Table of Contents

1.0 Overview

1.1 Introduction

Welcome to the Axiom Manufacturing Lab Manual for the CME-11E9-EVBU.

The CME-11E9-EVBU (EVBU) is a low-cost, entry-level development system for the MC68HC11E9 (HC11) microcontroller. The EVBU provides the HC11 single chip and expanded mode operation with the Buffalo Monitor Version 3.4 in the internal Read Only Memory (ROM). EVBU application development is not complicated, but students need basic knowledge of logic circuits, electronic theory, and Microsoft Windows to fully benefit from the manual.

The Axiom Manufacturing Lab Manual supplements textbooks and does not saturate the subject. The manual provides an understanding of the EVBU hardware and software needed to configure, program and debug embedded systems. This manual also provides lab experiments for the EVBU. The enclosed CD contains manuals for the HC11.

1.2 EVBU Features and Specifications

The EVBU features:

- HC11 Central Processing Unit (CPU)
- COM1 serial port-HC11 Serial Communication Interface (SCI) port to RS232 translator and DB9-S connector
- Three Input/Output (I/O) ports in all modes
  - Port A: 3 input, 3 output, 2 I/O. Timer module interface
  - Port D: 5 I/O. SCI and Serial Peripheral Interface (SPI) serial module interface
  - Port E: 8 inputs. Analog converter module interface
- Two additional I/O ports in single chip mode
  - Port B: 8 outputs. High order address in expanded mode
  - Port C: 8 I/O, latched port. Low order address and data in expanded mode
- Three memory sockets, 28-pin standard
  - U5: 32 Kbyte RAM
  - U6: No device installed default

  - U7: 8 Kbyte Electronically Erasable Programmable Read Only Memory (EEPROM)
- Input/Output Connectors
  - P4/EVBU Port: All HC11 I/O connections
  - Bus.Port: HC11 Expanded Mode address and data bus
  - MCU.Port: HC11 I/O ports available in all modes
  - LCD.Port: LCD module connection port, expanded mode operation
  - SS:Keypad port: Keypad connection port, 4 bits of port D and port E

Specifications

- Board size: 5.5” x 6.0”
- Power Input: +7 to +18VDC
- Current Consumption: 80ma Standard

2.0 Hardware

2.1 EVBU Hardware

The EVBU hardware processes, stores and communicates information to its connected system, i.e. a PC, LCD, keypad, etc. The EVBU hardware contains a microcontroller, memory and peripheral support systems. (See Section 1.2 EVBU Features and Specifications).

2.2 HC11 MCU

A high-density CMOS device makes up the HC11. The HC11 features:

- M68HC11 CPU
- Power-saving stop and wait modes
- 512 bytes of on-chip RAM, data retained during standby
- 12 Kbytes of on-chip ROM or Erasable Programmable Read Only Memory (EPROM)
- 512 bytes of on-chip EEPROM with erase or write protection
- Asynchronous non-return-to-zero (NRZ) SCI Serial Port
- Synchronous Serial Port (SPI)
- 8-channel, 8-bit A/D converter
- 16-bit timer system
- 8-bit pulse accumulator
- Real-time interrupt circuit
- Computer operating properly (COP) watchdog system
- 38 general-purpose input/output I/O pins

2.3 Interrupts and RESET

Interrupts allow the CPU real time process control or error handling services, and allows the CPU to respond to a hardware or physical event when it occurs. When an event occurs, interrupt stops the program flow and forces the CPU to respond.

The HC11 provides 21 possible interrupts, each associated with a dedicated position in the interrupt vector table ($FFD6-$FFFF). The CPU uses the interrupt vector table to find the correct service routine for the interrupt.

Each 16-bit entry contains the starting address for service. Proper operation calls for the complete interrupt vector table with unused interrupt vectors set to the RESET vector. This method provides an unscheduled interrupt recovery mechanism that would otherwise allow unspecified code execution.

Vector Address	Interrupt Source	CCR Mask Bit	Local Mask
FFCO, C1-FFD4, D5	Reserved	---	---
FFD6, D7	SCI Serial System • Receive data register • Receiver overrun • Transmit data register empty • Transmit complete • Idle line detect	I	 RIE RIE TIE TCIE ILIE
FFD8, D9	SPI serial transfer complete	I	SPIE
FFDA, DB	Pulse accumulator input edge	I	PAII
FFDC, DD	Pulse accumulator overflow	I	PAOVI
FFDE, DF	Timer overflow	I	TOI
FFE0, E1	Timer input capture 4/output compare 5	I	I4/O5I
FFE2, E3	Timer output compare 4	I	OC41
FFE4, E5	Timer output compare 3	I	OC3I
FFE6, E7	Timer output compare 2	I	OC2I
FFE8, E9	Timer output compare 1	I	OC1I
FFEA, EB	Timer input capture 3	I	IC3I
FFEC, ED	Timer input capture 2	I	IC2I
FFEE, EF	Timer input capture 1	I	IC1I
FFF0, F1	Real time interrupt	I	RTII
FFF2, F3	IRQ (External pin)	I	None
FFF4, F5	XIRQ pin	X	None
FFF6, F7	Software interrupt	None	None
FFF8, F9	Illegal opcode trap	None	None
FFFA, FB	COP failure	None	NOCOP
FFFC, FD	Clock monitor fail	None	CME
FFFE, FF	RESET	None	None

Figure 2.3 Interrupt Vector Table

2.3.1 Nonmaskable Interrupts

Nonmaskable interrupts are interrupts that are always enabled or that cannot be disabled after being enabled. Nonmaskable interrupts have the highest priority, and always service RESET first. Nonmaskable interrupts occur once enabled and cannot be disabled by software control. The

nonmaskable interrupts are XIRQ, SWI, TRAP, COP, CLOCK FAIL, and RESET.

2.3.2 Maskable Interrupts

When necessary the HC11 internal peripherals use the maskable interrupt. Maskable interrupts may prioritize events. As a result if two or more interrupts occur simultaneously, the programmer can prioritize service. Maskable interrupts may enable or disable under software control, so unnecessary service can discontinue.

Special instructions globally enable or disable maskable interrupts. The instructions include:

- Set Interrupt Mask Bit (SEI)
- Clear Interrupt Mask Bit (CLI)
- Return from Interrupt (RTI)

2.3.2.1 SEI

The SEI instruction disables all maskable interrupts by setting the I bit in the Condition Code Register (CCR). (See Section 2.4 HC11 Registers.) If the I bit sets, maskable interrupts do not occur or cause CPU action.

2.3.2.2 CLI

The CLI instruction enables all maskable interrupts that are enabled by their associated peripheral interrupt mask bit. Maskable interrupts only occur when the CCR I bit is clear.

2.3.2.3 RTI

The special case RTI instruction is the last instruction an interrupt service must execute. After servicing an interrupt, execute RTI instructions. The RTI instruction enables interrupts while the CPU

returns to normal program flow after service. Registers return to their former values. The I bit returns to 0.

