EEL 4744C: Microprocessor Applications
Lecture 2
Programming Model, Address Mode, HC12 Hardware Introduction

Reading Assignment

- Microcontrollers and Microcomputers: Chapter 3, Chapter 4
- Software and Hardware Engineering: Chapter 2
  Or
- Software and Hardware Engineering: Chapter 4
  Plus
  - CPU12 Reference Manual: Chapter 3
  - M68HC12B Family Data Sheet: Chapter 1, 2, 3, 4

CPU Registers

- **Accumulators**
  - Registers that accumulate answers, e.g. the A Register
  - Can work simultaneously as the source register for one operand and the destination register for ALU operations

- **General-purpose registers**
  - Registers that hold data, work as source and destination register for data transfers and source for ALU operations

- **Doubled registers**
  - An N-bit CPU in general uses N-bit data registers
  - Sometimes 2 of the N-bit registers are used together to double the number of bits, thus “doubled” registers

CPU Registers (2)

- **Pointer registers**
  - Registers that address memory by pointing to specific memory locations that hold the needed data
  - Contain memory addresses (without offset)

- **Stack pointer registers**
  - Pointer registers dedicated to variable data and return address storage in subroutine calls

- **Index registers**
  - Also used to address memory
  - An effective memory address is found by adding an offset to the content of the involved index register

CPU Registers (3)

- **Segment registers**
  - In some architectures, memory addressing requires that the physical address be specified in 2 parts
    - Segment part: specifies a memory page
    - Offset part: specifies a particular place in the page

- **Condition code registers**
  - Also called flag or status registers
  - Hold condition code bits generated when instructions are executed, e.g. overflow in ADD
Register Transfers

- MOV A, B
  - B is the source, A is the destination
- ADD A, B
  - Add (A) to (B), then transfer the answer from ALU to A
- Register transfer language
  - Register name in ( ) means “content” of the register
  - “−” means replacement, e.g. (A) − (B)

Condition Code Register (CCR)

- Also called flag or status registers
- Contain bits that are set or reset due to instructions
  - ALU, load or move can all modify the CCR
- Most processors also provide instructions that modify the CCR directly
- What are the “bit” in the CCR?

Condition Code Register (2)

- Carry bit
  - Set to 1 if there is a carry/borrow out of the most significant bit during an ADD or SUB
- Overflow
  - The result is too large to be represented by available bits
- Underflow
  - The result is too small to be represented by available bits
- Examples: 10010011 (147\text{ hex}) +/− 10110011 (179\text{ hex})
  - ADD gives overflow, SUB gives underflow, setting the carry (or borrow) bit to 1

Condition Code Register (3)

- Carry bit for multiple-byte ADD/SUB
  - Carry from the less significant bytes is added into the ADD/SUB of the next more significant bytes
- Example: 00110010 11001001 + 00011011 10110110
  - A carry of 1 from the LSB’s is added into the MSB’s
- What if we try to add two numbers that are encoded in 2’s-complement bits?!
  - Example: 10010011 (-109\text{ dec}) +/− 10110011 (-77\text{ dec})
    - Give 1 01000110 (-186\text{ dec}) and 1 11100000 (-32\text{ dec})
    - The ADD results in overflow, the SUB has no problem

Condition Code Register (4)

- Conclusion: carry/borrow bit can not be used to indicate overflow/underflow for 2’s-complement!
- A separate bit is needed to indicate overflow for 2’s-complement numbers
  - Overflow occurs if the 2 operands have the same sign AND the result is of different sign
  - Overflow cannot occur if the 2 operands have opposite signs
  - Which one is the sign bit? The most significant bit, not the carry/borrow bit

Condition Code Register (5)

- Example: 10010011 (-109\text{ dec}) +/− 10110011 (-77\text{ dec})
  - Produce 1 01000110 (-186\text{ dec}) and 1 11100000 (-32\text{ dec})
  - In 2’s-complement number encoding, the ADD had an overflow, the SUB was OK
- Examples: 10010011 (147\text{ hex}) +/− 10110011 (179\text{ hex})
  - Produce 1 01000110 (326\text{ hex}) and 1 11100000 (-32\text{ hex})
  - In unsigned binary encoding, the ADD had an overflow, the SUB had an underflow
- We notice that the bit patterns remain the same, also the hardware remain the same. The hardware can provide overflow flag bit for 2’s-complement
• **Sign bit**
  - Most significant bit of the number (not the carry bit)
  - Gives the sign only of signed number encoding is used

