```
;================================================================================
; ADAM System Final Test Module Firmware
; MC68701 Microcontroller (2KB internal ROM at $F800-$FFFF)
;
; Dumped: 12/29/2025 by John Lundy
; Module supplied by: Rich DiRocco
; Checksum: $A38B (verified - matches label on chip)
;
; Disassembly with comments by Claude
; Using ADAM Technical Reference Manual for hardware context
;
; OVERVIEW:
; This is factory test firmware for testing the Master 6801 microcomputer
; on the ADAM Memory and I/O Board. The 6801 Master controls:
;   - ADAMnet (62.5 kbps half-duplex serial network)
;   - Keyboard, Printer, Tape Drives via ADAMnet
;   - Communication with Z80 via MIOC (Memory I/O Controller)
;
; The test module appears to verify:
;   - Internal RAM functionality ($0080-$00FF, $0400-$07FF)
;   - Port I/O operations
;   - Timer/counter functions
;   - Serial communication (likely ADAMnet timing)
;   - Memory pointer boundary conditions
;================================================================================

;--------------------------------------------------------------------------------
; MC68701 MEMORY MAP
;--------------------------------------------------------------------------------
; $0000-$001F  Internal I/O Registers
; $0080-$00FF  Internal RAM (128 bytes)
; $0100-$03FF  (External - depends on system)
; $0400-$07FF  External test RAM area
; $F800-$FFFF  Internal ROM (2KB) - This firmware
;--------------------------------------------------------------------------------

;--------------------------------------------------------------------------------
; MC68701 I/O REGISTER DEFINITIONS
;--------------------------------------------------------------------------------
DDR1    EQU  $00    ; Port 1 Data Direction Register
DDR2    EQU  $01    ; Port 2 Data Direction Register
PORT1   EQU  $02    ; Port 1 Data - directly controls test signals
PORT2   EQU  $03    ; Port 2 Data
DDR3    EQU  $04    ; Port 3 Data Direction Register
DDR4    EQU  $05    ; Port 4 Data Direction Register
PORT3   EQU  $06    ; Port 3 Data
```

```
PORT4       EQU     $07         ; Port 4 Data - used for status/sync detection
TCSR        EQU     $08         ; Timer Control/Status Register
COUNTER_H   EQU     $09         ; Free-running Counter High Byte
COUNTER_L   EQU     $0A         ; Free-running Counter Low Byte
OCR_H       EQU     $0B         ; Output Compare Register High
OCR_L       EQU     $0C         ; Output Compare Register Low
ICR_H       EQU     $0D         ; Input Capture Register High
ICR_L       EQU     $0E         ; Input Capture Register Low
SCI_RMCR    EQU     $10         ; SCI Rate/Mode Control Register
SCI_TRCSR   EQU     $11         ; SCI Transmit/Receive Control Status
SCI_RDR     EQU     $12         ; SCI Receive Data Register
SCI_TDR     EQU     $13         ; SCI Transmit Data Register
RAM_CTRL    EQU     $14         ; RAM Control Register

;--------------------------------------------------------------------------
; PORT BIT DEFINITIONS (based on test code analysis)
;--------------------------------------------------------------------------
; PORT1 ($02):
;   Bit 3 ($08) - Clock/sync signal toggle (XOR'd frequently)
;   Bits 4-5 ($30) - Status bits, checked for $30 pattern
;   Bits 6-7 ($C0) - Mode/control bits
;   Bit 2 ($04) - Secondary control bit
;
; PORT4 ($07):
;   Bits 3-5 ($38) - Sync detection bits
;   Bit 6 ($40) - Additional status
;   Bit 7 ($80) - High bit status
;--------------------------------------------------------------------------

;--------------------------------------------------------------------------
; RAM VARIABLE DEFINITIONS (Internal RAM $0080-$00FF)
;--------------------------------------------------------------------------
chksum_src_lo   EQU $80         ; Checksum source data low byte
chksum_src_hi   EQU $81         ; Checksum source data high byte
chksum_comp     EQU $82         ; Checksum comparison value
                                ; $83 - additional checksum byte
chksum_inv      EQU $84         ; Inverted checksum value
                                ; $85 - additional byte
base_addr       EQU $86         ; Base address for operations
                                ; $87 - high byte
calc_sum        EQU $88         ; Calculated checksum result
sum_accum       EQU $89         ; Running sum accumulator (16-bit, $89-$8A)
mem_ptr         EQU $8B         ; Memory pointer for boundary tests (16-bit, $8B-$8C)
bit_count       EQU $93         ; Bit counter for serial operations
end_ptr         EQU $94         ; End pointer for memory tests (16-bit, $94-$95)
temp_byte       EQU $96         ; Temporary storage byte
sync_byte       EQU $97         ; Sync pattern byte ($16 = SYN character)
state_var       EQU $98         ; State machine variable
```

```
status_flags    EQU $99         ; Combined status flags from ports
boundary_flags  EQU $9D         ; Memory boundary condition flags
                                ; Bit 0: Upper boundary reached ($0080)
                                ; Bit 2: Lower boundary reached ($FFFF)

;-------------------------------------------------------------------------
; CONSTANTS
;-------------------------------------------------------------------------
RAM_START       EQU $0080       ; Start of internal RAM
RAM_END         EQU $00FF       ; End of internal RAM (for clear loop)
EXT_RAM_START   EQU $0400       ; Start of external test RAM
EXT_RAM_END     EQU $0800       ; End of external test RAM
SYN_CHAR        EQU $16         ; ASCII SYN (synchronous idle) character
TIMER_DELAY     EQU $03E8       ; Timer delay constant (1000 decimal)

;=========================================================================================
; ROM START - $F800
; First 146 bytes ($F800-$F891) appear to be data tables, not code
