Motorola HC11

Fun with Microcontrollers and Embedded Systems
What is a microcontroller?

• A processor
  • Usually not cutting edge
  • Dependable
    • All major bugs well known
  • Predictable
    • Critical for real-time processing
• On-chip peripherals and memory
• Parallel and serial digital I/O
• Analog I/O
• Counters and timers
• Internal ROM and/or EPROM
What are microcontrollers used in?

- Watches
- Microwaves
- Stereo Receivers
- ATMs
- PDA’s, MP3 players
- Automobiles

Some products that you might know:

- NASA’s Sojourner Rover – 8-bit Intel 80C85
- Palm Vx handheld – 32-bit Motorola Dragonball EZ
- Sonicare toothbrush – 8-bit Zilog Z8
- The Vendo V-MAX 720 Soda Machine – Motorola HC11
- Miele dishwasher – 8-bit Motorola 68HC05
- Hunter 44550 Programmable Thermostat – (4-bit cpu)
Microcontrollers

Microcontroller unit sales are 15x higher than microprocessors.
• ... and are MUCH cheaper.
Microcontrollers are a large market. 8-bit controllers are the largest, but not growing the fastest.

Microcontrollers

- 8-bit microcontroller growth rate for 2003 is at 9.42%.
- Microcontroller growth rate in general is 11.5%.

Source: Cahners In-Stat Group
• 16- and 32-bit and higher are on the rise. They will double their unit market share from 15.11% in 1998 to 31.56% in 2003, decreasing 4-bit and 8-bit devices.

• But, in 2003, the 8-bit microcontrollers will outnumber the higher bit units by almost 80% in the market place.

Source: Cahners In-Stat Group
So what languages are they being programmed in?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly</td>
<td>~ 21%</td>
<td>~ 10%</td>
</tr>
<tr>
<td>C</td>
<td>~ 69%</td>
<td>~ 80%</td>
</tr>
<tr>
<td>C++</td>
<td>~ 5%</td>
<td>~ 6%</td>
</tr>
<tr>
<td>Java</td>
<td>~ 1%</td>
<td>~ 2%</td>
</tr>
<tr>
<td>Other</td>
<td>~ 3%</td>
<td>~ 2%</td>
</tr>
</tbody>
</table>

Motorola M68HC11

- M6801 CPU core
- ROM (8KB), EEPROM (512B), RAM (256B)
- Counter/Timer system
- A/D converter
  - D/A in kit
- Parallel I/O
- Serial I/O (SPI and SCI)
- Expansion bus
  - To add more memory
HC11 Architecture
M8601 CPU Core

- 8-bit
- CISC
- Accumulator-based
- Results wind up in a 8-bit accumulator A or B
- 16-bit results go to accumulator AB, called D
- Two index registers for addressing memory or for counting - X and Y
- Dedicated stack pointer
- Push and Pop instructions use this - grows toward smaller memory addresses like in MAL
- Program Counter
- Condition Codes
M8601 CPU Core

ALU - Arithmetic Logic Unit

- ALU
- Accumulator
- Memory
- Op1
- PC
- Instr
- Op1 Addr
HC11 Microcontroller

M68HC11E Series Programming Model

- 8-BIT ACCUMULATORS A & B
- OR 16-BIT DOUBLE ACCUMULATOR D
- INDEX REGISTER X
- INDEX REGISTER Y
- STACK POINTER
- PROGRAM COUNTER
- CONDITION CODES
- CARRY/BORROW FROM MSB
- OVERFLOW
- ZERO
- NEGATIVE
- I-INTERRUPT MASK
- HALF CARRY (FROM BIT 3)
- X-INTERRUPT MASK
- STOP DISABLE
HC11 Microcontroller

Condition Codes

- Not present for MIPS integer instructions
- Single-bit flags set appropriately for most instruction (several instructions do not, including push and pop)

| C | Carry/Borrow |
| V | Overflow     |
| Z | Zero         |
| N | Negative     |
| H | Half-Carry   |
### Example of how condition codes are used