• **Zero bit**
  - Set to 1 if the result of an operation equals to zero
  - Otherwise it is reset to 0 or false

• **Parity bit**
  - Even-parity: set if result has an even number of 1's
  - Odd-parity: set if result has an odd number of 1's
  - Useful for checking errors in long-haul data transmission
  - Parity-even/parity-odd works with conditional branches

• **Carry: C**
• **Overflow: V**
• **Sign: S**
• **Zero: Z**
• **Parity: P**

**Using the CCR**
- CCR is attached to the sequence controller
- For use by the conditional branch instructions
- Results of operations will set or reset C, V, S, or Z
- Conditional branch instructions checks C, V, S, or Z

**The Programmer’s Model**
- The set of registers the programmer can manipulate and must manage to program the processor
- Include accumulator, data registers, memory addressing registers, stack point registers, condition code registers, etc.
- Also include memory locations used for data storage

**Physical address**
- The actual address supplied to the memory
- Number of bits in physical address determines the maximum number of memory locations that can be addressed

**Segment address**
- Gives the location of a segment of memory (e.g. block, page, etc.) that is smaller than the full memory

**Offset address**
- One that is calculated from the start of a segment of memory

**Logical address**
- Used when the complete physical is not needed or possible
- The offset address is one kind of logical address

**Effective address**
- One that is calculated by the processor
- Can be a physical or a logical address

**Auto-increment/-decrement**
- Provide efficient addressing for stepping through data tables
- Registers that address memory tend to have these capabilities

---

**EEL 4744C: Microprocessor Applications**

**Lecture 2**

Part 2

**Addressing Modes**

---

**Addressing Terminology**

---

**Addressing Terminology (2)**
Addressing Terminology (3)

- **RAM**
  - Stands for random access memory
  - Memories that can be read from and written to
- **ROM**
  - Stands for read only memory
- **Memory map**
  - Shows which addresses are used for which purposes
  - May show which addresses contain ROM, RAM, and which have no memory installed at all
- **I/O map**
  - Similar to a memory map for I/O functions

Memory Architectures

- **Linear addressing**
  - Instructions specify the full physical address
  - Favored by Motorola processors
  - Motorola MC68020 used 32-bit addresses, giving a 4GB addressable memory size

- **ROM**
  - Stands for read only memory

Memory Architectures (2)

- **Segmented addressing**
  - Reduces the number of bits needed to specify an address
  - Favored by Intel processors with 16B to 64KB segments
  - Intel 8086 used 20-bit addresses, 16 bits for segment address and 16 bits for offset address

Memory Architectures (3)

- **Segmented addressing (continued)**
  - Physical address is computed by shifting the segment register contents left by 4 bits, then adding the offset
  - Allows efficient allocation of memory to code and data
  - Special techniques needed to exceed page boundary

Addressing Modes

- **Register addressing**
  - When operands are held in the registers, e.g. MOV A, B
  - Register addressing instructions are the fastest and use the fewest bits compared to others
  - Some call it inherent addressing also

Addressing Modes (2)

- **Direct (absolute) memory addressing**
  - Instruction specifies the address of the data
  - The data address (in the instruction) can be the full physical address, or an offset address, depending on the memory architecture in use
  - 2 variations: direct addressing, reduced direct addressing

Variable sized segments used in a segmented memory architecture
Addressing Modes (3)

- **Register indirect addressing**
  - Also called pointer register addressing
  - Instruction contains address of the register that contains the address of the needed data
  - A 2-level addressing mode

Addressing Modes (4)

- **Register indirect addressing with auto-increment and auto-decrement**
  - For stepping through a table of data using register indirect addressing mode
  - The register pointing to the data must be incremented/decremented
  - Pre- and post-incrementing/decrementing are available
  - Essentially the same as register indirect addressing except each pointer register can have its contents incremented or decremented

