**

```
LDAA B,X    Loads A with contents of X+B
```

- **Accumulator D Indirect Indexed Addressing**
  - **Similar to above, except that it points to the memory location which contains the value to load**

```
JMP         [D,PC]
GO1         DC.W            PLACE1
GO2         DC.W            PLACE2
GO3         DC.W            PLACE3
```

  - **This instruction acts as a computed GOTO. When the JMP instruction is executed, PC points to GO1. Therefore values for D of 0, 2, 4, will point to GO1, GO2, or GO3**

Addressing Mode

- **Addressing above the 64K boundary**
  - **Systems that employ a large linear address space also require wider instruction sets that address the larger memory space. This causes an unnecessary overhead for the CPU12 class of processors.**
  - **The CPU12 employs a transparent bank-switching scheme**
  - **Interrupts need not be turned off during bank-switching**
  - **The CPU12 treats a 16K range of memory from $8000 to $BFFF as a program memory window**
  - **The Program Page Register (PPAGE) allows for 256 16Kbyte pages to be switched in and out of the program memory window**
  - **This provides up to 4 Megabytes of memory**

Exception Processing

- **The CPU12 has several sources of Interrupts**
  - **Reset**
    - **Power-on Reset (POR) and RESET#**
    - **Clock Monitor Reset**
    - **COP Watchdog Reset**
  - **Unimplemented Opcode Trap**
  - **Software Interrupt Instruction (SWI)**
  - **Non Maskable Interrupt (X-bit)**
  - **Non Maskable Interrupt (I-bit)**
- **The CPU12 can handle up to 128 exception vectors, but the number varies with each device**

Exception Processing

- **CPU12 Exception Vector Map**

Vector Address	Source
$FFFE–$FFFF	System reset
$FFFC–$FFFD	Clock monitor reset
$FFFA–$FFFB	COP reset
$FFF8–$FFF9	Unimplemented opcode trap
$FFF6–$FFF7	Software interrupt instruction (SWI)
$FFF4–$FFF5	$\overline{XIRQ}$ signal
$FFF2–$FFF3	$\overline{IRQ}$ signal
$FF00–$FFF1	Device-specific interrupt sources (HCS12)
$FFC0–$FFF1	Device-specific interrupt sources (M68HC12)

Exception Processing

- **CPU12 Exception Priority**
  1. **RESET# or POR**
  2. **Clock Monitor Reset**
  3. **COP Watchdog Reset**
  4. **NMI (XIRQ#)**
  5. **Unimplemented Opcode**
  6. **Software Interrupt (SWI)**
- **The remaining interrupts are subject to masking via the I bit in the CCR.**
  - **However, other than the IRQ# pin and the internal periodic real-time interrupt generator, all other maskable sources follow a priority directly related to the address of their interrupt vectors.**
  - **The higher the address, the higher the Priority**

Instruction Set Review

- **Load and Store Instructions**

Mnemonic	Function	Operation
	Load Instructions	
LDAA	Load A	(M) ⇨ A
LDAB	Load B	(M) ⇨ B
LDD	Load D	(M : M + 1) ⇨ (A:B)
LDS	Load SP	(M : M + 1) ⇨ SP_H:SP_L
LDX	Load index register X	(M : M + 1) ⇨ X_H:X_L
LDY	Load index register Y	(M : M + 1) ⇨ Y_H:Y_L
LEAS	Load effective address into SP	Effective address ⇨ SP
LEAX	Load effective address into X	Effective address ⇨ X
LEAY	Load effective address into Y	Effective address ⇨ Y

	Store Instructions	
STAA	Store A	(A) ⇨ M
STAB	Store B	(B) ⇨ M
STD	Store D	(A) ⇨ M, (B) ⇨ M + 1
STS	Store SP	(SP_H:SP_L) ⇨ M : M + 1
STX	Store X	(X_H:X_L) ⇨ M : M + 1
STY	Store Y	(Y_H:Y_L) ⇨ M : M + 1

