Microcontrollers

–and–

Motorola/Freescale

HC11 ISA
What is a microcontroller?

• A computer on a chip used to control electronic devices
• A microprocessor
  • Usually not cutting edge (4-bit to 32-bit)
  • 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, RAM 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)
- Apple 4s – ARM A9MP Core + GPU, etc. as Package on Package(?)
How prevalent are microcontrollers?

1. In 2006 over 4 billion 8-bit microcontrollers sold.
2. 55% of all CPUs sold are 8-bit microcontrollers/microprocessors.
3. Mid-priced car has about 30 MCs.
4. Typical US home has about 3 dozen microcontrollers (mostly in appliances)
Microcontrollers (1/4)

- **AMCC**
  - Until May 2004, these μCs were developed and marketed by IBM, whose
  - 4xx family was sold to Applied Micro Circuits Corporation.
  - **403 PowerPC CPU**
    - PPC 403GCX
  - **405 PowerPC CPU**
    - PPC 405EP
    - PPC 405GP/CR
    - PPC 405GPr
    - PPC NPe405H/L
  - **440 PowerPC Book E CPU**
    - PPC 440GP
    - PPC 440GX
    - PPC 440EP/EPx/GRx
    - PPC 440SP/SPe

- **Altera**
  - Nios II 32-bit configurable soft processor
  - Nios 16-bit configurable soft processor

- **Atmel**
  - AT89 series (Intel 8051 architecture)
  - AT90, ATtiny, ATMega series (AVR architecture) (Atmel Norway design)
  - AT91SAM (ARM architecture)
  - AVR32 (32-bit AVR architecture)

- **Charmed Labs**
- **Cypress MicroSystems**
- **Dallas Semiconductor**
Microcontrollers (2/4)

- Freescale Semiconductor
  - Until 2004, these µCs were developed and marketed by Motorola, whose semiconductor division was spun-off to establish Freescale.
  - 8-bit
    - 68HC05 (CPU05)
    - 68HC08 (CPU08)
    - 68HC11 (CPU11)
  - 16-bit
    - 68HC12 (CPU12)
    - 68HC16 (CPU16)
    - Freescale DSP56800 (DSPcontroller)
  - 32-bit
    - Freescale 683XX
    - MPC500
    - MPC 860 (PowerQUICC)
    - MPC 8240/8250 (PowerQUICC II)
    - MPC 8540/8555/8560 (PowerQUICC III)

- Fujitsu
- Holtek
- Infineon
  - 8-bit
    - XC800 family
    - C500/C800 family
  - 16-bit
    - XC166 family
  - 32-bit family
    - TRICORE family
Microcontrollers (3/4)

- **Intel**
  - 8-bit
    - **MCS-48** (8048 family – also incl. 8035, 8038, 8039, 8040, 8X42, 8X49, 8050; X=0 or 7)
    - **MCS-51** (8051 family – also incl. 8X31, 8X32, 8X52; X=0, 3, or 7)
    - 8xC251
  - 16-bit
    - **MCS-96** (8096 family – also incl. 8091)
    - Intel MCS 296
- **Lattice Semiconductor**
  - Mico8 softcore 8 bit microcontroller
- **Microchip Technology**
  - 8 and 16-bit microcontrollers with 12 to 24-bit instructions
  - ability to include DSP function
  - 12-bit instruction PIC
  - 14-bit instruction PIC
    - **PIC16F84**
  - 16-bit instruction PIC
- **National Semiconductor**
  - COP's
  - CR116
- **NEC**
- **Philips Semiconductors**
- **Rabbit Semiconductor**
Microcontrollers (4/4)

- **Renesas Technology**
- (Renesas is a joint venture of Hitachi and Mitsubishi Electric.)
- **Silabs**
- **Silicon Motion**
- **STMicroelectronics**
- **Texas Instruments**
- **Toshiba**
- **Western Design Center**
- **Ubcicom**
- **Xemics**
- **Xilinx**
- **Zilog**
Microcontroller for iPhone
ARM1176JZF 32-bit processor
## Languages for microcontrollers

<table>
<thead>
<tr>
<th>Language</th>
<th>‘98-’99</th>
<th>‘99-’00</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/Freescale M68HC11