Addressing Modes (5)

- **Memory indirect addressing**
  - Instruction contains memory address of the address of the needed data; less efficient than register indirect addressing
  - Data address can be calculated and stored in memory before use, and can be changed while running the program

Addressing Modes (6)

- **Indexed addressing**
  - Finds a memory location based on an index
  - Instruction contains the starting address of the array, the index register contains the offset to reach the data being addressed

Addressing Modes (7)

- **Based addressing**
  - Index register has the starting address of the data table
  - Instruction specifies the index register (ID) and offset, not the full starting address of data table; useful for transferring data

Addressing Modes (8)

- **Relative addressing**
  - Effective address = (PC) + offset
  - Used for branching short distances in well-written programs
  - Offset is a 2's complement number to enable branch forward and backward
  - Offset is 0 for the address of Next Op Code
Addressing Modes (9)

- Bit addressing
  - Read or write 1 bit at a time, which is within a byte location
  - Instruction supplies the address of the byte, plus a mask to specify which bit within the byte is to be addressed
  - The instruction provides the address of the byte and the number of the bit to be accessed.

Addressing Modes (10)

- Based indexed addressing
  - Effective address = (base register) + (index register) + displacement. Intel 80x86 use this mode
- Relative addressing with index plus displacement
  - Effective address = (PC) + (index register) + displacement
- Stack addressing
  - Saves the return address when the program calls a subroutine
  - After fetching the jump op code and subroutine address, the PC is made to point to the next op code (the return address)
  - This return address is pushed onto the stack
  - The "return" instruction at the end of the subroutine pops the return address from the stack, incrementing the stack pointer

Addressing Modes (11)

- Push operations

Addressing Modes (12)

- Pull (Pop) operations

Addressing Modes (13)

- Subroutine call and return
  - Program execution resumes from here after subroutine return (Return address)

Addressing Modes (14)

- Subroutine call and return
  - Return address is pushed onto stack before branching to the subroutine
  - The return instruction at the end of subroutine pops return address back to PC
An Overview of M68HC12

- **HCMOS microcontroller family**
- **16-bit machine**
  - 16-bit data and address buses $\Rightarrow$ 64KB address space
  - May also operate with 8-bit data bus
  - Contains CPU (register, ALU), memory (RAM, EEPROM, and Flash), timer section, and variety of I/O features
- **Many devices in this family, which vary by on-board features**
  - We’ll focus on MC68HC812A4 and MC68HC912B32
  - B32 device is used in lab board
  - B32 best suited for single-chip applications (32KB Flash EEPROM); A4 better suited for expanded-memory applications (w/ memory management system to address over 5MB)

M68HC12 B32 Block Diagram

- **8-bit accumulators (A,B)**
  - concatenation called D register
- **16-bit index registers (X,Y)**
  - primarily for indexed addressing, but also some arithmetic instructions
- **16-bit stack pointer (SP)**
  - after initialized, always points to last used memory location for a push operation
  - grows downward (i.e. toward start of address space)
  - automatically decremented for push and incremented for pop operations

Programming Model

- 8-bit condition code register w/ bits S, X, H, I, N, Z, V, C (used w/ conditional branching and some arithmetic operations)
  - C: carry/borrow
  - V: 2’s complement overflow
  - Z: zero
  - N: negative
  - H: half-carry/borrow (out of bit 3 in arithmetic operation)
  - I: interrupt mask to globally mask/unmask interrupt features
  - X: mask bit for non-maskable interrupt request (XIRQ*) pin; once unmasked, cannot be masked again until 68HC12 is reset
  - S: STOP disable bit, allows or disallows STOP instruction for low-power consumption
### Programming Model (4)

- **Data types**
  - Bit
  - 5-bit and 9-bit signed integers (only for offsets in indexed addressing)
  - 8-bit and 16-bit signed and unsigned integers
  - 16-bit effective addresses
  - 32-bit signed and unsigned integers (for extended div., mult., and mult.& accum. instructions)

- **Control registers**
  - 512 registers used to I/O data and control how CPU uses its I/O resources
  - Initially mapped to $0000$-$01FF$ in address space