Instruction Set Review

- **Transfer and Exchange Instructions**

Mnemonic	Function	Operation
	Transfer Instructions	
TAB	Transfer A to B	(A) ⇨ B
TAP	Transfer A to CCR	(A) ⇨ CCR
TBA	Transfer B to A	(B) ⇨ A
TFR	Transfer register to register	(A, B, CCR, D, X, Y, or SP) ⇨ A, B, CCR, D, X, Y, or SP
TPA	Transfer CCR to A	(CCR) ⇨ A
TSX	Transfer SP to X	(SP) ⇨ X
TSY	Transfer SP to Y	(SP) ⇨ Y
TXS	Transfer X to SP	(X) ⇨ SP
TYS	Transfer Y to SP	(Y) ⇨ SP
	Exchange Instructions	
EXG	Exchange register to register	(A, B, CCR, D, X, Y, or SP) ⇔ (A, B, CCR, D, X, Y, or SP)
XGDX	Exchange D with X	(D) ⇔ (X)
XGDY	Exchange D with Y	(D) ⇔ (Y)
	Sign Extension Instruction	
SEX	Sign extend 8-Bit operand	Sign-extended (A, B, or CCR) ⇨ D, X, Y, or SP

Instruction Set Review

- **Move Instructions**

Mnemonic	Function	Operation
MOVB	Move byte (8-bit)	$(M_s) \Rightarrow M_d$
MOVW	Move word (16-bit)	$(M:M+1_s) \Rightarrow M:M+1_d$

- **Binary Coded Instructions**

Mnemonic	Function	Operation
ABA	Add B to A	(A) + (B) ⇒ A
ADCA	Add with carry to A	(A) + (M) + C ⇒ A
ADCB[1]	Add with carry to B	(B) + (M) + C ⇒ B
ADDA[1]	Add memory to A	(A) + (M) ⇒ A
ADDB	Add memory to B	(B) + (M) ⇒ B
DAA	Decimal adjust A	(A)=⊡

Instruction Set Review

- **Addition and Subtraction Instructions**

Mnemonic	Function	Operation
	Addition Instructions	
ABA	Add B to A	(A) + (B) ⇒ A
ABX	Add B to X	(B) + (X) ⇒ X
ABY	Add B to Y	(B) + (Y) ⇒ Y
ADCA	Add with carry to A	(A) + (M) + C ⇒ A
ADCB	Add with carry to B	(B) + (M) + C ⇒ B
ADDA	Add without carry to A	(A) + (M) ⇒ A
ADDB	Add without carry to B	(B) + (M) ⇒ B
ADDD	Add to D	(A:B) + (M : M + 1) ⇒ A : B
	Subtraction Instructions	
SBA	Subtract B from A	(A) – (B) ⇒ A
SBCA	Subtract with borrow from A	(A) – (M) – C ⇒ A
SBCB	Subtract with borrow from B	(B) – (M) – C ⇒ B
SUBA	Subtract memory from A	(A) – (M) ⇒ A
SUBB	Subtract memory from B	(B) – (M) ⇒ B
SUBD	Subtract memory from D (A:B)	(D) – (M : M + 1) ⇒ D

Instruction Set Review

- **Decrement and Increment Instructions**

Mnemonic	Function	Operation
	Decrement Instructions	
DEC	Decrement memory	(M) – $01 ⇒ M
DECA	Decrement A	(A) – $01 ⇒ A
DECB	Decrement B	(B) – $01 ⇒ B
DES	Decrement SP	(SP) – $0001 ⇒ SP
DEX	Decrement X	(X) – $0001 ⇒ X
DEY	Decrement Y	(Y) – $0001 ⇒ Y
	Increment Instructions	
INC	Increment memory	(M) + $01 ⇒ M
INCA	Increment A	(A) + $01 ⇒ A
INCB	Increment B	(B) + $01 ⇒ B
INS	Increment SP	(SP) + $0001 ⇒ SP
INX	Increment X	(X) + $0001 ⇒ X
INY	Increment Y	(Y) + $0001 ⇒ Y