2.4 HC11 Register

The HC11 CPU contains:

- Two 8-bit (byte size) A and B accumulators or 16-bit Accumulator D
- Index Registers X and Y: 16-bit
- Stack Pointer: 16-bit
- Program Counter: 16-bit
- Condition Code Register, 8-bit, with five codes: C, V, Z, N, and H.
- Interrupt Masking bits: IRQ and XIRQ
- Stop Disable: S

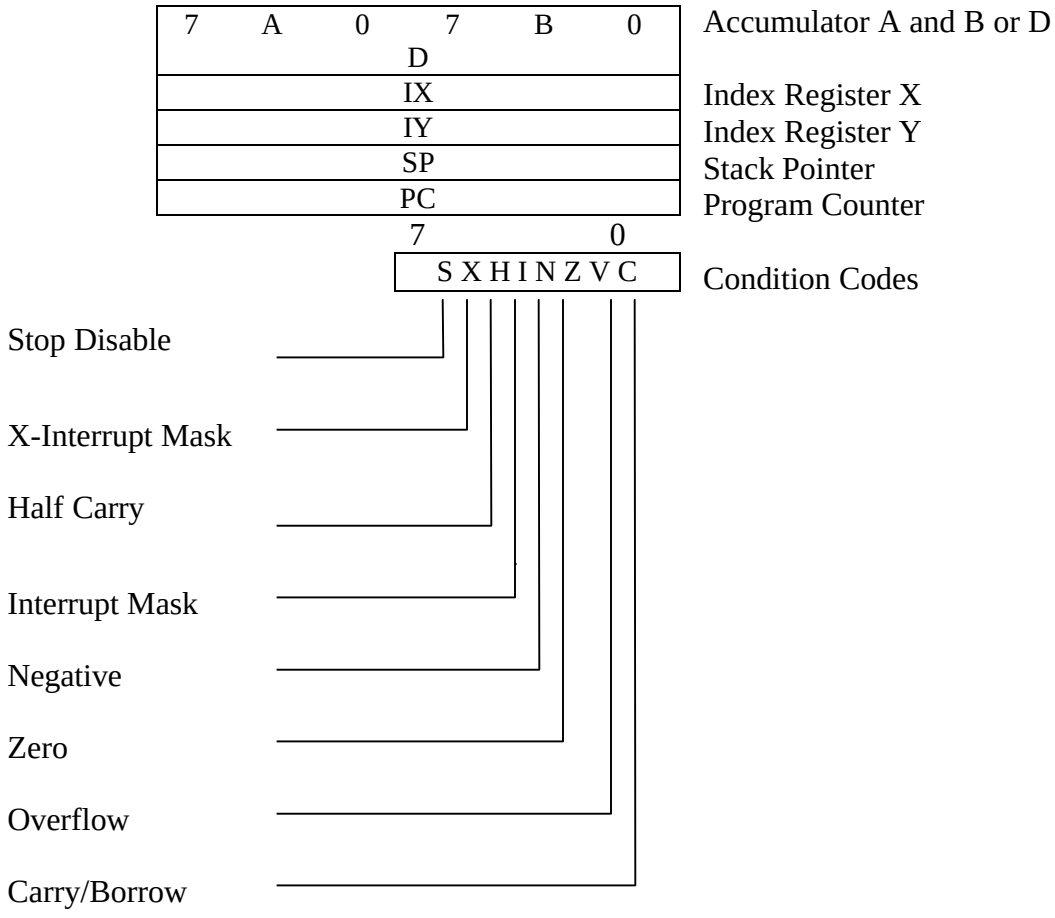

Figure 2.4 HC11 Register

2.4.1 Accumulators A, B, and D

The general-purpose accumulators A and B hold operands and results of arithmetic calculations and data manipulations. In some instances the accumulators work as a single double-byte Accumulator D. Most operations can use accumulator A and B interchangeably.

The exceptions are:

- The Add Accumulator B to X (ABX) and Add Accumulator B to Y (ABY) instructions add the contents of accumulator B to Index register X or Y.
- No Transfer A to CCR (TAP) or Transfer CCR to A (TPA) instructions exist to transfer data from between accumulator B and the condition code register.
- Unlike in accumulator A, there is no decimal adjustment accumulator (DAA) instruction in accumulator B for binary-coded decimal (BCD) arithmetic operations.
- Add, subtract and compare instructions involving both accumulator A and B operate in only one direction.

2.4.2 Index Register X (IX)

The 16-bit IX register supplies a 16-bit indexing value that can be added to the 8-bit offset in an instruction to create an operational address. The register may also be used as a counter or for temporary storage.

2.4.3 Index Register Y (IY)

The 16-bit IY register supplies an indexing function similar to the IX register. The IY register requires an extra machine code byte and an additional execution time cycle.

2.4.4 Stack Pointer (SP)

The stack pointer provides the CPU with temporary data memory space. It is a critical register for CPU operation and must be located with a valid RAM type memory address. When a byte enters the stack, the SP decreases. When a byte exits the stack, the SP increases. The stack is

automatically applied by the CPU to store the program counter and/or registers for subroutines or interrupt services.

The *M68HC11E Family Technical Data Manual, Section 3.3.4 Stack Pointer (SP)*, located on the 68HC11 Development Board CD, provides more information about the stack pointer.

2.4.5 Program Counter (PC)

The program counter points to the next executable instruction. In normal modes a CPU RESET initializes the PC to the RESET vector provided at address $FFFE/F. Program counter operation will then provide the CPU with the location of the next instruction to be executed.

2.4.6 Condition Code Register (CCR)

The 8-bit condition code register contains five condition code indicators, two interrupt masking bits, and a stop disable bit.

2.4.6.1 Carry/Borrow (C)

The C bit indicates that an addition overflow or subtraction underflow occurred. The C bit also operates as an error flag during multiplication and division operations.

Shift and rotate instructions operate with and through the C bit to facilitate multiple word shift operations. By setting or clearing the C bit, the programmer controls the information into the operand. The arithmetic shift right (ASR) instruction maintains the most significant bit's (MSB) original value, which facilitates the two's-complement (signed) number manipulation.

2.4.6.2 Overflow (V)

The V bit sets only if a two's-complement overflow occurs. Otherwise V bit is cleared. The LDAA and STAA instructions automatically set or clear the V bit.

2.4.6.3 Zero (Z)

The Z bit sets when all bits of an arithmetic, logic or data operation result in 0. An accumulator instruction, such as CLRB, sets Z bit due to the fact that the contents are equal to zero. Otherwise Z bit is cleared. The instruction LDAA#$80 clears the Z bit due to the fact that contents are not equal to zero.

Compare instructions perform an internal implied subtraction, and condition codes reflect the subtraction results. The Increment Index Register (INX), Decrement Index Register X (DEX), Increment Index Register Y (INY), and Decrement Index Register (DEY) operations affect only the Z bit. Each operation causes the particular index register to increase or decrease by 1. When these operations occur only = and ≠ 0 may be determined.

2.4.6.4 Negative (N)

The N bit reflects the state of MSB. The N bit sets when an arithmetic, logic or data operation returns negative results, i.e. MSB=1. Assigning an often-tested register or memory location MSB as a flag bit provides a method to test the N bit.

The LDAA and STAA instructions automatically set or clear the N bit. The instruction LDAA#$80 sets N. Otherwise N is cleared. Accumulator code CLRB clears the N bit.

2.4.6.5 Interrupt Mask (I)

The global mask I bit disables maskable interrupt sources. When an interrupt occurs, the I bit automatically sets after the registers are stacked but before fetching the interrupt vector. While set, I bit interrupts can become pending and are remembered, but CPU operation continues uninterrupted until the I bit clears. After any RESET, the I bit sets by default. Software instruction clears the I bit.

2.4.6.6 Half Carry (H)

The H bit sets when a carry occurs between bits 3 and 4 of the logic arithmetic unit during an Add Memory (ADD), Add Accumulators (ABA), or Add With Carry (ADC) instruction. Otherwise the H bit is cleared. The H bit needs no set or clear instructions because only the Decimal Adjust A (DDA) instruction uses the H bit to adjust BCD add or subtract operations. The H bit is not used as a test condition.

2.4.6.7 X Interrupt Mask (X)

The X bit disables interrupts from XIRQ and is only set by hardware (RESET or XIRQ acknowledge). When an interrupt occurs, the X bit, along with the I bit, automatically set after the registers are stacked but before fetching the interrupt vector. The CPU operation continues until the X bit is cleared.