- M6801 CPU core
- Memory:
  - 8KB ROM
  - 512B EEPROM
  - 256B RAM
- Counter/Timer system
- 8-channel, 8-bit A/D conv.
  (external D/A in our kit)
- Parallel I/O port
- Two serial I/O ports:
  - Asynchronous, SCI
  - Synchronous, SPI
- Expansion bus for external memory
**High-level HC11 architecture**

![Diagram showing the components of a HC11 architecture: Interrupt logic, MEMORY (CPU core), TIMER and COUNTER, A/D converter, Serial I/O, PORT A, PORT B, PORT C, PORT D, PORT E.]}
M8601 CPU core

• Mixed 8-bit/16-bit architecture
• CISC, with more than 100 opcodes
• Accumulator architecture
• Two 8-bit accumulators, $A$ or $B$
• One 16-bit accumulator $[A:B]$, called $D$
• Two 16-bit index registers, $X$ and $Y$, used for addressing memory or for counting
• 16-bit dedicated stack pointer, $SP$
• 16-bit program counter, $PC$
• 8-bit condition codes register, $CCR$
Accumulator architecture
M8601 registers

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
Accumulators: A, B, and D

- General-purpose 8-bit accumulators
- Source/destination of most operations
- Almost always interchangeable – ex:
  - **ABA**, add B to A: A ← A + B, B is unchanged
    (note that there’s no AAB instruction)
  - **SBA**, subtract B from A: A ← A − B
    (note that there is no SAB instruction)
- Some instructions use \([A:B]\) as a single 16-bit register D
### Index registers $X$ and $Y$

<table>
<thead>
<tr>
<th>Bit</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>$X$</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>$Y$</td>
<td>0</td>
</tr>
</tbody>
</table>

- 16-bit index registers $X$ and $Y$:
  - Used for *indexed addressing mode*
  - Swap with register $D$ to perform complex address operations
HC11 User Stack

- The HC11 has a stack set up at start-up
- Memory is byte-addressable

Examples:

- **PSHA** // pushes accumulator A onto stack
  // SP = SP – 1 since a byte size
- **PSHY** // pushes index register Y onto stack
  // SP = SP – 2 since a word size
- **PULA** // Pops off a byte from the stack into A
  // SP = SP + 1
- **PULY** // Pops off a word from the stack into Y
  // SP = SP + 2
Stack pointer: \( SP \)

- 16-bit stack pointer, \( SP \)
  - automatically incremented or decremented on pushes and pops
  - can be used for data and return addresses
  - return address for subroutines is automatically pushed/popped with \texttt{jsr/rts}
Program counter: $PC$

- 16-bit program counter, $PC$
Condition code register: CCR

- Five status indicators
  - H, half carry (used only for BCD operations)
  - N, negative
  - Z, zero
  - V, overflow (two’s complement)
  - C, carry/borrow
    many instructions set them all, but not all instructions
- Two interrupt masking bits
  - I, disable all maskable interrupts (1 = disable)
  - X, disable external interrupts (1 = disable)
- A STOP-disable bit disables the STOP instruction
HC11 assembly

- Instructions:

  `<Label:> [<Mnemonic> [<Operand>]] // Comment`

- **ALL labels end with a colon. Ex:**
  ```
  LOOP:
  ```

  Labels are case-sensitive, instructions are not.

- Comments
  ```
  /* */  C-style comment
  //     C++-style line comment
  ```
HC11 assembler directives

C-like directives:

`#include` to include a file

and

`#define` to define a constant

```
#include <v2_18g3.asm>
#define MAX 10000
```
HC11 assembler directives

- Separate sections for data declarations and for instructions
  
  `.sect .data` for data sections  
  `.sect .text` for instructions sections.

- Data types
  
  `.space` for array declarations  
  `.byte` declares a single byte  
  `.word` declares a 16-bit word  
  `.asciz` a 0-terminated string of characters  
  `.ascii` a string, no NULL terminator
Data declaration examples

.sect .data
    myvar:    .word   0x1000  // init a word
                 // LSB in highest
                 // address
    myarray:  .space   20  // 20 bytes
    mychar:   .byte    'a'  // a character
    mybyte:   .byte    23  // decimal integer
    myhi:     .asciz   "Hi"  // a string + \0
    myhi2:    .ascii   "Hi"  // a string no \0

.sect .text
    // Instructions would start here.
    // Can have many .text and .data sections.