<table>
<thead>
<tr>
<th>MAL code</th>
<th>HC11 code</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>bge $t0, 4, mylabel</td>
<td>cmpa #4</td>
<td>Subtract AccA - 4, set CCR</td>
</tr>
<tr>
<td></td>
<td>bge mylabel</td>
<td>Branch if (CCR[Z]=1 OR CCR[N]=0)</td>
</tr>
</tbody>
</table>
Configuration Registers

• Bits set by the user to tell the processor how to do things

I  Interrupt Mask
X  XIRQ mask
S  Disable STOP instructions
Return to Addressing Modes

Addressing Modes

MIPS (TAL) has:
• Register direct
• Base Displacement

HC11 has several:
• Check opcode listings to see what modes work with what instructions
• Also check what condition codes are set
HC11 Addressing Modes

Supports these addressing modes:

- Immediate (IMM)
- Extended (EXT)
- Direct (DIR)
- Indexed (INDX and INDY)
- Inherent (INH)
- Relative (REL)
Immediate addressing

• 1 or 2 byte immediate depending on register involved (LDAA vs. LDD)
• **ALWAYS** include a #
• Several formats for different bases -- C-style constants instead of what is in the HC11 manual (don't use !,$,@,%)
  • Decimal: \( \text{LDAA \#245} \)
  • Hexadecimal: \( \text{LDAA \#0x61} \)
  • Octal: \( \text{LDAA \#041} \)
  • Binary: \( \text{LDAA \#0b11000011} \)
  • ASCII: \( \text{LDAA \#'a'} \)
Extended and Direct addressing

• Access an absolute memory location
• Essentially the same mode, but with 8-bit (direct) or 16-bit (enhanced or extended) addresses
• The assembler will decide on which to use based on how many bits are needed
• Example

  // Your data starts at address 0x4000:
  .sect .data
  var: .word 0x1000 // Allocate/initialize a word
      // Note: a word is 16 bits!

  .sect .text
  SUBD var       // Subtract [var] from D
  SUBD 0x4000    // Subtract [var] from D
Indexed addressing

- Uses index register X or Y
- Similar to MAL “lw $t0, 4($sp)”
  - But can access memory and use it all at once!!
- Example

```c
#define mydef 4 // c preprocessor used
ldx #var // Like MAL load address
addd 0,X // add contents of 0($X) to D
// (Note “addd X” doesn’t work)
addd 2,X // add contents of 2($X) to D
addd mydef*2, X // add contents of 8($X) to D
```
Inherent addressing

- Opcode fully specifies operation; no addressing
- Examples:

  **INCB** increment accumulator B
  **ASLA** Arithmetic Shift Left accumulator A
  **PSHY** Push index register Y onto stack
Relative addressing

• Used for branches by the assembler
• Offsets from -128 to +128 bytes
• Use jumps for longer branches
HC11 Address Modes Review

ldab #0  // loads the number 0 into b
ldaa foo  // loads the contents of byte variable foo into “a” acc.
ldy #foo  // loads the address of foo into y
ldab 0, x  // loads the byte at address x+0 into “b” acc.
ldx 0  // loads whatever is at memory address 0 into “x” index.
        // You don't want this in 12c ever.
Motorola 68HC11 Instructions

- HC11 Instructions have variable lengths (not like MAL)
- Careful coding can keep applications small and able to fit in the EPROM
  - We don’t have to worry about this since we’re using Expansion Mode: there is extra memory in the microkits.
- The opcode is always 1 byte
- An additional 0-3 bytes specify the data to work with
Motorola 68HC11 Instruction Set

- Accumulator and Memory Instructions
- Stack and Index Register Instructions
- Condition Code Register Instructions
- Program Control Instructions
HC11 Instructions

Accumulator and Memory Instructions

Can be broken up into these 6 general types:

1. Loads, stores, and transfers
2. Arithmetic operations
3. Multiply and divide
4. Logical operations
5. Data testing and bit manipulation
6. Shifts and rotates
Almost all MCU activities involve transferring data from memories or peripherals into the CPU or transferring results from the CPU into memory or I/O devices.

