LA TROBE UNIVERSITY
DEPARTMENT OF ELECTRONIC ENGINEERING

ELECTRONICS-2

Microprocessors: Lecture 3
Introduction to the 68HC11
Address Map, Registers
and Basic Instruction Set

Contents:

16 Bit Memory Address Map

Address:	Value (Binary)								(Hex)
FFFF	0	0	0	0	0	0	0	0	00
FFFE	1	1	1	0	0	0	0	0	E0
FFFD	1	1	1	1	1	1	0	1	FD
FFFC	0	0	0	0	0	0	0	0	00
FFFB	1	1	1	1	1	0	1	0	FA
FFFA	0	0	0	0	0	0	0	0	00
FFF9	1	1	1	1	0	1	1	1	F7
FFF8	0	0	0	0	0	0	0	0	00
:									
:									
0007	0	0	0	0	0	1	1	1	07
0006	0	0	0	0	0	1	1	0	06
0005	0	0	0	0	0	1	0	1	05
0004	0	0	0	0	0	1	0	0	04
0003	0	0	0	0	0	0	1	1	03
0002	0	0	0	0	0	0	1	0	02
0001	0	0	0	0	0	0	0	1	01
0000	0	0	0	0	0	0	0	0	00

Figure 1 -Memory Address map. 2^{16} Addresses

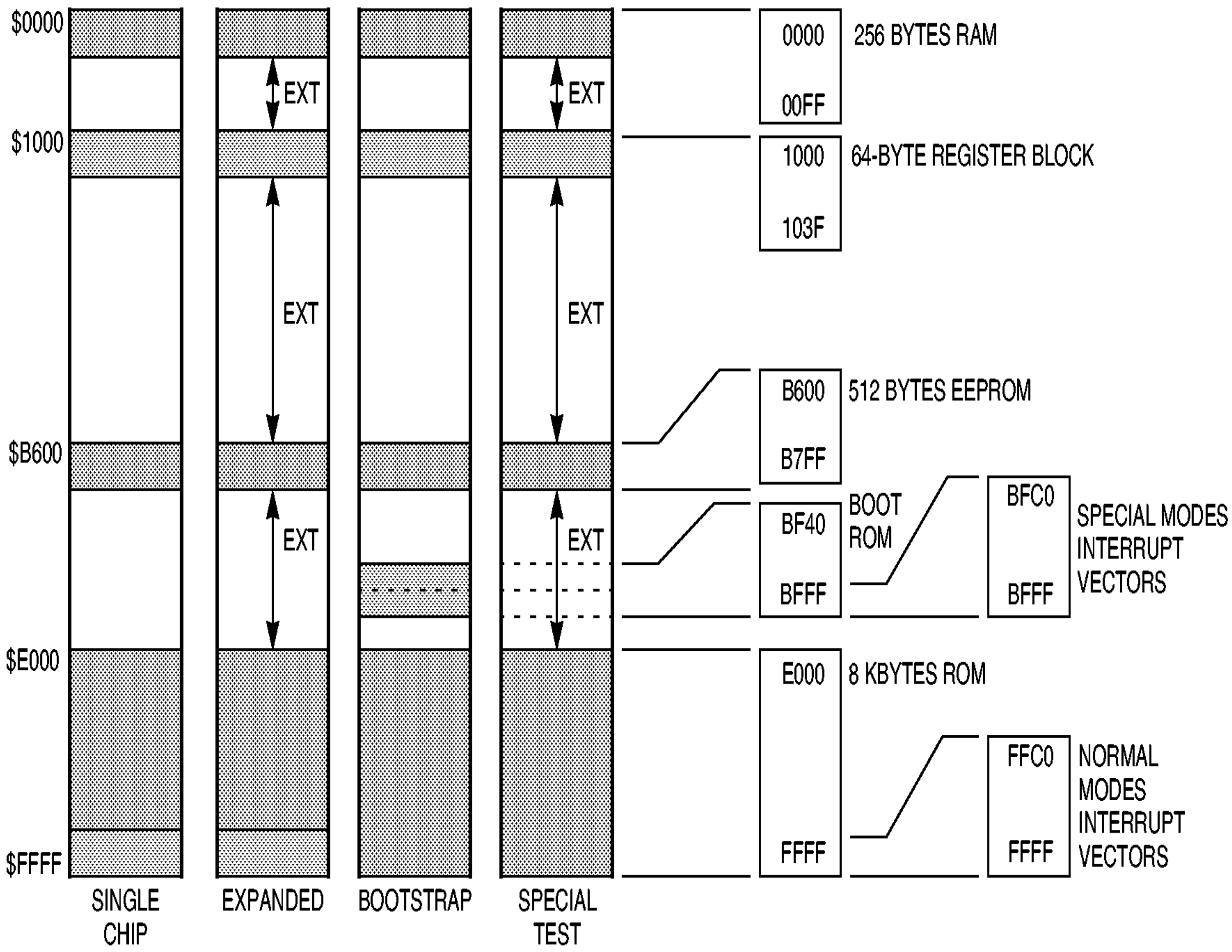


Figure 2- 68HC11 Memory Map.

Which mode this chip starts in is determined by the MODA & MODB pins.

68HC11 Instructions

Example Register Usage:

```
        LDAA #$12       ; Load Accumulator A with the number $12 (=18 decimal)
        LDAB #$34       ; Load Accumulator B with the number $34 (=52 decimal)
        ABA             ; Add accumulator B to A
        STAA RESULT     ; Store Accumulator A in an address labeled RESULT
        SWI             ; Stop the program execution - Exit to BUFFALO Monitor
                        ; Result should contain $46 (=70 decimal)
RESULT  RMB 0           ; RMB = Reserve Memory Byte
```