NOTE: must initialize or the assembler won't really allocate!
HC11 addressing modes

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

• 1 or 2 byte immediate depending on register involved (LDAA vs. LDD)
• **Use # prefix**
• Several formats for different bases -- C-style constants instead of what is in the HC11 manual (don’t use !,$,@,%)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA</td>
<td>#245</td>
<td>// decimal</td>
</tr>
<tr>
<td>LDAA</td>
<td>#0x61</td>
<td>// hex</td>
</tr>
<tr>
<td>LDAA</td>
<td>#041</td>
<td>// octal</td>
</tr>
<tr>
<td>LDAA</td>
<td>#0b11000011</td>
<td>// bin</td>
</tr>
<tr>
<td>LDAA</td>
<td>’a’</td>
<td>// ascii</td>
</tr>
</tbody>
</table>
Direct addressing

- Access an absolute memory location
- 8-bit addresses – 256 bytes
- Premium addressing space: only 1 byte required!
- Ex:

```assembly
// Your data starts at address 0x0040:
.sect .data
var: .word 0x1000 // Allocate/init a word
    // Note: a word is 16 bits!
    // 10 in address 0x40 and
    // 00 in 0x41.

.sect .text
SUBD    var    // Subtract M[0x0040] from D
ADDA    0x11   // ?
ADDA    #0x11  // ?
```
Extended addressing

• Access an absolute memory location
• Essentially the same mode as direct, but with 16-bit (enhanced or extended) addresses
• The assembler will decide on which to use based on where the reference is located. Ex:

```asm
// Your data starts at address 0x4000:
.sect .data
var: .word 0xAA12 // Allocate/initialize a word
        // Note: a word is 16 bits!
.sect .text
SUBD var  // Subtract M[x4000] from D
SUBD 0x4230 // Subtract M[x4230] from D
```
Indexed addressing

• Uses index register $X$ or $Y$
• $EA = X$ (or $Y$) + offset
• Similar to LC-3’s `LDR R0, R2, #3`
• Offset precedes $X$ or $Y$
• can access memory and perform operation all at once.

NOTE: the offset is 1 byte, unsigned: 0-255 only

```c
#define mydef 4     // c preprocessor used
ldx #var         // Like LC-3 load address
addd 0, X        // add contents of M[0+X] to D
                 // (Note “addd X” doesn’t work)
addd 2, X         // add contents of M[2+X] to D
addd mydef*2, X   // add contents of M[8+X] to D
```
Inherent addressing

• Opcode fully specifies operands
• Examples:

  **INCB**  // increment accumulator B
  **ASLA**  // arithmetic shift left accumulator A
  **PSHY**  // push index register Y onto stack
Relative addressing

- Used only for branch instructions, set by the assembler
- One-byte offset, two’s complement
- Offsets from −128 to +127 bytes, with respect to the incremented program counter (PC+2)
- Use jumps for longer branches

```
THERE:   BRA   WHERE
WHERE:   BEQ   HERE
HERE:    BHI   HERE  // possibly hangs
```
HC11 addressing modes review

```
foo: .byte 0xAB
...
ldab #0       // loads the number 0 into b
ldaa foo      // loads the contents of byte foo into acc. A
ldy #foo      // loads the address of foo into Y
ldab 0, x     // loads the byte at address X+0 into acc. B
```

NOTE that many instructions can be used with different addressing modes
**HC11 Kit Memory Map**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 - 0x00FF</td>
<td>Internal SRAM</td>
</tr>
<tr>
<td>0x00FF</td>
<td>STACK</td>
</tr>
<tr>
<td>0x0100</td>
<td>4k System Memory</td>
</tr>
<tr>
<td>0x0FFF</td>
<td>User Area</td>
</tr>
<tr>
<td>0x1000</td>
<td>User Interrupt vector jump table</td>
</tr>
<tr>
<td>0x7BC0 - 0x7BC1</td>
<td>I/O ports</td>
</tr>
<tr>
<td>0x7BFF</td>
<td>I/O ports</td>
</tr>
<tr>
<td>0x7C00 - 0x7FFF</td>
<td>Internal 64-byte register block</td>
</tr>
<tr>
<td>0x8000 - 0x8040</td>
<td>External ROM (system code)</td>
</tr>
<tr>
<td>0xFFFF</td>
<td></td>
</tr>
</tbody>
</table>
HC11 Architecture

* $V_{PPE}$ applies only to devices with PROM or OTPROM.
Instruction Set

• Description of all operations listed in the “Programming Reference Guide” available on course website
  – See ‘Instruction Set’ Section