### Table 6-1. Load, Store, and Transfer Instructions

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>IMM</th>
<th>DIR</th>
<th>EXT</th>
<th>INDX</th>
<th>INDY</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Memory Byte</td>
<td>CLR</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Clear Accumulator A</td>
<td>CLRA</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear Accumulator B</td>
<td>CLRB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Load Accumulator A</td>
<td>LDAA</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Load Accumulator B</td>
<td>LDAB</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Load Double Accumulator D</td>
<td>LDD</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Pull A from Stack</td>
<td>PULA</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull B from Stack</td>
<td>PULB</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push A onto Stack</td>
<td>PSHA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Push B onto Stack</td>
<td>PSHB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Store Accumulator A</td>
<td>STAA</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Store Accumulator B</td>
<td>STAB</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Store Double Accumulator D</td>
<td>STD</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Transfer A to B</td>
<td>TAB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Transfer A to CCR</td>
<td>TAP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Transfer B to A</td>
<td>TBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Transfer CCR to A</td>
<td>TPA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Exchange D with X</td>
<td>XGDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Exchange D with Y</td>
<td>EGDY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
**HC11 Instructions**

This group of instructions supports arithmetic operations on a variety of operands; 8- and 16-bit operations are supported directly and can easily be extended to support multiple-word operands. Two's complement (signed) and binary (unsigned) operations are supported directly.

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>IMM</th>
<th>DIR</th>
<th>EXT</th>
<th>INDX</th>
<th>INDY</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Accumulators</td>
<td>ABA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Add Accumulator B to X</td>
<td>ABX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Add Accumulator B to Y</td>
<td>ABY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Add with Carry to A</td>
<td>ADCA</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Add with Carry to B</td>
<td>ADCB</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Add Memory to A</td>
<td>ADDA</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Add Memory to B</td>
<td>ADDB</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Add Memory to D (16 Bit)</td>
<td>ADDD</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Compare A to B</td>
<td>CBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Compare A to Memory</td>
<td>CMPA</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Compare B to Memory</td>
<td>CMPB</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Compare D to Memory (16 Bit)</td>
<td>CPD</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Decimal Adjust A (for BCD)</td>
<td>DAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Decrement Memory Byte</td>
<td>DEC</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Decrement Accumulator A</td>
<td>DECA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Decrement Accumulator B</td>
<td>DECB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Increment Memory Byte</td>
<td>INC</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Increment Accumulator A</td>
<td>INCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Increment Accumulator B</td>
<td>INCB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
**HC11 Instructions**

Compare instructions perform a subtract within the CPU to update the condition code bits without altering either operand. Although test instructions are provided, they are seldom needed since almost all other operations automatically update the condition code bits.

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>IMM</th>
<th>DIR</th>
<th>EXT</th>
<th>INDX</th>
<th>INDY</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twos Complement Memory Byte</td>
<td>NEG</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Twos Complement Accumulator A</td>
<td>NEGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Twos Complement Accumulator B</td>
<td>NEGB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtract with Carry from A</td>
<td>SBCA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Subtract with Carry from B</td>
<td>SBCB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtract Memory from A</td>
<td>SUBA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtract Memory from B</td>
<td>SUBB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtract Memory from D (16 Bit)</td>
<td>SUBD</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test for Zero or Minus</td>
<td>TST</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Test for Zero or Minus A</td>
<td>TSTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test for Zero or Minus B</td>
<td>TSTB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HC11 Instructions

One multiply and two divide instructions are provided. The 8-bit by 8-bit multiply produces a 16-bit result. The integer divide (IDIV) performs a 16-bit by 16-bit divide, producing a 16-bit result and a 16-bit remainder. The fractional divide (FDIV) divides a 16-bit numerator by a larger 16-bit denominator, producing a 16-bit result (a binary weighted fraction between 0 and 0.99998) and a 16-bit remainder.