Instruction Set Review

- **Compare and Test Instructions**

Mnemonic	Function	Operation
	Compare Instructions	
CBA	Compare A to B	(A) – (B)
CMPA	Compare A to memory	(A) – (M)
CMPB	Compare B to memory	(B) – (M)
CPD	Compare D to memory (16-bit)	(A : B) – (M : M + 1)
CPS	Compare SP to memory (16-bit)	(SP) – (M : M + 1)
CPX	Compare X to memory (16-bit)	(X) – (M : M + 1)
CPY	Compare Y to memory (16-bit)	(Y) – (M : M + 1)
	Test Instructions	
TST	Test memory for zero or minus	(M) – $00
TSTA	Test A for zero or minus	(A) – $00
TSTB	Test B for zero or minus	(B) – $00

Instruction Set Review

- **Boolean Logic Instructions**

Mnemonic	Function	Operation
ANDA	AND A with memory	(A) • (M) ⇨ A
ANDB	AND B with memory	(B) • (M) ⇨ B
ANDCC	AND CCR with memory (clear CCR bits)	(CCR) • (M) ⇨ CCR
EORA	Exclusive OR A with memory	(A) ⊕ (M) ⇨ A
EORB	Exclusive OR B with memory	(B) ⊕ (M) ⇨ B
ORAA	OR A with memory	(A) + (M) ⇨ A
ORAB	OR B with memory	(B) + (M) ⇨ B
ORCC	OR CCR with memory (set CCR bits)	(CCR) + (M) ⇨ CCR

- **Bit Test & Manipulation Instructions**

Mnemonic	Function	Operation
BCLR	Clear bits in memory	(M) • ($\overline{mm}$) ⇨ M
BITA	Bit test A	(A) • (M)
BITB	Bit test B	(B) • (M)
BSET	Set bits in memory	(M) + (mm) ⇨ M

Instruction Set Review

- **Clear, Complement, and Negate Instructions**

Mnemonic	Function	Operation
CLC	Clear C bit in CCR	0 ⇨ C
CLI	Clear I bit in CCR	0 ⇨ I
CLR	Clear memory	$00 ⇨ M
CLRA	Clear A	$00 ⇨ A
CLRB	Clear B	$00 ⇨ B
CLV	Clear V bit in CCR	0 ⇨ V
COM	One's complement memory	$FF – (M) ⇨ M or $\overline{(M)}$ ⇨ M
COMA	One's complement A	$FF – (A) ⇨ A or $\overline{(A)}$ ⇨ A
COMB	One's complement B	$FF – (B) ⇨ B or $\overline{(B)}$ ⇨ B
NEG	Two's complement memory	$00 – (M) ⇨ M or $\overline{(M)}$ + 1 ⇨ M
NEGA	Two's complement A	$00 – (A) ⇨ A or $\overline{(A)}$ + 1 ⇨ A
NEGB	Two's complement B	$00 – (B) ⇨ B or $\overline{(B)}$ + 1 ⇨ B

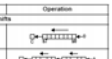
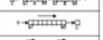
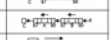
Instruction Set Review

- **Multiplication and Division Instructions**

Mnemonic	Function	Operation
	Multiplication Instructions	
EMUL	16 by 16 multiply (unsigned)	(D) × (Y) ⇒ Y : D
EMULS	16 by 16 multiply (signed)	(D) × (Y) ⇒ Y : D
MUL	8 by 8 multiply (unsigned)	(A) × (B) ⇒ A : B
	Division Instructions	
EDIV	32 by 16 divide (unsigned)	(Y : D) ÷ (X) ⇒ Y Remainder ⇒ D
EDIVS	32 by 16 divide (signed)	(Y : D) ÷ (X) ⇒ Y Remainder ⇒ D
FDIV	16 by 16 fractional divide	(D) ÷ (X) ⇒ X Remainder ⇒ D
IDIV	16 by 16 integer divide (unsigned)	(D) ÷ (X) ⇒ X Remainder ⇒ D
IDIVS	16 by 16 integer divide (signed)	(D) ÷ (X) ⇒ X Remainder ⇒ D