Only software instruction clears X, i.e. transfer accumulator A to the CCR TAP instruction, where the associated bit of A is 0; or RTI, where bit 6 of the value loaded into the CCR from the stack has cleared.

2.4.6.8 Stop Disable (S)

The S bit allows or disallows the STOP instruction. A STOP instruction stops the oscillator. Setting the S bit prevents a STOP instruction from implementing a low power stop condition in the HC11. While set, a STOP instruction is treated as a no-operation (NOP) instruction. Processing continues to the next operation.

Although there are no set or clear instructions for the S bit, the TAP instruction changes the S bit but reduces unwanted changes to S. The accumulator A value at the time of executing the TAP instruction determines whether S changes. RESET sets S.

2.5 Data Types

The M8HC11 CPU supports four data types:

- Bit data
- 8-bit and 16-bit signed and unsigned integers
- 16-bit unsigned fractions
- 16-bit addresses

2.6 Opcodes

Opcodes identify an instruction and its related addressing mode to the CPU. Each instruction generally provides several opcodes to access a range of addressing capabilities.

2.6.1 Prebytes

An additional byte, called a prebyte, expands the 256 8-bit binary number opcodes. A prebyte affects certain instructions, mostly associated with the Y register. When the prebyte is used, the prebyte precedes the opcode, and applies only to that opcode. Opcode map prebyte codes are $18 for page 2, $1A for page 3 and $CD for page 4.

2.7 Memory

All embedded systems are memory map dependent. The EVBU provides the default memory internally located in the HC11 and an additional external memory of 32K bytes of RAM and 8K bytes of EEPROM. The HC11 dictates memory type locations in the memory map. For example, RESET VECTOR is at the fixed address 0xFFFE, which is a primary HC11 memory requirement. As a result non-volatile memory is applied to the 0xFFFE address, so the RESET vector will be valid when the EVBU is powered on. The HC11 internal ROM or the external EEPROM in the U7 socket provides EVBU RESET vector.

2.7.1 Accumulator and Memory Instructions

Most instructions use two operands. The first operand is for the accumulator or index register. The second is normally for an addressing mode. (See Section 2.8, Addressing Modes.) The accumulator operands contain six subgroups:

- Loads, stores, and transfers
- Arithmetic operations
- Multiply and divide
- Logical operations
- Data testing and manipulation
- Shifts and rotates

2.7.1.1 Loads, Stores, and Transfers

Loads, stores, and transfers manipulated data from memory and peripherals to the CPU, or transfer CPU results to memory or I/O devices. See the *M68HC11 Reference Manual on the 68HC11 CD* for summary of store and transfer instructions.

2.7.1.2 Arithmetic Operations

- 8-bit and 16-bit operations are supported directly and extendable to support multiple-word operands.
- Two’s-complement (signed) and binary (unsigned) operations are supported directly.

- BCD arithmetic is supported by following normal instruction sequences using DAA instruction, restoring results to BCD format.
- Compare instruction perform subtract in the CPU to update condition code bits without altering either operand.
- All other operations automatically update the condition code bits.

See the *M68HC11 Reference Manual on the 68HC11 CD* for summary of arithmetic functions.

2.7.1.3 Multiply and Divide

- 8-bit by 8-bit multiply produces a 16-bit result.
- The integer divide (IDIV) 16-bit by 16-bit divide produces a 16-bit result with a 16-bit remainder.
- The factional divide (FDIV) divides a 16-bit numerator by a larger 16-bit denominator and produces a 16bit result (a binary weighted fraction between 0 and 0.99998) and a 16-bit remainder. The FDIV also resolves an IDIV or FDIV operation.

See the *M68HC11 Reference Manual on the 68HC11 CD* for summary of multiply and divide functions.

2.7.1.4 Logical Operations

Logical instructions perform the Boolean logical operations AND, inclusive or exclusive OR, and one's compliment.

See the *M68HC11 Reference Manual on the 68HC11 CD* for summary of logical operations.

2.7.1.5 Data Testing and Manipulation

This operand group operates on operands of a single bit or any bit combination within any byte in the 64-K memory space.

See the *M68HC11 Reference Manual on the 68HC11 CD* for summary of data testing and manipulation.

2.7.1.6 Shifts and Rotates

Shift and rotate functions involve the carry bit in the CCR in addition to the 8-bit or 16-bit operand in the instruction, permitting easy multiple-word operand extensions.

See the *M68HC11 Reference Manual on the 68HC11 CD* for summary of shifts and rotates.

2.8 Addressing Modes

Six addressing modes are provided by the HC11:

- Immediate
- Direct
- Extended
- Indexed
- Inherent
- Relative

2.8.1 Immediate

Immediate addressing initializes a register with an immediate provided value. The bytes immediately following the opcode (#) contain the value.

2.8.2 Direct

In the direct addressing mode, a single byte following the opcode contains the operand address's low-order byte. Assume $00 is the high-order byte. This mode allows quick access to the HC11 zero page RAM memory.

2.8.3 Extended

In the extended addressing mode, two bytes following the opcode byte contain the argument's effective address. This mode allows access to any location in the HC11 memory map.

2.8.4 Indexed

In the indexed addressing mode, the instruction contains an 8-bit unsigned offset. The unsigned offset is added to the value in the IX or IY, creating the effective address. This mode allows use of certain instructions on any memory location or provides table lookup operations.

2.8.5 Inherent

In the inherent addressing mode, the opcode contains all the information necessary for instruction execution. Operands use one or more CPU registers and are not drawn from memory. The addressing mode includes operations that use only index registers or accumulators, and control instructions with no arguments.

2.8.6 Relative

The relative addressing mode is used only with branch operations, which generate two machine bites, one for opcode and one for relative offset. If the branch operation is true, an 8-bit signed offset is added to the program counter's contents to form the effective address branch. If the relative branch operation is false, control proceeds to the next instruction.

2.9 Operating Modes

The HC11 provides two normal operating modes:

- Single Chip
- Expanded.

Two special modes are also provided:

- Bootstrap
- Test

The logic states of the MODA and MODB pins during RESET select one of the four modes by determining the logic state of RBOOT, SMOD, MOA, and IRV

control bits in the highest priority interrupt (HPRIO) register. The control bits configure the logic circuits in the HC11 hardware selection mode. After RESET the mode selection pins do not influence the MCU operating mode.

EVBU-MODA	EVBU-MODB	OPERATION MODE
ON	OFF	Single Chip
OFF	OFF	Expanded
ON	ON	Bootstrap
OFF	ON	Test

Figure 2.9 EVBU Operating Mode Selection at RESET

2.9.1 Single Chip Mode

In single chip mode the HC11 accesses only on-chip memory. The mode uses ports B and C and strobe pins A (STRA) and B (STRB) for general purpose input/output ports. On RESET the internal ROM is enabled to provide vectors and program memory.

2.9.2 Expanded Mode

The expanded mode allows access to external memory and peripherals. The mode provides 64-Kbyte of external address space by using I/O ports B and C for the address and data bus. The HC11's internal memories and registers take priority over the external memory.

The HC11 internal ROM may be enabled (default) or disabled by the ROMON bit in the CONFIG REGISTER for this mode. If ROMON is disabled, the external EEPROM (U7) must provide vectors and program memory.

2.9.3 Bootstrap Mode

The bootstrap mode is a special mode variation of the single-chip mode. Bootstrap mode allows special purpose programs to be entered into RAM.

When bootstrap mode is selected after RESET, a small bootstrap ROM, at address $BF00-$BFFF, contains the bootloader program and a special set of interrupt and RESET vectors at $BFC0-$BFFF. The bootstrap program initializes the SCI and allows program downloading to on-chip RAM. The downloaded program size can be as large as on-chip RAM size. The mode is applied by AxIDE utility operations.