;=========================================================================================

                ORG     $F800

;-------------------------------------------------------------------------
; DATA TABLE 1: Test Pattern Data ($F800-$F891)
; These appear to be test patterns for RAM/port verification
; Format seems to be groups of related bytes
;-------------------------------------------------------------------------
TEST_PATTERNS:
                FCB     $47,$57,$30,$00     ; "GW0" + null - possibly ID string
                FCB     $28,$18,$38,$00     ; Pattern group 1
                FCB     $00,$00,$30,$00
                FCB     $28,$18,$38,$00
                FCB     $00,$00,$30,$00
                FCB     $28,$40,$38,$00     ; Note $40 variant
                FCB     $08,$08,$30,$00
                FCB     $28,$10,$38,$00
                FCB     $08,$08,$30,$00
                FCB     $28,$18,$38,$00
                FCB     $08,$08,$30,$00
                FCB     $40,$40,$38,$00     ; Double $40 pattern
                FCB     $08,$08,$30,$00
                FCB     $28,$18,$38,$00
                FCB     $08,$08,$30,$00
                FCB     $28,$18,$38,$00
                FCB     $08,$08,$30,$00
                FCB     $28,$20,$38,$00
                FCB     $08,$08
```

```
;--------------------------------------------------------------------------
; DATA TABLE 2: Secondary Test Patterns ($F84A-$F891)
;--------------------------------------------------------------------------
            FCB     $0C,$00,$0A,$10,$0E,$00
            FCB     $00,$00,$0C,$00,$0A,$10,$0E,$00
            FCB     $00,$00,$0C,$00,$0A,$06,$0E,$00
            FCB     $02,$02,$0C,$00,$0A,$04,$0E,$02
            FCB     $02,$02,$0C,$00,$0A,$10,$0E,$02
            FCB     $02,$02,$0C,$00,$06,$06,$0E,$02
            FCB     $02,$02,$0C,$00,$0A,$10,$0E,$00
            FCB     $02,$02,$0C,$00,$0A,$10,$0E,$02
            FCB     $02,$02,$0C,$00,$0A,$08,$0E,$00
            FCB     $02,$02

;--------------------------------------------------------------------------
; JUMP TABLE: Subroutine Address Table ($F892-$F8A1)
; Used by test dispatcher - each entry is a 16-bit address
;--------------------------------------------------------------------------
JUMP_TABLE:
            FDB     $FCFB           ; Test 0: -> $FCFB (may be invalid/unused)
            FDB     $FA7A           ; Test 1: -> $FA7A
            FDB     $FB80           ; Test 2: -> $FB80 (CALC_SUM_DELAY)
            FDB     $FA10           ; Test 3: -> $FA10
            FDB     $FC2C           ; Test 4: -> $FC2C (likely SET_BIT_WAIT area)
            FDB     $FC16           ; Test 5: -> $FC16
            FDB     $FBBE           ; Test 6: -> $FBBE
            FDB     $FFFF           ; Test 7: End marker / unused

;--------------------------------------------------------------------------
; STATE TRANSITION TABLE ($F8A2-$F8AF)
; Maps current state to next state based on test results
;--------------------------------------------------------------------------
STATE_TABLE:
            FCB     $FF,$FF,$FF,$FF     ; Unused/terminal states
            FCB     $FF,$FF,$FF,$FF
            FCB     $FF,$FF,$FF,$FF
            FCB     $FF,$0F             ; Transition markers

;==========================================================================================
; RESET ENTRY POINT - $F8B1
; This is where execution begins after hardware reset
;==========================================================================================

            ORG     $F8B1

RESET_ENTRY:
            LDS     #$00FF          ; Initialize stack pointer to top of internal RAM
                                    ; Stack grows downward from $FF
```

```
            LDAA    #$04            ; Value for SCI Rate/Mode Control
            STAA    SCI_RMCR        ; Configure serial communication
                                    ; Sets baud rate divisor

            LDAA    #$08            ; Enable transmitter/receiver
            STAA    SCI_TRCSR       ; Write to SCI control register

;-------------------------------------------------------------------------
; Wait for SCI to be ready - check for specific status
;-------------------------------------------------------------------------
WAIT_SCI_READY:
            LDAA    SCI_TRCSR       ; Read SCI status
            LDAA    SCI_RDR         ; Read receive data (clears flags)
            CMPA    #$11            ; Check for specific response ($11 = DC1/XON)
            BNE     WAIT_SCI_READY  ; Loop until ready

            JMP     MAIN_LOOP       ; Jump to main test loop

;=========================================================================================
; PORT INITIALIZATION ROUTINE - $F8C7
; Configures all I/O ports for test operations
;=========================================================================================

INIT_PORTS:
            LDAA    #$0E            ; Set DDR1: bits 1-3 as outputs
            STAA    DDR1            ; Port 1 direction

            LDAA    #$00            ; All inputs
            STAA    DDR2            ; Port 2 direction

            LDAA    #$FF            ; All outputs
            STAA    DDR3            ; Port 3 direction

            LDAA    #$07            ; Bits 0-2 as outputs
            STAA    DDR4            ; Port 4 direction

            LDAA    #$FF            ; Enable all RAM
            STAA    RAM_CTRL        ; RAM control register

;-------------------------------------------------------------------------
; Clear internal RAM ($0080-$00FF)
;-------------------------------------------------------------------------
            LDX     #$00FF          ; Start at top of internal RAM
CLEAR_INT_RAM:
            CLR     0,X             ; Clear byte at X
            DEX                     ; Decrement pointer
            CPX     #$0080          ; Reached bottom?