Instruction Set Review

- **Shift and Rotate Instructions**

Mnemonic	Function	Operation
	Logical Shifts	
LSL LSLA LSLB	Logic shift left memory Logic shift left A Logic shift left B	
LSLD	Logic shift left D	
LSR LSRA LSRB	Logic shift right memory Logic shift right A Logic shift right B	
LSRD	Logic shift right D	
	Arithmetic Shifts	
ASL ASLA ASLB	Arithmetic shift left memory Arithmetic shift left A Arithmetic shift left B	
ASLD	Arithmetic shift left D	
ASR ASRA ASRB	Arithmetic shift right memory Arithmetic shift right A Arithmetic shift right B	
	Rotates	
ROL ROLA ROLB	Rotate left memory through carry Rotate left A through carry Rotate left B through carry	
ROR RORA RORB	Rotate right memory through carry Rotate right A through carry Rotate right B through carry	

Instruction Set Review

- **Fuzzy Logic Weighted Average Instruction**
  - **Computes a sum-of-products and a sum-of-weights for defuzzification**
- **Maximum and Minimum Instructions**
  - **Used to make comparisons between an accumulator and a memory location. Used for fuzzification**
- **Multiply and Accumulate Instruction**

Mnemonic	Function	Operation
EMACS	Multiply and accumulate (signed) 16 bit by 16 bit ⇒ 32 bit	$([M_{(Y)}:M_{(Y+1)}] \times [M_{(Y)}:M_{(Y+1)}]$ $+ (M - M + 3) \Rightarrow M - M + 3$

Instruction Set Review

- **Table Interpolation Instructions**
  - **Interpolate values from tables stored in memory.**
  - **Any function which can be represented as a series of linear equations can be represented by this method**

Mnemonic	Function	Operation
ETBL	16-bit table lookup and interpolate (no indirect addressing modes allowed)	(M : M + 1) + [(B) × ((M + 2 : M + 3) − (M : M + 1))] ⇨ D Initialize B, and index before ETBL. <ea> points to the first table entry (M : M + 1) B is fractional part of lookup value
TBL	8-bit table lookup and interpolate (no indirect addressing modes allowed)	(M) + [(B) × ((M + 1) − (M))] ⇨ A Initialize B, and index before TBL. <ea> points to the first 8-bit table entry (M) B is fractional part of look-up value.

Instruction Set Review

- **Short Branch Instructions**

Mnemonic	Function	Equation or Operation
Unary Branches		
BRA	Branch always	1 = 1
BRN	Branch never	1 = 0
Simple Branches		
BCC	Branch if carry clear	C = 0
BCS	Branch if carry set	C = 1
BEQ	Branch if equal	Z = 1
BMI	Branch if minus	N = 1
BNE	Branch if not equal	Z = 0
BPL	Branch if plus	N = 0
BVC	Branch if overflow clear	V = 0
BVS	Branch if overflow set	V = 1
Unsigned Branches		
		Relation
BHI	Branch if higher	R > M
BHS	Branch if higher or same	R ≥ M
BLO	Branch if lower	R < M
BLS	Branch if lower or same	R ≤ M
Signed Branches		
BGE	Branch if greater than or equal	R ≥ M
BGT	Branch if greater than	R > M
BLE	Branch if less than or equal	R ≤ M
BLT	Branch if less than	R < M