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply (A \times B \Rightarrow D)</td>
<td>MUL</td>
<td>X</td>
</tr>
<tr>
<td>Fractional Divide (D ÷ X \Rightarrow X; r \Rightarrow D)</td>
<td>FDIV</td>
<td>X</td>
</tr>
<tr>
<td>Integer Divide (D ÷ X \Rightarrow X; r \Rightarrow D)</td>
<td>IDIV</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 6-3. Multiply and Divide Instructions
HC11 Instructions

Table 6-4. Logical Operation Instructions

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>IMM</th>
<th>DIR</th>
<th>EXT</th>
<th>INDX</th>
<th>INDY</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND A with Memory</td>
<td>ANDA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>AND B with Memory</td>
<td>ANDB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Bit(s) Test A with Memory</td>
<td>BITA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit(s) Test B with Memory</td>
<td>BITB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One's Complement Memory Byte</td>
<td>COM</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One's Complement A</td>
<td>COMA</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>One's Complement B</td>
<td>COMB</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OR A with Memory (Exclusive)</td>
<td>EORA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OR B with Memory (Exclusive)</td>
<td>EORB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR A with Memory (Inclusive)</td>
<td>ORAA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR B with Memory (Inclusive)</td>
<td>ORAB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This group of instructions is used to perform the Boolean logical operations AND, inclusive OR, exclusive OR, and one’s complement.
HC11 Instructions

This group of instructions is used to operate on operands as small as a single bit, but these instructions can also operate on any combination of bits within any 8-bit location in the 64-Kbyte memory space.

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>IMM</th>
<th>DIR</th>
<th>EXT</th>
<th>INDX</th>
<th>INDY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit(s) Test A with Memory</td>
<td>BITA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Bit(s) Test B with Memory</td>
<td>BITB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Clear Bit(s) in Memory</td>
<td>BCLR</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Bit(s) in Memory</td>
<td>BSET</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch if Bit(s) Clear</td>
<td>BRCLR</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch if Bit(s) Set</td>
<td>BRSET</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
All the shift and rotate functions in the M68HC11 CPU involve the carry bit in the CCR in addition to the 8- or 16-bit operand in the instruction, which permits easy extension to multiple-word operands.
Stack and Index Register Instructions
### HC11 Instructions

This table summarizes the instructions available for the 16-bit index registers (X and Y) and the 16-bit stack pointer.

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>IMM</th>
<th>DIR</th>
<th>EXT</th>
<th>INDEX</th>
<th>INDY</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Accumulator B to X</td>
<td>ABX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add Accumulator B to Y</td>
<td>ABY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare X to Memory (16 Bit)</td>
<td>CPX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Compare Y to Memory (16 Bit)</td>
<td>CPY</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Decrement Stack Pointer</td>
<td>DES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decrement Index Register X</td>
<td>DEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decrement Index Register Y</td>
<td>DEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increment Stack Pointer</td>
<td>INS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increment Index Register X</td>
<td>INX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increment Index Register Y</td>
<td>INY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Index Register X</td>
<td>LDX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Load Index Register Y</td>
<td>LDY</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Load Stack Pointer</td>
<td>LDS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Pull X from Stack</td>
<td>PULX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Y from Stack</td>
<td>PULY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push X onto Stack</td>
<td>PSHX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Y onto Stack</td>
<td>PSHY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store Index Register X</td>
<td>STX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Store Index Register Y</td>
<td>STY</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Store Stack Pointer</td>
<td>STS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Transfer SP to X</td>
<td>TSX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer SP to Y</td>
<td>TSY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer X to SP</td>
<td>TXS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer Y to SP</td>
<td>TYS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exchange D with X</td>
<td>XGDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exchange D with Y</td>
<td>XGDY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HC11 Instructions

Condition Code Register Instructions

These instructions allow a programmer to manipulate bits of the CCR.