2.9.4 Test Mode

The test mode, a special variation of the expanded mode, allows privileged access to internal resources. Access is available for the configuration (CONFIG) register, programming calibration data into EEPROM, and supporting emulation and debugging during development. This mode is normally reserved for factory use.

2.10 Input/Output (I/O) Ports

The HC11 contains five ports: A, B, C, D, and E. Each port has a shared function, and port pin functions are mode dependent.

Port	Shared Function
Port A	Timer
Port B	High-order address
Port C	Low-order address and data bus
Port D	Serial communications interface (SCI) Serial peripheral interface (SPI)
Port E	Analog to Digital (ADC) Converter

Figure 2.10 Ports and Shared Function

2.10.1 Port A

Port A shares functions with the timer system and contains:

- Three input-only pins
- Three output-only pins
- Two bi-directional I/O pins

The Port A address is $1000.

Input/Only		**Output-Only**		**Bi-directional**		
Read/ Write	**Alternate Function**	**Read/ Write**	**Alternate Function**	**Read/ Write**	**Alternate Function**	**Alternate Function**
PAO (0) I	IC3 (0) I	PA4 (4)	OC4 (4)	PA3 (3) I	OC5 (3) I	IC4 (3) I
PA1 (1) I	IC2 (1) I	PA5 (5)	OC3 (5)	PA7 (7) I	PA1 (7) I	
PA2 (2) I	IC1 (2) I	PA6 (6)	OC2 (6)			
			And/Or OC1		And/Or OC1	And/Or OC1

( ) = Bit I = Indeterminate after RESET

Figure 2.10.1 Port A Data Register (PORTA)

2.10.2 Port B

Port B contains eight parallel output-only pins in single chip and bootstrap modes. Simple and full handshake input and output functions are available in single-chip mode. In simple strobe mode, port B is a strobe output port. Port C is a latching input port, simultaneously. In expanded or test modes, port B pins are high-order address outputs. The address for port B is $1004 in single chip mode. Port B register is not in the memory map in expanded mode.

Single-Chip/Bootstrap	**Expanded/Test**	**RESET**
PB0	ADDR8	0
PB1	ADDR9	0
PB2	ADDR10	0
PB3	ADDR11	0
PB4	ADDR12	0
PB5	ADDR13	0
PB6	ADDR14	0
PB7	ADDR15	0

Figure 2.10.2 Port Data Register (PORTB)

2.10.3 Port C

Port C contains eight parallel I/O pins. In single-chip and bootstrap modes, port C pins RESET to high-impedance inputs. The port C address

is $1003. In expanded and special test modes, port C pins operate as multiplex address/data bus. The port C register address is treated as an external memory location.

Port C provides a latched input port in single chip mode. The handshake clearing mechanism uses the Port C Latched Register (PORTCL). Reads of this register return the last value latched into PORTCL and clear Strobe A flag (STAF) following a read of parallel input/output control (PIOC) with STAF set.

	Single-Chip/Bootstrap Address $1003	**Expanded/Special Test Address $1003**		**Latched Register Address $1005**	**Data Direction Register Address $1007**
Bit	**Read/Write**	**Read**	**Write**	**Read/Write**	**Read/Write**
0	PC0	ADDR0	DATA0	PLC0	DDRC0
1	PC1	ADDR1	DATA1	PLC1	DDRC1
2	PC2	ADDR2	DATA2	PLC2	DDRC2
3	PC3	ADDR3	DATA3	PLC3	DDRC3
4	PC4	ADDR4	DATA4	PLC4	DDRC4
5	PC5	ADDR5	DATA5	PLC5	DDRC5
6	PC6	ADDR6	DATA6	PLC6	DDRC6
7	PC7	ADDR7	DATA7	PLC7	DDRC7

Indeterminate after RESET

Figure 2.10.3 Port C Data, Latched, and Data Direction Register (PORTC, PORTCL, AND DDRC)

2.10.4 Port D

In all modes port D bits [0:5] can be used for general purpose I/O or with the SCI and SPI subsystems. During RESET port D pins PD[5:0] are configured as high-impedance inputs (DDRD bits cleared). The port D address is $1008.

Read/Write and Alternate function bits [0:5] are indeterminate after RESET. Data Register bits are 0 after RESET.

Bit	Read/Write Address $1008	Alternate Function	Data Direction Register Address $1009
0	PD0	SCI-RxD	DDRD0
1	PD1	SCI-TxD	DDRD1
2	PD2	SCI-MISO	DDRD2
3	PD3	SCI-MOSI	DDRD3
4	PD4	SCI-SCK	DDRD4
5	PD5	SCI-SS	DDRD5
6	0		Unimplemented
7	0		Unimplemented

Figure 2.10.4 Port D Data Register and Data Direction Register (PORTD AND DDRD)

2.10.5 Port E

Port E provides eight general-purpose inputs. The A/D converter system also uses port E. Do not read PORTE during the sample portion of an A/D conversion when some port E pins are simultaneously used for general-purpose input and others are used for A/D conversion. The port E address is $100A.

2.11 EVBU Ports and Connectors

The EVBU provides several ports and connectors for access to the HC11 I/O ports, interfaces, or user expansion. Many of the HC11 I/O ports are applied for more than one purpose. When applying the EVBU ports and connectors, the user should be aware of HC11 mode of operation and duplicate connections to different connectors. In simple terms—the MCU ports and bus port provide available expanded mode access points. The P4/ EVBU connector provides available single chip mode or all HC11 I/O access points.

2.11.1 COM1 Serial Port

The onboard COM1 serial port is a simple three-wire asynchronous serial interface with hard-wired Clear-to-Send (CTS) and Data Terminal (DTR). The HC11 SCI port pins PD0 and PD1 drive COM1. The two logic

signals couple to the COM1 connector through an RS232 level shifter. COM1 is set to connect to a PC serial port with straight-through cable.

Cut-away jumpers are between the following pins:

- 4 ▶ 1 and 6 (DTR/Read Data Setup Line (DSR)/DCD)
- 7 ▶ 8 (RTS/CTS)

		5	GND
9			
		4	
8			
		3	RXD
7			
		2	TXD
6			
		1	

Figure 2.11.1 COM1 DB9S Style Connector

2.11.2 SS:Keypad Port

The 10-pin SS:Keypad connector implements 4 bits of port D and 4 bits of port E as a simple serial or keypad interface. The interface connects to the SPI port on port D as a simple interface or may be implemented as a software key scan for a passive keypad. The SS:Keypad can provide interface for one SPI serial peripheral device. Power pins 1 and 2 are not normally installed.

1	2	3	4	5	6	7	8	9	10
	o	o	o	o	o	o	o	o	o
+5	GND	D2	D3	D4	D5	E0	E1	E2	E3

Figure 2.11.2 SS:Keypad Port Pin Connector

2.11.3 MCU Port

The MCU 26-pin port connector is a dual-row 13-pin one-inch grid pin connector, containing input and output only and input/output lines. All

MCU ports serve dual functions with CPU peripherals and are available in all modes.

In addition to the SCI using PD0 and PD1 to implement COM1, the HC11 SPI I/O PD[2:5] are applied to implement the SS:Keypad port. When parallel I/O uses these port D lines, the lines are unavailable for COM1 or SS:Keyport ports. Similarly the HC11 port E 0-3 lines are applied to the SS:Keyport port. If a keypad is applied, those lines cannot be used for any other purpose.