```

```
            BCC     CLEAR_INT_RAM       ; No, continue clearing

;-------------------------------------------------------------------------
; Clear external RAM ($0400-$07FF) - 1K test area
;-------------------------------------------------------------------------
            LDX     #$0400              ; Start of external RAM
CLEAR_EXT_RAM:
            CLR     0,X                 ; Clear byte
            INX                         ; Next address
            CPX     #$0800              ; End of test area?
            BNE     CLEAR_EXT_RAM       ; No, continue

;-------------------------------------------------------------------------
; Initialize state variables
;-------------------------------------------------------------------------
            LDAA    #$00
            STAA    state_var           ; Clear state variable

            JSR     SET_STATUS_FLAGS    ; Build initial status flags

            LDAA    #$0E                ; Configure PORT1
            STAA    PORT1               ; Set initial port state

            JMP     $FFED               ; Jump to secondary entry (near vectors)

;-------------------------------------------------------------------------
; Alternate entry point
;-------------------------------------------------------------------------
            JMP     DISPATCH_TEST       ; $F8FF -> Jump to test dispatcher

;=========================================================================================
; CHECKSUM CALCULATION ROUTINE - $F902
; Calculates 8-bit checksum of ROM data for verification
; Uses complement arithmetic for error detection
;=========================================================================================

CALC_CHECKSUM:
            LDD     $F800               ; Load first word of ROM (test pattern)
            STD     chksum_src_lo       ; Store as source low

            LDD     chksum_comp         ; Load comparison value
            COMA                        ; Complement A
            COMB                        ; Complement B
            STD     chksum_inv          ; Store inverted value

            LDD     #$0080              ; Base address constant
            STD     base_addr           ; Store base
```

```
        LDAA    chksum_src_lo       ; Start accumulating
        ADDA    chksum_src_lo+1     ; Add bytes
        ADDA    chksum_comp
        ADDA    chksum_comp+1
        ADDA    chksum_inv
        ADDA    chksum_inv+1
        ADDA    base_addr
        ADDA    base_addr+1
        COMA                        ; Complement result
        STAA    calc_sum            ; Store calculated checksum
        RTS

;================================================================================
; STATUS FLAG CHECK ROUTINE - $F926
; Reads PORT1 and PORT4 to build combined status flags
; Status used to determine which test mode to execute
;================================================================================

CHECK_STATUS:
        LDAA    PORT4               ; Read Port 4 status
        ANDA    #$78                ; Mask bits 3-6 (test mode bits)
        STAA    status_flags        ; Store partial flags

        LDAA    PORT1               ; Read Port 1
        ANDA    #$C0                ; Mask bits 6-7 (high control bits)
        CMPA    #$C0                ; Both high?
        BNE     CHK_STAT_SKIP1      ; No, skip

        LDAA    #$80                ; Set bit 7 flag
        ORAA    status_flags        ; Combine with existing
        STAA    status_flags

CHK_STAT_SKIP1:
        LDAA    PORT1               ; Read Port 1 again
        ANDA    #$30                ; Mask bits 4-5
        CMPA    #$30                ; Both high?
        BEQ     CHK_STAT_SKIP2      ; Yes, skip setting bit 1

        LDAA    #$02                ; Set bit 1 flag
        ORAA    status_flags
        STAA    status_flags

CHK_STAT_SKIP2:
        CLR     boundary_flags      ; Clear boundary flags
        LDX     mem_ptr             ; Get memory pointer
        CPX     #$FFFF              ; At lower boundary?
        BNE     CHK_BOUNDARY_HI     ; No, check upper
```

```
;-------------------------------------------------------------------------------
; Set status for lower boundary condition
;-------------------------------------------------------------------------------
SET_STATUS_FLAGS:                       ; Also entry point from INIT
            LDAA    #$04                ; Set boundary flag bit 2
            STAA    boundary_flags
            ORAA    status_flags        ; Add to status
            STAA    status_flags
            BRA     DECODE_TEST_MODE    ; Go decode test

CHK_BOUNDARY_HI:
            CPX     #$0080              ; At upper boundary?