Pointer & Index Registers

SP - Stack Pointer

IX - Index X

IY - Index Y

PC Program Counter

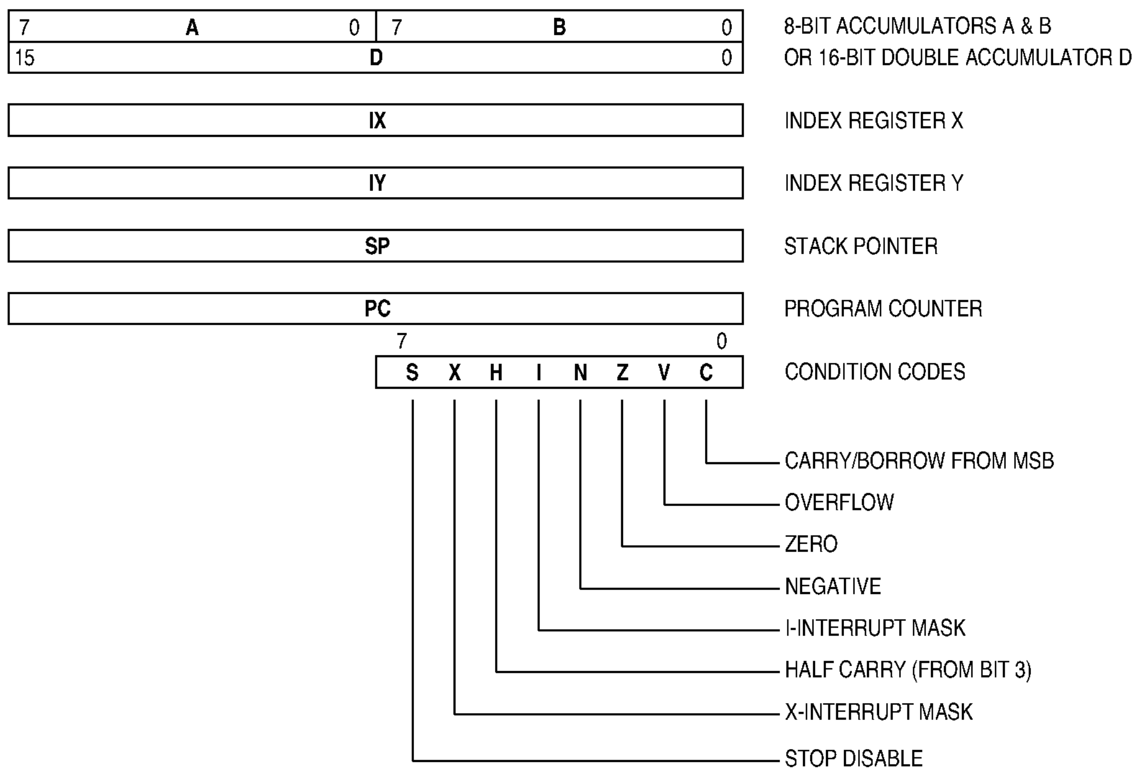


Figure 1-2 M68HC11 Programmer's Model

Figure 3 - 68HC11 Register Set

Stack

The stack is a section of memory set aside for, temporary storage during the execution of a program.

The stack grows downwards through memory as it fills (PUSH), and recedes up through memory as it shrinks (PULL).

Stack Pointer

The stack pointer is a 16-bit register that points to the next free location on the stack.
When a value is **push**ed onto the stack, the stack pointer automatically decrements.
When a value is **pul**led from the stack, the stack pointer automatically increments.

Stack Operations : PUSH

When a value located in an accumulator or register is pushed onto the stack, it is copied from the accumulator or register and stored to the location pointed to by the stack pointer.
The stack pointer then decrements to point to the next free location.

Stack Operation : PULL

When a value is pulled from the stack into an accumulator or register, the stack pointer increments to point to last used location on the stack. The value at that location is then copied into the accumulator or register. (equivalent instruction on 80x86 - **POP**)

Stack Pointer

Not normally altered directly except once on power on reset (eg: LDS #STACK ;initialize start of stack) The stack is used to hold parameters passed to subroutines and to save values temporarily

Example: Using the Stack Pointer hold data

```
>asm 2100                ; Assemble code at 2100 Hex.
2100 PSHA         >      ; Push Accumulator A
2101 PSHB         >      ; Push Accumulator B
2102 PSHX         >      ; Push Register X