Table 6-8. Condition Code Register Instructions

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Carry Bit</td>
<td>CLC</td>
<td>X</td>
</tr>
<tr>
<td>Clear Interrupt Mask Bit</td>
<td>CLI</td>
<td>X</td>
</tr>
<tr>
<td>Clear Overflow Bit</td>
<td>CLV</td>
<td>X</td>
</tr>
<tr>
<td>Set Carry Bit</td>
<td>SEC</td>
<td>X</td>
</tr>
<tr>
<td>Set Interrupt Mask Bit</td>
<td>SEI</td>
<td>X</td>
</tr>
<tr>
<td>Set Overflow Bit</td>
<td>SEV</td>
<td>X</td>
</tr>
<tr>
<td>Transfer A to CCR</td>
<td>TAP</td>
<td>X</td>
</tr>
<tr>
<td>Transfer CCR to A</td>
<td>TPA</td>
<td>X</td>
</tr>
</tbody>
</table>
Program Control Instructions

1. Branches
2. Jumps
3. Subroutine calls and returns
4. Interrupt handling
5. Miscellaneous
These instructions allow the CPU to make decisions based on the contents of the condition code bits. All decision blocks in a flow chart would correspond to one of the conditional branch instructions summarized here.
HC11 Instructions

The jump instruction allows control to be passed to any address in the 64-Kbyte memory map.

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>DIR</th>
<th>EXT</th>
<th>INDX</th>
<th>INDY</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump</td>
<td>JMP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
These instructions provide an easy way to divide a programming task into manageable blocks called subroutines. The CPU automates the process of remembering the address in the main program where processing should resume after the subroutine is finished. This address is automatically pushed onto the stack when the subroutine is called and is pulled off the stack during the RTS instruction that ends the subroutine.
This group of instructions is related to interrupt operations, we will get to there use later.
**HC11 Instructions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Operation (2-cycle delay)</td>
<td>NOP</td>
<td>X</td>
</tr>
<tr>
<td>Stop Clocks</td>
<td>STOP</td>
<td>X</td>
</tr>
<tr>
<td>Test</td>
<td>TEST</td>
<td>X</td>
</tr>
</tbody>
</table>

NOP is a do nothing instruction, just wastes an instruction cycle. STOP is used to put the CPU into a low power mode. TEST is a reserved instruction only used at the factory when making the chips.
“C” Conversions

“C” Addition

c = x + y

HC11 Assembly

// Integer variables
ldd x
addd y
std c

// Character variables
ldab x
addb y
stab c
“C” Conversions

“C” Addition

\[ c = x + y \]

HC11 Assembly

// c goes in double acc
// x, y are the index regs
pshx
xgdx
pshy
tsx
add 0,x
puly
pulx

“C” Addition

\[ x = x + y \]

HC11 Assembly

// Using character registers
// acc a = x, acc b = y.
aba
“C” Conversions

“C” Multiplication

// c = short variable
// a, b = char variables
c = a * b

HC11 Assembly

ldaa a
ldab b
mul
std c
"C" if statement:  "HC11" Assembly:

if(a > 10) y++;

cmpa #10 ble endif

endif:
“C” if/else statement:

```c
char a; short y;
if(a > 10)
    y++;
else
    y--;
```

“HC11” Assembly:

```assembly
cmpa #10
bgt if
dey
bra endif
if:
    iny
endif:
```
“C” Conversions

“C” if/else statement:

```c
char foo;
short qux;
if(foo > 10)
    qux++;
else
    qux--;
```

“HC11” Assembly:

```assembly
ldaa foo
cmpa #10
bgt if
ldy qux
iny
sty qux
bra endif
if:
    ldy qux
    iny
    sty qux
endif:
```
“C” Conversions