            BNE     DECODE_TEST_MODE    ; No, proceed normally

            LDAA    #$01                ; Set boundary flag bit 0
            STAA    boundary_flags
            ORAA    status_flags
            STAA    status_flags

;===============================================================================
; TEST MODE DECODER - $F969
; Examines status_flags to determine which test routine to execute
; Uses priority-based checking of flag bits
;===============================================================================

DECODE_TEST_MODE:
            LDAA    status_flags        ; Get combined status

            BITA    #$80                ; Bit 7 set? (highest priority)
            BEQ     CHK_MODE_6
            LDAB    #$07                ; Mode 7
            BRA     DO_DISPATCH

CHK_MODE_6:
            BITA    #$08                ; Bit 3 set?
            BEQ     CHK_MODE_5
            LDAB    #$06                ; Mode 6
            BRA     DO_DISPATCH

CHK_MODE_5:
            BITA    #$40                ; Bit 6 set?
            BEQ     CHK_MODE_4
            BITA    #$04                ; Also bit 2?
            BNE     CHK_MODE_5A
            LDAB    #$00                ; Mode 0
            BRA     DO_DISPATCH
CHK_MODE_5A:
            LDAB    #$01                ; Mode 1
```

```
        BRA     DO_DISPATCH

CHK_MODE_4:
        BITA    #$01            ; Bit 0 set?
        BEQ     CHK_MODE_3
        LDAB    #$05            ; Mode 5
        BRA     DO_DISPATCH

CHK_MODE_3:
        BITA    #$10            ; Bit 4 set?
        BEQ     CHK_MODE_2
        LDAB    #$04            ; Mode 4
        BRA     DO_DISPATCH

CHK_MODE_2:
        BITA    #$02            ; Bit 1 set?
        BEQ     USE_MODE_3
        LDAB    #$02            ; Mode 2
        BRA     DO_DISPATCH

USE_MODE_3:
        LDAB    #$03            ; Default: Mode 3

;================================================================================
; TEST DISPATCHER - $F9A5
; Adds mode offset to state, looks up routine address, and calls it
;================================================================================

DO_DISPATCH:
        ADDB    state_var       ; Add current state to mode
        LDX     #$F802          ; Base of state table
        ABX                     ; Add offset
        LDAA    0,X             ; Get new state
        STAA    state_var       ; Update state variable

        LDX     #$F84A          ; Base of mode table
        ABX                     ; Add offset
        LDAB    0,X             ; Get mode byte

        LDX     #$F892          ; Base of jump table
        ABX                     ; Add offset (B*2 for word table)
        LDX     0,X             ; Load routine address
        JSR     0,X             ; Call the test routine

DISPATCH_TEST:
        JMP     CHECK_STATUS    ; Return to status check loop

;================================================================================
```

```
; TEST ROUTINE 0: WRITE RAM TEST - $F9C0
; Tests writing patterns to external RAM area ($0400-$07FF)
;================================================================================

TEST_WRITE_RAM:
        LDX     #$0800          ; End of test area
        STX     end_ptr         ; Save end pointer

        LDX     #$0400          ; Start of test area
        JSR     SEND_SYNC_BYTES     ; Send sync pattern

        LDX     #$008B          ; Address of mem_ptr
        STX     end_ptr

        LDX     #$0089          ; Address of sum_accum
        JSR     SEND_SYNC_BYTES     ; Send more sync

WAIT_PORT_READY:
        LDAA    PORT1           ; Check port status
        ANDA    #$30            ; Mask bits 4-5
        CMPA    #$30            ; Both set?
        BNE     WAIT_PORT_READY     ; No, wait
        RTS

;================================================================================
; TEST ROUTINE 1: SERIAL RECEIVE TEST - $F9DF
; Tests serial data reception with bit synchronization
;================================================================================

TEST_SERIAL_RX:
        JSR     WAIT_PORT_CHANGE    ; Wait for port change

        LDAA    #$08            ; 8 bits to receive
        STAA    bit_count
        CLRA                    ; Clear accumulator

SERIAL_RX_LOOP:
        LDAB    PORT4           ; Sample port
        CMPB    PORT4           ; Compare (wait for stable)
        BEQ     SERIAL_RX_LOOP      ; Loop if same

        LDAB    PORT4           ; Get new value
        BRN     *               ; Delay (BRN = branch never)
        BRN     *               ; More delay
        NOP                     ; Timing adjustment
        NOP
        NOP
        NOP
```

```
        NOP
        NOP
        NOP
        NOP

        CMPA    #$16            ; Received SYN character?
        BEQ     SERIAL_RX_DONE  ; Yes, done

        EORB    PORT4           ; XOR with current
        ASLD                    ; Shift left into A
        BRA     SERIAL_RX_LOOP+2 ; Continue receiving

;-------------------------------------------------------------------------
; Wait for PORT4 change helper
;-------------------------------------------------------------------------
WAIT_PORT_CHANGE:
        LDAB    PORT4
WAIT_P4_LOOP:
        CMPB    PORT4           ; Changed?