		C + Z = 0
		C = 0
		C = 1
		C + Z = 1
		N ⊕ V = 0
		Z + (N ⊕ V) = 0
		Z + (N ⊕ V) = 1
		N ⊕ V = 1

Instruction Set Review

- **Long Branch Instructions**

Mnemonic	Function	Equation or Operation
Unary Branches		
LBRA	Long branch always	1 = 1
LBRN	Long branch never	1 = 0
Simple Branches		
LBCC	Long branch if carry clear	C = 0
LBCS	Long branch if carry set	C = 1
LBEQ	Long branch if equal	Z = 1
LBMI	Long branch if minus	N = 1
LBNE	Long branch if not equal	Z = 0
LBPL	Long branch if plus	N = 0
LBVC	Long branch if overflow clear	V = 0
LBVS	Long branch if overflow set	V = 1
Unsigned Branches		
LBHI	Long branch if higher	C + Z = 0
LBHS	Long branch if higher or same	C = 0
LBLO	Long branch if lower	Z = 1
LBLS	Long branch if lower or same	C + Z = 1
Signed Branches		
LBGE	Long branch if greater than or equal	N ⊕ V = 0
LBGT	Long branch if greater than	Z + (N ⊕ V) = 0
LBLE	Long branch if less than or equal	Z + (N ⊕ V) = 1
LBLT	Long branch if less than	N ⊕ V = 1

Instruction Set Review

- **Bit Condition Branch Instructions**

Mnemonic	Function	Equation or Operation
BRCLR	Branch if selected bits clear	(M) • (mm) = 0
BRSET	Branch if selected bits set	(M) • (mm) = 0

- **Loop Primitive Instructions**

Mnemonic	Function	Equation or Operation
DBEQ	Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1 ⇒ counter If (counter) = 0, then branch; else continue to next instruction
DBNE	Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1 ⇒ counter If (counter) not = 0, then branch; else continue to next instruction
IBEQ	Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1 ⇒ counter If (counter) = 0, then branch; else continue to next instruction
IBNE	Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1 ⇒ counter If (counter) not = 0, then branch; else continue to next instruction
TBEQ	Test counter and branch if = 0 (counter = A, B, D, X,Y, or SP)	If (counter) = 0, then branch; else continue to next instruction
TBNE	Test counter and branch if ≠ 0 (counter = A, B, D, X,Y, or SP)	If (counter) not = 0, then branch; else continue to next instruction

Instruction Set Review

- **Jump and Subroutine Instructions**

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

Instruction Set Review

- **Interrupt Instructions**

Mnemonic	Function	Operation
RTI	Return from interrupt	$(M_{(SP)})$ ⇒ CCR; (SP) + $0001 ⇒ SP $(M_{(SP)}$: $M_{(SP+1)})$ ⇒ B ; A; (SP) + $0002 ⇒ SP $(M_{(SP)}$: $M_{(SP+1)})$ ⇒ X_H : X_L; (SP) + $0004 ⇒ SP $(M_{(SP)}$: $M_{(SP+1)})$ ⇒ PC_H: PC_L; (SP) + $0002 ⇒ SP $(M_{(SP)}$: $M_{(SP+1)})$ ⇒ Y_H : Y_L; (SP) + $0004 ⇒ SP
SWI	Software interrupt	SP – 2 ⇒ SP; RTN_H: RTN_L ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 2 ⇒ SP; Y_H : Y_L ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 2 ⇒ SP; X_H : X_L ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 2 ⇒ SP; B : A ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 1 ⇒ SP; CCR ⇒ $M_{(SP)}$
TRAP	Unimplemented opcode interrupt	SP – 2 ⇒ SP; RTN_H: RTN_L ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 2 ⇒ SP; Y_H : Y_L ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 2 ⇒ SP; X_H : X_L ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 2 ⇒ SP; B : A ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 1 ⇒ SP; CCR ⇒ $M_{(SP)}$