2103 PSHY         >      ; Push Register Y
2105 PULX         >      ; Pull Register   X      (= pop)
2106 PULY         >      ; Pull Pegister   Y
2108 PULA         >      ; Pull Accumulator A
2109 PULB         >      ; Pul Accumulator B
210A RTS          >      ; Return from subroutine (PC is on stacktop)
210B TEST         >♥     Press Control-C to exit the assembler
```

Calling the subroutine at $2100 saves A, B, X then Y. Normally the reverse order is used to restore the values to their original registers, however, for this illustration we have swapped the order of X with Y and B with A. The result of swapping registers can be seen using the BUFFALO command **rm** (Register Modify) as follows:

```
>rm p
P-E328 Y-BAAA X-F000 A-00 B-FF C-D0 S-0041
P-E328 2100

>call 2100

P-2100 Y-F000 X-BAAA A-FF B-00 C-D0 S-0041
>
```

Example: Single Hex Number Conversion to ASCII using C Language

```
// The algorithm to convert Hex2Ascii in the C language:
// We know ASCII characters
//          0..9 are $30..$39 - So the function is Add $30, and for
//         A..F are $41..47   - Add ($41 - $0A) = $37
char hex2ascii (char hex_in) // returns an ASCII character
{ char result;
   result = (hex_in & 0x0F);    // Mask input to be in the range $0..$F
   result += 0x30;              // assume range 0..9, so add 0x30
   if (result > 0x39)           // check above assumption
     {                          // result is 0x3A or more,
                                // incorrect assumption so fix it
       result += 7;             // add 7 to get the final result
                                // in the range $41..$47
     }
  return (result);              // return the result - is returned in Accumulator A
}
```