        BEQ     WAIT_P4_LOOP    ; No, keep waiting

        LDAB    PORT4           ; Get new value
        NOP
        BRN     *               ; Delay
        BRN     *

        DEC     bit_count       ; Decrement bit counter
        BNE     STORE_RX_BIT    ; More bits to go

        STAA    0,X             ; Store received byte
        LDAA    #$08            ; Reset bit counter
        STAA    bit_count
        INX                     ; Next storage location

STORE_RX_BIT:
        EORB    PORT4           ; Track changes
        ASLD                    ; Shift into accumulator
        BRA     WAIT_PORT_CHANGE ; Continue

SERIAL_RX_DONE:
        BRN     *               ; Delay
        CPX     end_ptr         ; Reached end?
        BNE     STORE_RX_BIT    ; No, continue
        CLC                     ; Clear carry (success)
        RTS

;=========================================================================================
; TEST ROUTINE 2: VERIFY TEST - $FA2A
```

```
; Verifies data patterns and sends sync bytes
;================================================================================

TEST_VERIFY:
        JSR     DEC_MEM_PTR         ; Adjust memory pointer
        LDAA    boundary_flags
        ANDA    #$01                ; Check bit 0
        BNE     VERIFY_EXIT         ; Exit if boundary reached

        JSR     WAIT_PORT_CHANGE    ; Wait for sync
        JSR     SET_STATUS_FLAGS

        CLR     temp_byte           ; Clear temp
        LDAA    #$16                ; SYN character
        STAA    sync_byte           ; Store sync byte

        LDAB    PORT1               ; Get port state
        LDX     #temp_byte          ; Point to temp
        JSR     SEND_BYTE_SYNC      ; Send sync pattern

; Send multiple sync sequences
        CLR     temp_byte
        NOP
        NOP
        JSR     SEND_BYTE_SYNC

        CLR     temp_byte
        NOP
        NOP
        JSR     SEND_BYTE_SYNC

        CLR     temp_byte
        NOP
        NOP
        JSR     SEND_BYTE_SYNC

; Send address/data bytes
        LDX     #sync_byte
        BRN     *               ; Delay
        NOP
        NOP
        JSR     SEND_BYTE_SYNC

        LDX     #$0080          ; RAM start
        BRN     *
        NOP
        NOP
        JSR     SEND_BYTE_SYNC
```

```
            INX
            CPX    #$0089           ; End of block?
            BNE    *-10             ; Loop back

            CLC                     ; Clear carry (success)
VERIFY_EXIT:
            RTS

;================================================================================
; TEST ROUTINE 3: EXTENDED VERIFY - $FA7C
; Extended verification with more sync patterns
;================================================================================

TEST_EXTENDED:
            JSR    DELAY_50_LOOPS    ; Initial delay
            JSR    WAIT_PORT_CHANGE

            CLR    temp_byte
            LDAA   #$16
            STAA   sync_byte

            LDAB   PORT1
            LDX    #temp_byte
            JSR    SEND_BYTE_SYNC

; Multiple sync sequences (similar to TEST_VERIFY)
            CLR    temp_byte
            NOP
            NOP
            JSR    SEND_BYTE_SYNC

            CLR    temp_byte
            NOP
            NOP
            JSR    SEND_BYTE_SYNC

            CLR    temp_byte
            NOP
            NOP
            JSR    SEND_BYTE_SYNC

            LDX    #sync_byte
            BRN    *
            NOP
            NOP
            JSR    SEND_BYTE_SYNC
```

```
; Test external RAM area
        LDX   #$0400          ; External RAM start
        BRN   *
        NOP
        NOP
        JSR   SEND_BYTE_SYNC

EXT_RAM_LOOP:
        INX
        CPX   #$0800          ; End of external RAM?