Instruction Set Review

- **Index Manipulation Instructions**

Mnemonic	Function	Operation
	Addition Instructions	
ABX	Add B to X	(B) + (X) ⇨ X
ABY	Add B to Y	(B) + (Y) ⇨ Y
	Compare Instructions	
CPS	Compare SP to memory	(SP) – (M : M + 1)
CPX	Compare X to memory	(X) – (M : M + 1)
CPY	Compare Y to memory	(Y) – (M : M + 1)
	Load Instructions	
LDS	Load SP from memory	M : M+1 ⇨ SP
LDX	Load X from memory	(M : M + 1) ⇨ X
LDY	Load Y from memory	(M : M + 1) ⇨ Y
LEAS	Load effective address into SP	Effective address ⇨ SP
LEAX	Load effective address into X	Effective address ⇨ X
LEAY	Load effective address into Y	Effective address ⇨ Y
	Store Instructions	
STS	Store SP in memory	(SP) ⇨ M:M+1
STX	Store X in memory	(X) ⇨ M : M + 1
STY	Store Y in memory	(Y) ⇨ M : M + 1
	Transfer Instructions	
TFR	Transfer register to register	(A, B, CCR, D, X, Y, or SP) ⇨ A, B, CCR, D, X, Y, or SP
TSX	Transfer SP to X	(SP) ⇨ X
TSY	Transfer SP to Y	(SP) ⇨ Y
TXS	transfer X to SP	(X) ⇨ SP
TYS	transfer Y to SP	(Y) ⇨ SP
	Exchange Instructions	
EXG	Exchange register to register	(A, B, CCR, D, X, Y, or SP) ⇔ (A, B, CCR, D, X, Y, or SP)
XGDX	Exchange D with X	(D) ⇔ (X)
XGDY	Exchange D with Y	(D) ⇔ (Y)

Instruction Set Review

- **Stacking Instructions**

Mnemonic	Function	Operation
	Stack Pointer Instructions	
CPS	Compare SP to memory	(SP) – (M : M + 1)
DES	Decrement SP	(SP) – 1 ⇨ SP
INS	Increment SP	(SP) + 1 ⇨ SP
LDS	Load SP	(M : M + 1) ⇨ SP
LEAS	Load effective address into SP	Effective address ⇨ SP
STS	Store SP	(SP) ⇨ M : M + 1
TSX	Transfer SP to X	(SP) ⇨ X
TSY	Transfer SP to Y	(SP) ⇨ Y
TXS	Transfer X to SP	(X) ⇨ SP
TYS	Transfer Y to SP	(Y) ⇨ SP
	Stack Operation Instructions	
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	$M_{(SP)}$: $M_{(SP+1)} \Rightarrow Y$; $(SP) + 2 \Rightarrow SP$

Instruction Set Review

- **Pointer and Index Calculation Instructions**

Mnemonic	Function	Operation
LEAS	Load result of indexed addressing mode effective address calculation into stack pointer	r ± constant ⇨ SP or (r) + (accumulator) ⇨ SP r = X, Y, SP, or PC
LEAX	Load result of indexed addressing mode effective address calculation into x index register	r ± constant ⇨ X or (r) + (accumulator) ⇨ X r = X, Y, SP, or PC
LEAY	Load result of indexed addressing mode effective address calculation into y index register	r ± constant ⇨ Y or (r) + (accumulator) ⇨ Y r = X, Y, SP, or PC

- **Stop and Wait Instructions**

Mnemonic	Function	Operation
STOP	Stop	$SP - 2 \Rightarrow SP$; $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP$; $CCR \Rightarrow M_{(SP)}$ Stop CPU clocks
WAI	Wait for interrupt	$SP - 2 \Rightarrow SP$; $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP$; $CCR \Rightarrow M_{(SP)}$

Instruction Set Review

- **Condition Code Instructions**

Mnemonic	Function	Operation
ANDCC	Logical AND CCR with memory	(CCR) • (M) ⇒ CCR
CLC	Clear C bit	0 ⇒ C
CLI	Clear I bit	0 ⇒ I
CLV	Clear V bit	0 ⇒ V
ORCC	Logical OR CCR with memory	(CCR) + (M) ⇒ CCR
PSHC	Push CCR onto stack	(SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$
PULC	Pull CCR from stack	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP
SEC	Set C bit	1 ⇒ C
SEI	Set I bit	1 ⇒ I
SEV	Set V bit	1 ⇒ V
TAP	Transfer A to CCR	(A) ⇒ CCR
TPA	Transfer CCR to A	(CCR) ⇒ A

- **Background Mode Instructions**

Mnemonic	Function	Operation
BGND	Enter background debug mode	If BDM enabled, enter BDM; else resume normal processing
BRN	Branch never	Does not branch
LBRN	Long branch never	Does not branch
NOP	Null operation	—