Signal	Pin	Pin	Signal
PD0/RXD	1	2	PD1/TXD
PD2/SI	3	4	PD3/SO
PD4/SCLK	5	6	PD5/SELD
PA0/IC3	7	8	PA1/IC2
PA2/IC1	9	10	PA3/OC5/IC4
PA4/OC4	11	12	PA5/OC3
PA6/OC2	13	14	PA7/PA/OC1
PE7/AN7	15	16	PE3/AN3
PE6/AN6	17	18	PE2/AN2
PE5/AN5	19	20	PE1/AN1
PE4/AN4	21	22	PE0/AN0
VRL	23	24	VRH
GND	25	26	+5V

Figure 2.11.3 MCU Port Connector

2.11.4 Bus Port

The bus port supports off-board parallel type peripheral devices. The 40-pin connector brings out power (+5V), ground, as well as address, data, and control lines. The bus port is only operational in expanded mode.

Gnd	1	2	D3
D2	3	4	D4
D1	5	6	D5
D0	7	8	D6
A0	9	10	D7
A1	11	12	A2
A10	13	14	A3
/OE	15	16	A4
A11	17	18	A5
A9	19	20	A6
A8	21	22	A7
A12	23	24	A13
/WR	25	26	/CS0
/CS1	27	28	/CS2
/CS3	29	30	/CS4
/CS5	31	32	/IRQ
+5V	33	34	/M2
R/W	35	36	/CS6
ECLK	37	38	/CS7
Gnd	39	40	/RESET

Figure 2.11.4 Bus_Port

2.11.5 LCD Port

The LCD display interface connects to the data bus. The LCD interface supports most standard character LCD modules up to 80 characters and provides the most common pin-out. The LCD port provides power, ground and Vee, as well as the control signals. Addresses $B5F0 and $B5F1 are the command register and data register, respectively, for the LCD module attached.

The LCD Port is configured for direct cable connection to a LCD Module with a 2 x 7 pin header on the backside of the module, opposite display. Modules with single row connectors will require even and odd pin swapping into the cable; i.e. 1-2 swap, 3-4 swap, 5-6 swap, 7-8 swap, 9-10 swap, 11-12 swap, 13-14 swap.

Lines	Pin	Pin	Lines
+5V	2	1	GND
A0	4	3	Vee
/LCDCS	6	5	/RW
D1	8	7	D0
D3	10	9	D2
D5	12	11	D4
D7	14	13	D6

Figure 2.11.5a LCD Port

2.11.6 P4/EVBU Connector

The P4/EVBU connector duplicates the original Motorola EVBU board I/O port. The connector provides access to all of the HC11 I/O ports. The HC11 operating mode will affect availability of HC11 ports B and C as input or output ports.

GND	O	O	GND
VCC	O	O	VCC
SPARE	O	O	SPARE
SPARE	O	O	SPARE
VRH	O	O	VRL
PE7	O	O	PE3
PE6	O	O	PE2
PE5	O	O	PE1
PE4	O	O	PE0
PB0/A8	O	O	PB1/A9
PB2/A10	O	O	PB3/A11
PB4/A12	O	O	PB5/A13
PA6/A14	O	O	PB7/A15
PA0/IC3	O	O	PA1/IC2
PA2/IC1	O	O	PA3/OC5/IC4
PA4/OC4	O	O	PA5/OC3
PA6/OC2	O	O	PA7/OC1
NC	O	O	PD5/SS*
PD4/SCK	O	O	PD3/MOSI
PD2/MISO	O	O	PD1/TXD
PD0/RXD	O	O	IRQ*
XIRQ*	O	O	RESET*
PC7/AD7	O	O	PC6/AD6
PC5/AD5	O	O	PC4/AD4
PC3/AD3	O	O	PC2/AD2
PC1/AD1	O	O	PC0/AD0
XTAL	O		EXTAL
STRB/R/W*	O	O	E
STRA/AS	O	O	MODA/LTR*
MODB/VSTBY	O		GND

P4

Figure 2.11.6 EVBU Connector

2.12 Expanded Bus

In expanded mode the HC11 provides an expanded address and data bus on I/O ports B and C. Port C's expanded bus operation provides a multiplex low-order address and data bus. The EVBU uses 74HC573-type logic (U3) to latch to the

low-order address. The bus provides access for the U5, U6 and U7 memory sockets; LCD port; and chip selects CS0-CS7.

The expanded bus provides 64K bytes memory capability to the HC11. The HC11 internal memories may be used or disabled for expanded bus operation. When enabled, HC11 internal space takes priority over external memory or devices. External devices will not appear in the memory map where internal memory or registers exist. However, internal writes performed by HC11 will appear on the external bus.

2.12.1 Memory Map Logic

A 16V8 type programmable logic device (PLD), U2, provides memory mapping on the EVBU. The logic device decodes input signals from address, R/W and E clock to provide M1, M2, M3, OE, WR, and P control output signals.

M1, M2 and M3 provide memory selects to U5, U6 and U7 memory sockets, respectfully. The OE and WR signals provide the valid data bus with direction strobes to peripheral and memory devices attached to the expanded bus. The P signal provides peripheral chip select CS0-CS7 address range. The EVBU REV.D provides a MEM_EN option to disable all external chip selects, allowing true single-chip mode port C operation, without conflicting with an installed memory device.

Chip	Range	Memory Select
U5	$0000-7FFF	M1
U6	$8000-CFFF	M2
U7	$E000-FFFF	M3
CS0-7	$B580-B5FF	P

Figure 2.12.1 Memory Map Logic

2.12.2 LCD Logic Control

The HC259 (U4) device provides the LCD enable on the EVBU. The LCD enable applies CS7, A2, A3 and the E clock. The LCD is accessed from address $B5F0-B5F1. Address $B5F0 provides LCD control command writes or cursor position reads. Address $B5F1 provides display data transfer to and from the LCD display.

2.12.3 Chip Select

The M1 (U5), M2 (U6) and M3 (U7) memory chip selects and the CS0-CS7 peripheral chip selects provide access signals to the expanded bus memory or peripherals. All chip selects are active logic low.

The M1-M3 chip selects are dedicated to the U5-7 memory sockets. If the U6 memory socket is not populated with a memory device, M2 chip select signal may be used for an off-board memory chip select via the bus port.

CS0-CS7 chip selects are available for connecting peripheral devices off-board. CS7 is decoded for LCD port access. When using the LCD port, do not apply CS7 to peripherals. Chip selects access up to 16 bytes, but chip select application does not require a peripheral to provide all 16 bytes.

The EVBU board provides a SYNC option jumper associated with the CS0-CS7 chip selects. This option effects all CS0-7 chip selects. The SYNC option open allows address valid decoding to the chip selects, so the chip selects will be valid for the duration of the valid address. This should be applied in conjunction with the OE and WR signals to control data movement to the associated peripheral. SYNC option installed will modify chip select timing to be valid with valid data. This is useful for applying logic latches as output or input data ports.

2.13 Miscellaneous Circuits

2.13.1 RS232 Level Translator

P_COM1 accomplishes EVBU RS232 communication. The HC11 SCI serial IN RxD and OUT TxD operate at logic levels and must be translated to RS232 levels to communicate with a PC. U8 translates the data on P_COM1 to the 0-5V level required by the HC11.

2.13.2 VPP Connector

The VPP +/- pins can apply a 12V DC voltage to program a 68HC711 with on-chip EPROM if installed in the EVBU U1 socket. The HC11 installed at the factory does not provide EPROM memory.

2.13.3 PWR Terminal Block

The PWR terminal block's three-connection points allow alternate power input and also output voltage supply. The +V and GND connections may be used to input power to the EVBU or to access the voltage from the wall plug. The +5V will provide up to 50 ma for user circuits.