        BNE   EXT_RAM_LOOP-6  ; Continue

; More verification patterns
        LDX   #temp_byte
        JSR   SEND_BYTE_SYNC

        CLR   temp_byte
        NOP
        NOP
        JSR   SEND_BYTE_SYNC

        LDAA  #$16
        STAA  sync_byte
        LDX   #sync_byte
        NOP
        JSR   SEND_BYTE_SYNC

        LDX   #$0089
        NOP
        NOP
        BRN   *
        JSR   SEND_BYTE_SYNC

        INX
        BRN   *
        NOP
        NOP
        JSR   SEND_BYTE_SYNC

        LDX   #temp_byte
        NOP
        BRN   *
        NOP
        JSR   SEND_BYTE_SYNC

        CLC
        RTS
```

```
;================================================================================
; SERIAL BIT OUTPUT ROUTINE - $FAFA
; Sends one byte bit-by-bit via PORT1
;================================================================================

SERIAL_BIT_OUT:
        EORB    #$08            ; Toggle bit 3 (clock)
        STAB    PORT1             ; Output to port
        LDAA    #$08            ; 8 bits
        BRA     BIT_OUT_ENTRY

BIT_OUT_LOOP:
        EORB    #$08            ; Toggle clock
        STAB    PORT1
        NOP
        BRN     *               ; Timing delay

BIT_OUT_ENTRY:
        NOP
        NOP
        NOP
        NOP
        BRN     *

        ROL     0,X             ; Rotate data byte left (MSB to carry)
        BCC     BIT_OUT_LOW       ; If carry clear, output low

        EORB    #$08            ; Toggle clock
        STAB    PORT1
        DECA                    ; Decrement bit count
        BEQ     BIT_OUT_DONE      ; Done if zero

; Timing delay loop
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BRA     BIT_OUT_LOOP
```

```
BIT_OUT_LOW:
        NOP
        BRA     BIT_OUT_LOOP+3      ; Skip toggle for low bit

BIT_OUT_DONE:
        ROL     0,X                 ; Final rotate
        RTS

;=============================================================================
; TEST ROUTINE 4: SYNC TEST - $FB30
; Tests synchronization with PORT4
;=============================================================================

TEST_SYNC:
        JSR     WAIT_PORT_CHANGE

        LDAB    PORT1
        EORB    #$08                ; Toggle bit 3
        STAB    PORT1

        LDAA    #$08                ; 8 iterations
        BRA     SYNC_TEST_ENTRY

SYNC_TEST_LOOP:
        LDAB    PORT1
        EORB    #$08
        STAB    PORT1
        NOP

SYNC_TEST_ENTRY:
        LDAB    PORT4               ; Read status
        ANDB    #$38                ; Mask bits 3-5
        BNE     SYNC_TEST_EXIT      ; Non-zero = fail

        NOP
        LDAB    PORT1
        ANDB    #$30                ; Check bits 4-5
        CMPB    #$30                ; Both set?
        BNE     SYNC_TEST_EXIT      ; No = fail

        LDAB    PORT4
        ANDB    #$38
        BNE     SYNC_TEST_EXIT

        NOP
        LDAB    PORT1
        ANDB    #$30
```

```
            CMPB    #$30
            BNE     SYNC_TEST_EXIT

            LDAB    PORT4
            ANDB    #$38
            BNE     SYNC_TEST_EXIT

            NOP
            DECA                    ; Decrement counter
            BNE     SYNC_TEST_LOOP      ; Continue

SYNC_TEST_EXIT:
            CLC
            RTS

;===============================================================================
; CHECKSUM WITH SUM LOOP - $FB6E
; Calculates running checksum of external RAM
;===============================================================================

CALC_SUM_LOOP:
            LDX     #$0400          ; Start of external RAM

            LDAA    PORT4           ; Sample initial state
            STAA    status_flags

            LDD     #$0000          ; Clear 16-bit accumulator
            STD     sum_accum

SUM_LOOP:
            LDAA    PORT1           ; Toggle PORT1 bit 3
            EORA    #$08
            STAA    PORT1

            LDAA    #$05            ; Delay loop
SUM_DELAY:
            DECA
            BNE     SUM_DELAY

            LDAB    0,X             ; Get byte from RAM
            CLRA                    ; Clear high byte
            ADDD    sum_accum       ; Add to running sum
            STD     sum_accum

            LDAA    PORT4           ; Check for state change
            EORA    status_flags
            ANDA    #$78            ; Mask relevant bits
            BNE     SUM_LOOP_EXIT       ; Exit if changed
```

```
            INX                 ; Next byte
            CPX   #$0800        ; End of RAM?
            BNE   SUM_LOOP      ; Continue

; Additional delay loop
            LDX   #$0384        ; 900 iterations
EXTRA_DELAY:
            JSR   CALC_SUM_DELAY    ; Delay subroutine
            DEX
            BNE   EXTRA_DELAY

SUM_LOOP_EXIT:
            RTS

;================================================================================
; DELAY ROUTINE - $FBA4
; Performs 50 iterations of DELAY_50_LOOPS
;================================================================================

DELAY_ROUTINE:
            LDAA  #$32          ; 50 decimal
DELAY_50_LOOPS:
            JSR   DELAY_50_LOOPS    ; Recursive call (nested delay)
            DECA
            BNE   DELAY_50_LOOPS

            LDAA  PORT1         ; Restore port state
            ANDA  #$F9          ; Clear bits 1-2
            STAA  PORT1
            CLC
            RTS

;================================================================================
; INVERT PORT BITS - $FBB4
; Inverts bits 6-7 of PORT1 and rotates into position
;================================================================================

INVERT_PORT_BITS:
            LDAA  PORT1
            COMA                ; Complement
            ANDA  #$C0          ; Keep bits 6-7
            CLC
            ROLA                ; Rotate left 4 times
            ROLA                ; (move bits 6-7 to bits 2-3)
            ROLA
            ROLA
```

```
        LDAB    PORT1
        ANDB    #$F9            ; Clear bits 1-2
        ABA                     ; Combine
        STAA    PORT1
        RTS

;===============================================================================
; TEST ROUTINE 5: WAIT FOR CHANGE - $FBC6
; Waits for PORT4 to change, monitors boundary
;===============================================================================

TEST_WAIT_CHANGE:
        JSR     WAIT_PORT_CHANGE
        LDAB    PORT4           ; Get initial state

WAIT_CHANGE_LOOP:
        JSR     DEC_MEM_PTR     ; Decrement pointer
        LDAA    boundary_flags
        ANDA    #$01            ; Check boundary
        BNE     WAIT_CHANGE_EXIT    ; Exit if at boundary

        JSR     INC_MEM_PTR     ; Restore pointer
        CMPB    PORT4           ; Changed?