Hex2ASCII using AS11 (the M68HC11 assembler)

```
; This subroutine converts a single hex number (0 to F) to ASCII
; ON ENTRY:  ACCA =  hex digit to be converted
; ON RETURN: ACCA = ASCII character code of hex digit
HEX2ASCII
        ANDA    #$0F    ; Ensure number is in range $0..$F
        ADDA    #$30    ; 30 Hex = ASCII character '0'
        CMPA    #$39    ; check if digit > '9'
        BMI     DONTADD
        ADDA    #$07    ; ACCA > '9' so add another 7
DONTADD
        RTS             ; return to calling routine
                        ; Accumulator A holds the returned result
```

Entering Hex2ASCII using BUFFALO Program

```
ASM 2000
ANDA #0F
ADDA #30
CMPA #39
BMI 200A  - Buffalo's inbuilt assembler cannot use symbols
ADDA #07
RTS
```

As you can see, BUFFALO assumes all numbers are hexadecimal.

Entering Hex2ASCII using Buffalo

```
La Trobe University HC-COM.
BUFFALO 3.41 (ext) - Bit User Fast Friendly Aid to Logical Operation
by Tony Fourcroy. Ported to HC-COM by jtc.

Free memory range: 2000 - 7FFF
>ASM 2000

2000 TEST             >ANDA #0F
     84 0F
2002 TEST             >ADDA #30
     8B 30
2004 TEST             >CMPA #39
     81 39
2006 TEST             >BMI 200A
     2B 02
2008 TEST             >ADDA #07
     8B 07
200A TEST             >RTS
     39
200B TEST             >

200C TEST             >♥
>_
```

Byte2ASCII in C

```
; This subroutine converts a Byte to 2 ASCII digits
; ON ENTRY:    ACCA =   byte to be converted
; ON RETURN:  2 ASCII character codes
int byte2ascii (char byte_in)            // returns an ASCII character
{ int result;                            // integer = 2 bytes
   result = (hex2ascii (byte_in >> 4);   // convert high hex digit to character
   result = (result  << 8);              // and store in upper 8 bits
   result += hex2ascii (byte_in);        // convert low hex digit to character
   return (result);                      // result is returned in A & B register
                                         // AccA=Most Significant Character
                                         // AccB=Least Significant Character
}
```

Byte2ASCII - AS11 (68HC11 assembler)

```
; This subroutine converts a Byte to 2 ASCII digits
; ON ENTRY:    ACCA =   byte to be converted
; ON RETURN: ACCA:ACCB = 2 ASCII character codes
BYTE2ASCII
        PSHA                        ; save Accumulator A on stack
        ANDA        #$0F            ; mask low byte
        JSR         HexToAscii
        TAB                         ; Transfer Character in AccA to AccB
        PULA                        ; restore Accumulator A from stack
        LSRA                        ; shift Accumulator A right four times
        LSRA                        ; to transfer high nybble to low nybble
        LSRA
        LSRA
        JSR         HexToAscii  ; most significant character in AccA
        RTS                         ; return to calling routine
                                    ; Accumulator A holds the returned result
```

The reverse problem:

Conversion of ASCII to Hex

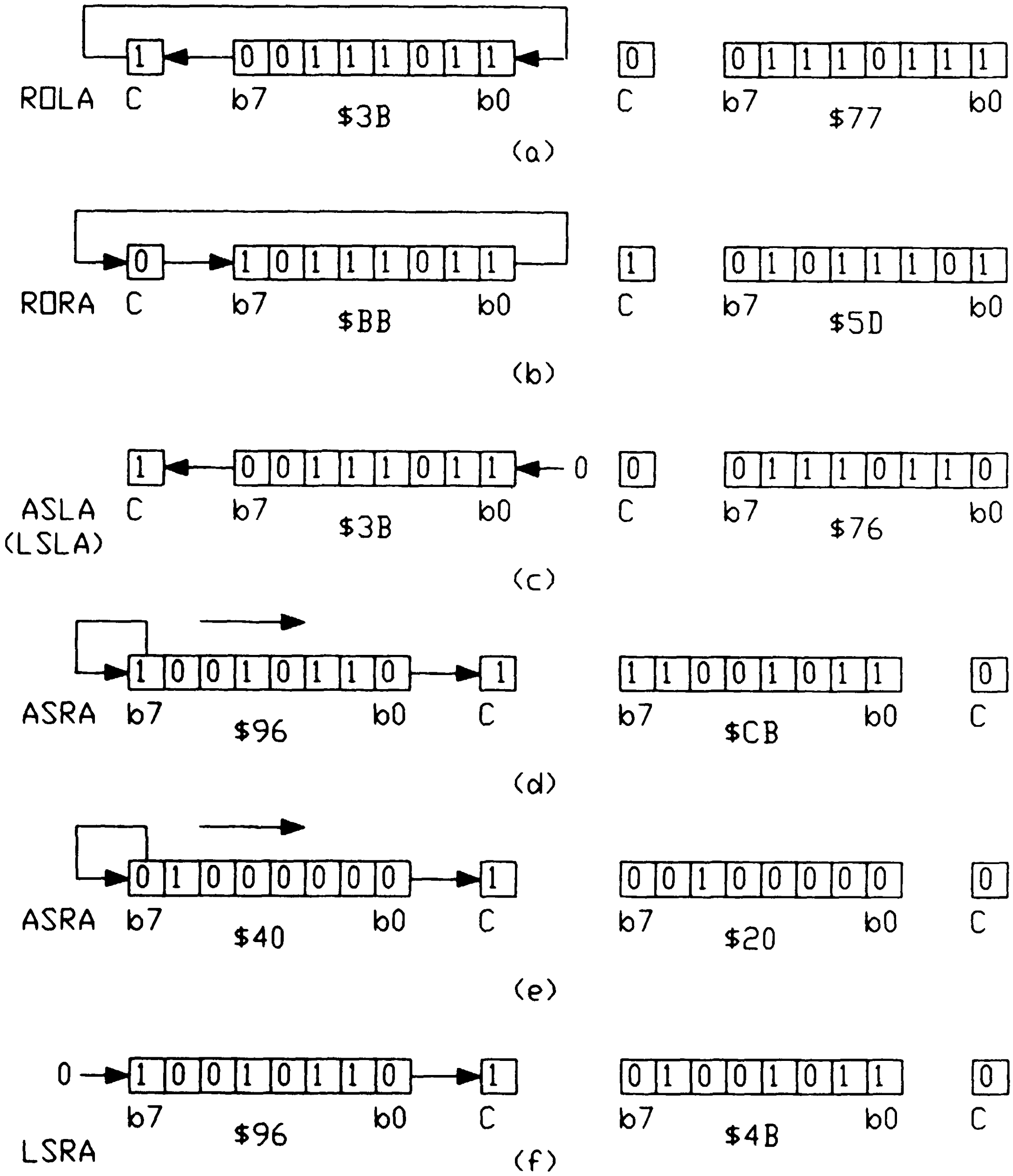
```
; This subroutine converts a single ASCII character to a hex digit
; ON ENTRY: ACCA = ASCII code of hex digit to be converted
; ON EXIT:     ACCA = a single-digit hex number
ASCII2HEX
        SUBA        #$30            ; assume $30-$39,
        CMPA        #$9             ; check if result > 9,
        BMI         DONTSUB         ; yes subtract another 7 if it is
        SUBA        #$07
DONTSUB
        RTS                         ; return to calling routine
                                    ; Accumulator A holds the returned result
```

USING AS11 Assembler Data Types

Expressions used by assembler to allocate initialised space (similar to constants in C)
FCB - Form Constant Byte
Creates initialised space for byte sized objects
FCC - Form Constant Character String
FCW - Define Word

Examples
Shift Left & Right
Binary Multiplication
Binary Division
Extending for larger numbers

Shift & Rotate - Left & Right



Next Lecture
Addressing Modes
Conditional Jumps

Acknowledgments
Notes revised by Paul Main, 2004, drawing from material originally written by John Catsoulis and Sen Goh
Most images are courtesy of Motorola technical data sheets 11A8TD.PDF and 11RM.PDF
Refer to resources world wide web:http://thor.ee.latrobe.edu.au/~pmain/