Point	Purpose
+V	+7 to 18V DC unregulated input or output
GND	Ground
+5V	+5V Input or Output

Figure 2.13.2 PWR Terminal Block Connection Point

2.14 Option Jumpers

Memory selection jumpers are two-pin and installed vertically.

2.14.1 Programming Enable Jumpers

MODA, MODB and WRITE_EN jumpers enable the programming of the EVBU external EEPROM using the utility software. Remove the

WRITE_EN jumper before RESET or removing power to guarantee program retention.

2.14.2 TRACE/PROG Jumper

The Buffalo Monitor Trace and Single Step functions are enabled by installing the TRACE jumper (Position 2-3 of the 3-position jumper). When installing the jumper, do not make other connections to the HC11 XIRQ or Port A3 I/O pins.

The PROG position (1-2) connects VPP+ to the XIRQ port line for programming a 68HC711 device's internal EPROM installed in U1. Install only the PROG jumper and apply VPP+ during programming operation.

2.14.3 SYNC Jumper

The SYNC jumper changes the CS0-7 timing. When open, the chip selects remain active for the entire bus access cycle of the connected peripheral. If the SYNC jumper is installed, the chip select cycle is timed to be valid when data on the data bus is valid. Timing required by the connected peripheral should be referred to when setting this option.

2.14.4 MEM-EN Jumper

Install the MEM-EN jumper during the Expanded Mode operation for access to the external memory devices. Remove the MEM-EN jumper for true Single-chip Mode operation to prevent memory data conflicts with port C when using port C strobes.

2.15 Jumper Settings Alteration

Adding or modifying factory installed memory settings requires the following changes.

2.15.1 JP3 – U5 Device Configuration

If a 32K device is installed, set to ON.

If an 8K device is installed:

- OFF configures 8K from 0000-1FFF and mirrors at 2000-3FFF, 4000-5FFF, and 6000-7FFF. Note that internal CPU register and RAM writes will modify the external memory.
- ON configures 8K from 2000-3FFF HEX and mirrors at 6000-7FFF, the recommended Buffalo Monitor and Small C operation position. No internal writes will corrupt the external memory.

2.15.2 JP4-JP6 – U6 Device Configuration

When OFF the JP6 write-protects the U6 device.

Configuration	JP4	JP5	JP7	JP6
8K RAM or EEPROM	OFF	OFF	OFF	ON
8K EPROM	OFF	OFF	OFF	OFF
32K EPROM	OFF	ON	ON	OFF
32K RAM or EEPROM	ON	OFF	ON	ON

Figure 2.15.2 U6 Device Configuration

2.15.3 JP8, JP9, WRITE_EN – U7 Device Configuration

Configuration	JP8	JP9	WRITE_EN
8K EEPROM	OFF	OFF	ON
8K EEPROM	OFF	OFF	OFF
32K EPROM	OFF	NO	OFF
32K EEPROM	ON	OFF	ON

Figure 2.15.3 U7 Device Configuration

3.0 Software

The AxIDE terminal program and the Buffalo Monitor assist in programming and debugging the HC11. The AxIDE provides an interface to the EVBU.

3.1 AxIDE

The AxIDE program is exclusive to the Axiom EVBU. AxIDE interfaces with the EVBU to make building, uploading and programming easier. The other AxIDE functions configure AxIDE and the processor. To configure the AxIDE:

- Open AxIDE from Programs on the Start Menu.
- Click the checkmark in the upper left corner of the program.
- Ensure the port settings are set to:

Port	COM1/COM2	Handshaking	
Baud Rate	9600		
Parity	None	Xon/Xoff	OFF
Data Bits	8	Rts/Cts	OFF
Stop Bits	1	Dtr/Dsr	OFF

Use a COM port not used by other resource. Check the Windows Device Manager.

- Click OK
- Choose CME11E9-EVBU on the Tool Bar drop down menu.
- Ensure the AxIDE is properly configured.
- Connect the DB9 serial cable to EVBU COM1 port.
- Apply power to the board with the 9V power supply
- The Buffalo Monitor prompt appears:

BUFFALO 3.4 (ext.) – Bit User Fast Friendly Aid to Logical Operation

>_

If the prompt does not appear, consult the EVBU Users Manual: Troubleshooting.

3.1.1 Upload

The AxIDE upload feature sends the serial port a text file. When applied with the Buffalo LOAD T command, a program S-Record can be loaded for test and debug.

3.1.2 Build

The Build feature takes assembly source code and automatically compiles it to machine code. The compiled code is placed in a S-Record file (.S19) and a listing file (.LST) is also generated. The file extension S19 is unique to Motorola microcontrollers. Note that DOS compatible path and filenames (8 characters max) should be used.

3.1.3 Configure

The Configure feature sets the HC11 configure register. The configure registers stores bits that set the on-chip EEPROM, on-chip ROM, Watchdog system and security. By default the ROMON bit is enabled and the Buffalo program is provided from the HC11 internal ROM. Disabling the ROMON bit allows the HC11 to execute a program stored in the U7 memory device at RESET. Other CONFIG bits should not be modified.

3.1.4 Program

The Program feature writes to either internal or external programmable memory. Use only S19 extension files with this feature.

3.1.5 Read

The Read feature reads the specified memory location contents with a starting and ending range. View the information via the monitor or saved in a file. Format the address with a HEX number proceeding 0x, i.e. 0x1FFF.

3.2 Buffalo Monitor

The EVBU firmware based support program Buffalo Monitor provides a self-contained development environment. The development environment provides method for testing, debugging software, and software/hardware applications. The monitor interacts with the user by predefined commands entered via the terminal.

3.2.1 Command Format

The following notes apply to entering Buffalo commands.

- Number space, commas and tab characters separates fields.
- Input numbers are interpreted as hexadecimal.
- Enter input commands as upper or lower case. Input commands convert to upper case, except for downloading commands to the host computer or when operating in the transparent mode.
- Command lines maximum equals 35 characters. When the 36th character is added, the monitor automatically terminates the command entry. The terminal CRT displays: “Too long.”
- Correct command line errors by backspacing (CTRL-H) or by aborting the command (CTRL-X or DELETE).
- Repeat commands by pressing [RETURN].

Character	Meaning
<>	Enclose syntactical variable
[]	Enclose optional fields
[]…	Enclose optional fields repeated

Figure 3.2.1.a Character and Syntactical Meaning

Buffalo Command	Result
ASM [<address>]	Assembler/disassembler
BF <addr1> <addr2> <data>	Block fill memory with data
BR [-] [<address>]	Breakpoint set
BULK	Bulk Erase EEPROM
BULKALL	Bulk Erase EEPROM+CONFIG register
CALL [<address>]	Execute subroutine
G [<address>]	Execute program
HELP	Display monitor command
LOAD <host download command>	Download (S-Records*) via host port
LOAD T	Download (S-Records*) via terminal port
MD [addr1>[<addr2>]]	Dump memory to terminal
MM [<address>]	Memory modify
MOVE <addr1><addr2>[,dest>]	Move memory to new location
P	Proceed/continue from breakpoint
RM [p, y, x, a, b, c, s,]	Register modify
T [<n>]	Trace $1-$FF instructions
TM	Enter transparent mode
VERIFY<host download command>	Compare memory to download via host port
VERIFY <T>	Compare memory to download vial terminal

Figure 3.2.1.b Buffalo Monitor Command

The command line format is:

><command>[<parameters>](ENTER)

where:

Prompt	Result
<command>	Command mnemonic (single letter for most commands)
<parameters>	Expression or address
(ENTER)	ENTER keyboard key – depressed to ENTER command

Figure 3.2.1.c Command Line Format

3.3 Source Code

Write program source code in ASC11 text with any file editor that will provide text only output. Use the DOS EDIT or Windows NOTEPAD programs for compatibility. After writing and saving the program source code, assemble or

compile code to the Motorola S-Record format with the AxIDE Build feature. The output file typically has a .S19 file extension. The Buffalo LOAD T command prepares the Buffalo Monitor for S-Record upload. The AxIDE Upload function sends the S-Record file to the Buffalo for placing the program into memory.