        BEQ     WAIT_CHANGE_LOOP    ; No, continue waiting

WAIT_CHANGE_EXIT:
        RTS

;===============================================================================
; TEST ROUTINE 6: MONITOR - $FBDC
; Monitors for PORT4 changes with boundary checking
;===============================================================================

TEST_MONITOR:
        JSR     WAIT_PORT_CHANGE
        LDAB    PORT4

MONITOR_LOOP:
        JSR     INC_MEM_PTR
        JSR     TIMER_WAIT

        LDAA    boundary_flags
        ANDA    #$04            ; Check bit 2
        BNE     MONITOR_EXIT    ; Exit if set

        CMPB    PORT4           ; Changed?
        BEQ     MONITOR_LOOP    ; No, continue
```

```
MONITOR_EXIT:
        RTS

;================================================================================
; MEMORY POINTER INCREMENT - $FBF2
; Increments mem_ptr with boundary detection
;================================================================================

INC_MEM_PTR:
        PSHB                ; Save B
        LDX   mem_ptr
        CPX   #$0080        ; At upper bound?
        BNE   DO_INC_PTR

        LDAA  boundary_flags    ; Set boundary flag
        ORAA  #$01
        BRA   SAVE_PTR

DO_INC_PTR:
        INX                 ; Increment pointer
        STX   mem_ptr
        LDAA  boundary_flags
        ANDA  #$FE          ; Clear bit 0

SAVE_PTR:
        STAA  boundary_flags
        STX   chksum_comp       ; Also store in comparison area
        PULB                ; Restore B
        RTS

;================================================================================
; MEMORY POINTER DECREMENT - $FC0D
; Decrements mem_ptr with boundary detection
;================================================================================

DEC_MEM_PTR:
        PSHB
        LDX   mem_ptr
        CPX   #$FFFF        ; At lower bound?
        BNE   DO_DEC_PTR

        LDAA  boundary_flags
        ORAA  #$04          ; Set bit 2
        BRA   SAVE_PTR_DEC

DO_DEC_PTR:
        DEX
        STX   mem_ptr
```

```
            LDAA    boundary_flags
            ANDA    #$FB            ; Clear bit 2

SAVE_PTR_DEC:
            STAA    boundary_flags
            STX     chksum_comp
            PULB
            RTS

;===============================================================================
; SET BIT AND WAIT - $FC28
; Sets PORT1 bit and waits for PORT4 change
;===============================================================================

SET_BIT_WAIT:
            PSHB
            PSHA

            LDAA    PORT1
            ORAA    #$04            ; Set bit 2
            STAA    PORT1

            LDX     #$0111          ; Timeout counter (273)
            LDAB    PORT4

SBW_WAIT_LOOP:
            JSR     TIMER_CHECK     ; Check timer
            CMPB    PORT4           ; Changed?
            BNE     SBW_EXIT
            DEX
            BNE     SBW_WAIT_LOOP   ; Continue waiting
            CLC                     ; Timeout - clear carry

SBW_EXIT:
            LDAA    PORT1
            ANDA    #$FB            ; Clear bit 2
            STAA    PORT1
            PULA
            PULB
            RTS

;===============================================================================
; TIMER WAIT ROUTINE 1 - $FC49
; Waits for timer output compare flag
;===============================================================================

TIMER_WAIT_1:
            PSHA
```

```
            PSHB

            LDAA    TCSR            ; Read timer control/status
            LDD     COUNTER_H       ; Read counter
            ADDD    #TIMER_DELAY    ; Add delay constant (1000)
            STD     OCR_H           ; Store in output compare

TIMER_WAIT_1_LOOP:
            LDAA    TCSR            ; Check status
            ANDA    #$40            ; Output compare flag?