“C” while statement:

```
char a, b;
while(a > 20)
{
    a--; b *= 3;
}
```

“HC11” Assembly:

```
while:
    cmpa #20
    ble endwhile
de
    psha
    ldaa #3
    mul
    pula

    bra while
endwhile:
```
“C” if/else statement:

```c
char foo;
short qux;
if(foo > 10)
    qux++;
else
    qux--;
```

“HC11” Assembly:

```assembly
ldaa foo
cmpa #10
bgt if
ldy qux
iny
sty qux
bra endif
if:
ldy qux
iny
sty qux
endif:
```
"C" do/while statement:

```c
int foo, bar, qux;
do {
    foo -= 3;
    bar += qux;
} while (foo > 7)
```

"HC11" Assembly:

```assembly
do:
    ldd foo
    subd #3
    std foo
    ldd bar
    addd qux
    std bar
    ldd foo
    cpd #7
    bgt do
```
HC11 Example

Summing the first 10 elements of an integer array, result in x:

array: .space 20

ldx #0
ldab #0
while:
cmpb #18
bgt endwhile
ldy #array
aby

xgdx
add 0,y
xgdx
incb
incb
bra while endwhile:
HC11 Example

/***********************************************************
*     Program 1.
*     A simple program to introduce the 68HC11 environment
***********************************************************/

#include <v2_18g3.asm>  // sets stuff up for you, especially I/O

.sect .data

SIGNON:     .asciz   "CMPE12C Simulator"
PROMPT:     .asciz   ">

// (continued...)
Your program starts where indicated by the label "main". After startup code in v2_18g3.asm is done, it jumps to this label and continues running.

```
.text

main:
    ldx #SIGNON
    jsr OUTSTRING

loop:
    ldx #PROMPT
    jsr OUTSTRING
    jsr GETCHAR
    jsr OUTCHAR
    jsr OUTCRLF
    jmp loop
```
HC11 Micro Kit Usage

• Design environment:
  • Edit & assemble on CATS machines
    > hc11build file.asm
  • Download program to the PC
  • Upload to the microkit using serial cable

• To use kits at home
  • 9-12V AC adapter
  • Serial download cable
  • Connection to CATS machines
  • Free serial communication software (e.g. TeraTerm)
Always #include <v2_18g3.asm> as part of your program

• Sets up memory, including stack pointer
• Jumps to your “main” label
• Also includes a basic library of I/O routines
• Take a look at it!

http://www.soe.ucsc.edu/classes/cmpe012c/Fall03/

In the handouts and links page under HC11 Stuff
Example recursive routine

Fibonacci on the switches and LEDs

/* Input: A. Returns A’th Fibonacci term in accumulator A */
fib:   cmpa  #2
       bgt  fibnot1
       ldaa  #1     // return 1 if n <= 2
       rts

fibnot1:  pshb     // caller save protocol
          deca
          psha     // save n-1
          deca     // call with n-2
          jsr  fib
          tab      // transfer F(n-2) to b
          pula     // a gets n-1
          jsr  fib
          aba     // add accumulators
          pulb     // restore B
          rts

main:    ldx  #SWITCHES
       ldaa  0,x
       jsr  fib
       ldx  #LEDS
       staa  0,x
       jmp  main

Cyrus Bazeghi
Microcontroller/HC11 Summary

- Microcontrollers are great little processors for I/O and data manipulation
- The CISC-style programming makes assembly programming easier
- Variable instruction length can be a problem for
  - Word-aligned machines
  - Super-scalar machines
  - Pipelines
- The RISC philosophy is to
  - Use simple instructions and let the compiler optimize
  - Use loads and stores to support higher-speed registers
- Neither RISC, its CISC predecessors, nor CISCY RISCY has the definitive advantage
  - Depends on application, architecture, and implementation.