3.3.1 Debug Mode

In the debug mode, locate your CODE in external RAM. After debugging, locate the code in EEPROM for dedicated operation without Buffalo supervision. CODE may be located in the internal or external memory. Typically code is originated at address $2000 for debug and $E000 for dedicated operation.

3.3.2 DATA

Start DATA and variables in an unused RAM location. After debugging, DATA may remain at the current location or moved to internal RAM starting at $0000. Note that Buffalo Monitor applies RAM from $40-$FF for its use.

3.3.3 Stack Register

Program the Stack Register somewhere near the top of the available RAM. Buffalo provides some stack space, but if C programs are applied, the stack will need to be moved to increase space. Typically the stack will be placed at the top of internal RAM, $01FF.

3.4 Assembly Language

Assembly language programs are low level and directly interact with hardware, making it the preferred EVBU programming method. Basic instructions correlate logical 1 and 0 to +5V high or 0V low on I/O ports.

Assembly language provides readable mnemonic symbols for the CPU. Each opcode represents an operation and an addressing mode. The default assembler applied by AxIDE Build operation is the AS11.exe program. The AS11.exe assembler is a two pass assembler that converts the input file mnemonic source statements into a object code S-Record file and listing file. The listing file provides the physical address and opcodes associated with the input source code statements. All files are ASCII text type and can be viewed or printed with Editor programs such as Notepad or Wordpad.

Reference the 68HC11 Reference Manual for information on the mnemonic commands and operations.

3.5 Freeware Assembler

3.5.1 Introduction

Programs written in assembly language consist of a sequence of source statements. Each source statement consists of a sequence of ASCII characters ending with a carriage return.

3.5.2 Source Statement Format

Each source statement may include up to four fields: a label (or "*" for a comment line), an operation (instruction mnemonic or assembler directive), an operand, or a comment.

3.5.2.1 Label Field

The label field occurs as the first field of a source statement. The label field can take one of the following forms:

1. An asterisk (*) as the first character in the label field indicates that the rest of the source statement is a comment. Comments are ignored by the Assembler and are printed on the source listing only for the programmer's information.

2. A white space character (blank or tab) as the first character indicates that the label field is empty. The line has no label and is not a comment.
3. A symbol character as the first character indicates that the line has a label. Symbol characters are the upper or lower case letters a-z, digits 0-9, and the special characters, period (.), dollar sign ($), and underscore (_). Symbols consist of one to 15 characters, the first of which must be alphabetic or the special characters (.) or underscore (_). All characters are significant, and upper and lower case letters are distinct.

A symbol may occur only once in the label field. If a symbol does occur more than once in the label field, then each reference to that symbol will be flagged with an error.

With the exception of some directives, a label is assigned the value of the program counter of the first byte of the instruction or data being assembled. The value assigned to the label is absolute. Labels may optionally be ended with a colon (:). If the colon is used, it is not part of the label but merely acts to set the label off from the rest of the source line. Thus the following code fragments are equivalent:

```
here:   deca
        bne here
here    deca
        bne here
```

A label may appear on a line by itself. The assembler interprets this as set the value of the label equal to the current value of the program counter.

The symbol table has room for at least 2000 symbols of length eight characters or less. Additional characters up to 15 are permissible at the expense of decreasing the maximum number of symbols possible in the table.

3.5.2.2 Operation Field

The operation field occurs after the label field and must be preceded by at least one white space character. The operation field must contain a legal opcode mnemonic or an assembler directive. Upper case characters in this field are converted to lower case before being checked as a legal mnemonic. Thus ‘nop’, ‘NOP’, or ‘NoP’ are recognized as the same mnemonic. Entries in the operation field may be one of two types:

Opcode: These correspond directly to the machine instructions. The operation code includes any register name associated with the instruction. These register names not must be separated from the opcode with any white space characters. Thus ‘clra’ means clear accumulator A, but ‘clr a’ means clear memory location identified by the label ‘a’.

Directive: These are special operation codes known to the Assembler which control the assembly process rather than being translated into machine instructions.

3.5.2.3 Operand Field

The operand field’s interpretation is dependent on the contents of the operation field. The operand field, if required, must follow the operation field and must be preceded by at least one white space character. The operand field may contain a symbol, an expression,

or a combination of symbols and expressions separated by commas.

The operand field of machine instructions is used to specify the addressing mode of the instruction, as well as the operand of the instruction. The following tables summarize the operand field formats for the various processor families.

NOTE: In these tables, parenthesis "()" signify optional elements, and angle backets "<>" denote an expression is inserted. These syntax elements are present only for clarification of the format and are not inserted as part of the actual source program. All other characters are significant and must be used when required.

3.5.2.4 M68HC11 Operand Syntax

For the M68HC11, the following operand formats exist:

Operand Format	M68HC11 Addressing Mode
No operand	Accumulator and inherent
<expression>	Direct, extended, or relative
#<expression>	Immediate
<expression>, X	Indexed with X register
<expression>, Y	Indexed with Y register
<expression> <expression>	Bit set or clear
<expression> <expression> <expression>	Bit test and branch

Figure 3.5.2.4 M68HC11 Operand Formats

The bit manipulation instruction operands are separated by spaces in this case since the HC11 allows bit manipulation instructions on indexed addresses. Thus a ',X' or ',Y' may be added to the final

two formats above to form the indexed effective address calculation.

3.5.2.5 Expressions

An expression is a combination of symbols, constants, algebraic operators, and parentheses. The expression is used to specify a value which is to be used as an operand.

Expressions may consist of symbols, constants, or the character '*' (denoting the current value of the program counter) joined together by one of the operators: + - * / % & | ^ .

3.5.2.6 Operators

The operators are the same as in c:

+	Add
-	Subtract
*	Multiply
/	Divide
%	Remainder after division
&	Bitwise and
|	Bitwise or
^	Bitwise exclusive or

Expressions are evaluated left to right, and there is no provision for parenthesized expressions. Arithmetic is carried out in signed two's-compliment integer precision (that's 16 bits on the IBM PC).

3.5.2.7 Symbols

Each symbol is associated with a 16-bit integer value, which is used in place of the symbol during the expression evaluation. The asterisk (*) used in an expression as a symbol represents the

current value the location counter (the first byte of a multi-byte instruction).

3.5.2.8 Constants

Constants represent quantities of data that do not vary in value during the execution of a program. Constants may be presented to the assembler in one of the five formats: decimal, hexadecimal, binary, octal, or ASCII. The programmer indicates the number format to the assembler with the following prefixes:

$	HEX
%	BINARY
@	OCTAL
‘	ASCII

Unprefixed constants are interpreted as decimal. The assembler converts all constants to binary machine code and is displayed in the assembly listing as hex.

A decimal constant consists of a string of numeric digits. The value of a decimal constant must fall in the range of 0-65535, inclusive. The following example shows both valid and invalid decimal constants:

VALID	INVALID	REASON INVALID
12	123456	More than 5 digits
12345	12.3	Invalid character

A hexadecimal constant consists of a maximum of four characters from the set of digits (0-9) and the upper case alphabetic (A-F), and is preceded by a dollar sign ($). Hexadecimal constants must

be in the range of $0000 to $FFFF. The following example shows both valid and invalid hexadecimal constants.