            BEQ     TIMER_WAIT_1_LOOP   ; No, keep waiting

            PULB
            PULA
            RTS

;===============================================================================
; TIMER WAIT ROUTINE 2 - $FC5D (TIMER_WAIT)
; Similar to TIMER_WAIT_1 but different register save order
;===============================================================================

TIMER_WAIT:
            PSHB
            PSHA

            LDAA    TCSR
            LDD     COUNTER_H
            ADDD    #TIMER_DELAY
            STD     OCR_H

TIMER_WAIT_LOOP:
            LDAA    TCSR
            ANDA    #$40
            BEQ     TIMER_WAIT_LOOP

            PULA
            PULB
            RTS

;===============================================================================
; DATA AREA / LOOKUP TABLES - $FC71-$FCAC
; Various data tables and possibly Z80-related code fragments
;===============================================================================

            ORG     $FC71

; This area contains mixed data that may include:
; - ADAMnet protocol patterns
```

```
; - Z80 communication sequences
; - Additional test patterns

DATA_TABLES:
        FCB     $02,$C9,$ED,$43,$61,$B9,$C3,$06
        FCB     $B6,$21,$00,$00,$19,$09,$6E,$26
        FCB     $00,$C9,$01,$07,$0D,$13,$19,$05
        FCB     $0B,$11,$17,$03,$09,$0F,$15,$02
        FCB     $08,$0E,$14,$1A,$06,$0C,$12,$18
        FCB     $04,$0A,$10,$16

; Possible keyboard scan code table or test pattern
SCAN_TABLE:
        FCB     $79,$32,$5B,$B9,$FE,$04,$CA,$FB
        FCB     $B6,$FE,$05,$C2,$1F,$B7,$E6,$01
        FCB     $4F,$D3,$0C,$C5,$21,$55,$B6,$11
        FCB     $18,$D9,$01,$1B,$00,$ED,$B0,$C1
        FCB     $CB,$21,$CB,$21,$CB,$21,$CB,$21
        FCB     $CD,$8B,$B6,$21,$30,$B8,$06,$00
        FCB     $09,$C9,$C5,$21,$70,$B6,$11,$18
        FCB     $D9,$01,$1B,$00,$ED,$B0,$C1,$C3
        FCB     $03,$B6,$ED,$43,$5D,$B9,$C9,$3A
        FCB     $5D,$B9,$D3,$03,$3E,$02,$D3,$08
        FCB     $3E,$1F,$D3,$00,$CD,$13,$B8,$DB
        FCB     $00,$CB,$47,$20,$FA,$C9,$3E,$0A
        FCB     $32,$59,$B9,$CD,$F2,$B7,$21,$1A
        FCB     $B8,$CD,$0A,$B8,$CD,$34,$B7,$2A
        FCB     $61,$B9,$01,$80,$00,$09,$7E,$32
        FCB     $5A,$B9,$2A,$61,$B9,$0E,$04,$06
        FCB     $81,$3E,$03,$D3,$08,$ED,$56,$3E
        FCB     $8C,$D3,$00,$FB,$76,$F3,$DB,$00
        FCB     $CD,$FE,$B7,$4F,$E6,$9C,$3E,$02
        FCB     $D3,$08,$2A,$61,$B9,$11,$80,$00
        FCB     $19,$3A,$5A,$B9,$77,$C2,$98,$B7
        FCB     $3E,$00,$C9,$3A,$59,$B9,$3D,$28
        FCB     $08,$32,$59,$B9,$CD,$AB,$B6,$18
        FCB     $AD,$3E,$01,$C9,$3E,$0A,$32,$59
        FCB     $B9,$CD,$F2,$B7,$21,$25,$B8,$CD
        FCB     $0A,$B8,$CD,$34,$B7,$2A,$61,$B9
        FCB     $7E,$0E,$04,$ED,$79,$23,$06,$80
        FCB     $ED,$56,$3E,$01,$D3,$08,$3E,$AC
        FCB     $D3,$00,$FB,$76,$F3,$DB,$00,$CD
        FCB     $FE,$B7,$4F,$E6,$DC,$C2,$E4,$B7
        FCB     $3E,$02,$D3,$08,$3E,$00,$C9,$3A
        FCB     $59,$B9,$3D,$28,$BC,$32,$59,$B9
        FCB     $CD,$AB,$B6,$18,$BF

;================================================================================================
```

```
; Additional routines and data continue through $FF13
; Much of this area contains test patterns, lookup tables,
; and possibly code for interfacing with the Z80/MIOC
;================================================================================

; ... (Additional data tables from $FDA4 through $FF13)

;--------------------------------------------------------------------------
; Fill area with $FF (unused ROM space)
;--------------------------------------------------------------------------
            ORG     $FF14

UNUSED_FILL:
            FCB     $FF,$FF,$FF         ; Repeated $FF fill
            ; ... continues to near vector table

;================================================================================
; INTERRUPT VECTORS - $FF20-$FFFF (partial)
; Note: Some vectors appear corrupted or are test patterns
;================================================================================

            ORG     $FF20

; The standard 68701 vector table would be at $FFF0-$FFFF:
;   $FFF0-$FFF1: SCI Serial
;   $FFF2-$FFF3: Timer Overflow
;   $FFF4-$FFF5: Timer Output Compare
;   $FFF6-$FFF7: Timer Input Capture
;   $FFF8-$FFF9: IRQ1 (External)
;   $FFFA-$FFFB: SWI (Software Interrupt)
;   $FFFC-$FFFD: NMI (Non-Maskable Interrupt)
;   $FFFE-$FFFF: RESET

; The vectors in this dump show mostly $FF (unused/not implemented)
; with RESET vector presumably pointing to $F8B1

;================================================================================
; END OF DISASSEMBLY
;================================================================================

            END
```