VALID	INVALID	REASON INVALID
$12	ABCD	No preceding "$"
$ABCD	$G2A	Invalid character
$001F	$2F018	Too many digits

A binary constant consists of a maximum of 16 ones or zeros preceded by a percent sign (%). The following example shows both valid and invalid binary constants:

VALID	INVALID	REASON INVALID
%00101	1010101	Missing percent
%1	%10011000101010111	Too many digits
%10100	%210101	Invalid digit

An octal constant consists of a maximum of six numeric digits, excluding the digits 8 and 9, preceded by a commercial at-sign (@). Octal constants must be in the ranges of @0 to @177777. The following example shows both valid and invalid octal constants.

VALID	INVALID	REASON INVALID
@17634	@2317234	Too many digits
@377	@277272	Out of range
@177600	@23914	Invalid character

A single ASCII character can be used as a constant in expressions. ASCII constants are preceded by a single quote (‘). Any character, including the single quote, can be used as a character constant.

The following example shows both valid and invalid character constants:

VALID	INVALID	REASON INVALID
'*	'VALID	Too long

For the invalid case above the assembler will not indicate an error. Rather it will assemble the first character and ignore the remainder.

3.5.2.9 Comment Field

The last field of an Assembler source statement is the comment field. This field is optional and is only printed on the source listing for documentation purposes. The comment field is separated from the operand field (or from the operation field if no operand is required) by at least one white space character. The comment field can contain any printable ASCII characters.

3.5.3 Assembler Directives

3.5.3.1 Introduction

The Assembler directives are instructions to the Assembler, rather than instructions to be directly translated into object code. This section describes the directives that are recognized by the Freeware Assemblers. Detailed descriptions of each directive are arranged alphabetically. The notations used in this chapter are:

( ) Parentheses denote an optional element.

XYZ The names of the directives are printed in capital letters.

< > The element names are printed in lower case and contained in angle brackets. All elements outside of the angle brackets '<>' must be specified as-is. For example, the syntactical element (<number>,) requires the comma to be

specified if the optional element <number> is selected. The following elements are used in the subsequent descriptions:

<comment>	A statement's comment field
<label>	A statement's label
<expression>	An Assembler's statement
<expr>	An Assembler's expression
<number>	A numeric constant
<string>	A string of ASCII
<delimiter>	A string delimiter
<option>	An Assembler option
<symbol>	An Assembler symbol
<sym>	An Assembler symbol
<sect>	A relocatable program section
<reg list>	M6809 register list
<reg exp>	M6809 register expression

In the following descriptions of the various directives, the syntax, or format, of the directive is given first. This will be followed with the directive's description.

3.5.3.2 BSZ – Block Storage of Zeros

(<label>) BSZ <expression> (<comment>)

The BSZ directive causes the Assembler to allocate a block of bytes. Each byte is assigned the initial value of zero. The number of bytes allocated is given by the expression in the operand field. If the expression contains symbols that are either undefined or forward referenced (i.e. the definition occurs later on in the file), or if the expression has a value of zero, an error will be generated.

3.5.3.3.EQU – Equate Symbol to a Value

<label> EQU <expression> (<comment>)

The EQU directive assigns the value of the expression in the operand field to the label. The EQU directive assigns a value other than the program counter to the label. The label cannot be redefined anywhere else in the program. The expression cannot contain any forward references or undefined symbols. Equates with forward references are flagged with Phasing Errors.

3.5.3.4 FCB – Form Constant Byte

(<label>) FCB <expr> (,<expr>…, <expr>) (<comment>)

The FCB directive may have one or more operands separated by commas. The value of each operand is truncated to eight bits and is stored in a single byte of the object program. Multiple operands are stored in successive bytes. The operand may be a numeric constant, a character constant, a symbol, or an expression. If multiple operands are present, one or more of them can be null (two adjacent commas), in which case a single byte of zero will be assigned for that operand. An error will occur if the upper eight bits of the evaluated operands' values are not all ones or all zeros.

3.5.3.5 FCC - Form Constant Character String

(<label>) FCC <delimiter> <string> <delimiter> (<comment>)

The FCC directive is used to store ASCII strings into consecutive bytes of memory. The byte storage begins at the current program counter. The label is assigned to the first byte in the string. Any of the printable ASCII characters can be contained in the string. The string is specified between two identical delimiters which can

be any printable ASCII character. The first non-blank character after the FCC directive is used as the delimiter.

Example:

```
LABEL1      FCC  , ABC,
```

Assembles ASCII ABC at location LABEL1

3.5.3.6 FDB – Form Double Byte Constant

(<label>) FDB <expr> (,<expr>,…,<expr> (<comment>)

The FDB directive may have one or more operands separated by commas. The 16-bit value corresponding to each operand is stored into two consecutive bytes of the object program. The storage begins at the current program counter. The label is assigned to the first 16-bit value. Multiple operands are stored in successive bytes. The operand may be a numeric constant, a character constant, a symbol, or an expression. If multiple operands are present, one or more of them can be null (two adjacent commas), in which case two bytes of zero will be assigned for the operand.

3.5.3.7 FILL – Fill Memory

(<label>) FILL <expression>, <expression>

The FILL directive causes the assembler to initialize an area of memory with a constant value. The first expression signifies the one byte value to be placed in the memory, and the second expression indicates the total number of successive bytes to be initialized. The first expression must evaluate to t he range of 0-255. Expressions cannot contain forward references or undefined symbols

3.5.3.8 OPT – Assembler Output Options

OPT <option> (,<option>,…<option>) (<comment>)

The OPT directive is used to control the format of the Assembler output. The options are specified in the operand field, separated by commas. All options have a default condition. Some options can be initialized from the command line that invoked the Assembler, however the options contained in the source file take precedence over any entered on the command line. In the following descriptions, the parenthetical inserts specify "DEFAULT", if the option is the default condition. All options must be entered in lower case.

c – Enable cycle counting in the listing. The total cycle count for that instruction will appear in the listing after the assemble bytes and before the source code.

cre – Print a cross reference table at the end of the source listing. This option, if used, must be specified before the first symbol in the source program is encountered.

1 – Print the listing from this point on.

noc – (DEFAULT) Disable cycle counting in the listing. If the "c" option was used previously in the program, this option will cause cycle counting to cease until the next "OPT c" statement.

no1 – (DEFAULT) Do not print the listing from this point on. An "OPT 1" can re-enable listing at a later point in the program.

s – Print symbol table at end of source listing.

3.5.3.9 ORG - Set Program Counter to Origin

The ORG directive changes the program counter to the value specified by the expression in the operand field. Subsequent statements are assembled into memory locations starting with the new program counter value. If no ORG directive is encountered in a source program, the program counter is initialized to zero. Expressions cannot contain forward references or undefined symbols.

3.5.3.10 PAGE – Top of Page

PAGE

The Page directive causes the Assembler to advance the paper to the top of the next page. If no source listing is being produced, the PAGE directive will have no effect. The directive is not printed on the source listing.

3.5.3.11 RMB – Reserve Memory Bytes

(<label>) RMB <expression> (<comment>)

The RMB directive causes the location counter to be advanced by the value of the expression in the operand field. This directive reserves a block of memory the length of which in bytes is equal to the value of the expression. The block of memory reserved is not initialized to any given value. The expression cannot contain any forward references or undefined symbols. The directive is commonly used to reserve a scratchpad or table area for later use.

3.5.3.12 ZMB – Zero Memory Bytes (same as BSZ)

(<label>) ZMB <expression> (<comment>)

The ZMB directive causes the Assembler to allocate a block of bytes. Each byte is assigned the initial value of zero. The number of bytes allocated is given by the expression in the operand field. If the expression contains symbols that are either undefined or forward references, or if the expression has value of zero, an error will be generated.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z