### Operating Modes

- Determined by states of 3 signals/pins when device reset
- Two categories: *special modes* for greater access to protected control regs. & bits for sys. development; *normal modes* protect some control regs. & bits from accidental change
- Normal modes include:
  - *Normal single-chip* (Ports A-D used for general-purpose I/O)
  - *Normal expanded-narrow* (external 8-bit data bus and 16-bit address bus provided)
  - *Normal expanded-wide* (16-bit data and address buses provided)

### Memory Map

- On-chip memory types (RAM, etc.) and capacities determined by device type; location in map depends on operation mode
- With expanded mode, some memory may be located off-chip

### Addressing Modes

- **Inherent**
  - Immediate (8-bit or 16-bit operands)
  - Direct (8-bit operands, first 256 bytes of memory)
  - Extended (16-bit operands, full 64K address space)
  - Indexed (5-bit, 9-bit, or 16-bit signed offset; A, B, or D register offset; pre/post auto. inc/dec)
  - Indexed-indirect (16-bit or D register offset)
  - Relative (8-bit or 16-bit signed offset)

### Examples

**Inherent**

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>B8 7F</td>
<td>sta</td>
<td>A &lt;- B</td>
</tr>
<tr>
<td>0003</td>
<td>B7 81</td>
<td>exg a, b</td>
<td></td>
</tr>
</tbody>
</table>

**Immediate**

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>86 40</td>
<td>ldaa $40</td>
<td>Decimal 40 -&gt; A</td>
</tr>
<tr>
<td>0001</td>
<td>86 41</td>
<td>ldaa $41</td>
<td>Decimal 41 -&gt; A</td>
</tr>
<tr>
<td>0004</td>
<td>DE 1234</td>
<td>ldx $1234</td>
<td>Hexadecimal 1234 -&gt; X</td>
</tr>
</tbody>
</table>

**Direct**

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>86 64</td>
<td>ldaa $64</td>
<td>($064) -&gt; A</td>
</tr>
<tr>
<td>0002</td>
<td>86 65</td>
<td>ldaa $65</td>
<td>($065) -&gt; X</td>
</tr>
<tr>
<td>0004</td>
<td>DE 0A</td>
<td>ldx 10</td>
<td>($00A) -&gt; X</td>
</tr>
</tbody>
</table>

### Addressing Modes (Extended)

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>B6 1234</td>
<td>ldaa $1234</td>
<td>($1234) &gt; A</td>
</tr>
<tr>
<td>0003</td>
<td>FC 1234</td>
<td>lda $1234</td>
<td>($1234-1235) &gt; D</td>
</tr>
<tr>
<td>0006</td>
<td>7E C000</td>
<td>stx $C000</td>
<td>X -&gt; ($C000:C001)</td>
</tr>
</tbody>
</table>

Address range: full 64KB
Indexed Addressing

- Basic form is Operation Offset, Index_Reg, where:
  - Index_Reg is X, Y, SP, or PC
  - Offset is signed 5-, 9-, or 16-bit value to be added to index reg. contents to produce EA
  - Instruction varies in length (Opcode, Postbyte, [Offset], [Index_Reg]):
    - 5-bit offset fits in postbyte ⇒ 2-byte instruction
    - 9-bit offset requires an extra byte ⇒ 3-byte instruction
    - 16-bit offset requires 2 offset bytes ⇒ 4-byte instruction
  - Addition for EA is modulo 64K (wraps to $0000 after $FFFF)
  - Option for pre/post inc. or dec. (only reg. changed is index reg.)

Examples

- Indexed using constant offset
  
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 A6 00</td>
<td>ldax, X</td>
</tr>
<tr>
<td>0002 A6 00</td>
<td>ldax, 0.X</td>
</tr>
<tr>
<td>0004 A6 00</td>
<td>ldax, 0.0</td>
</tr>
<tr>
<td>0007 A6 E9 C0</td>
<td>ldax, -64.y</td>
</tr>
<tr>
<td>000A 6A 9F</td>
<td>ldax, -1.SP</td>
</tr>
<tr>
<td>000C A6 FA 1388</td>
<td>ldax, 5000,PC</td>
</tr>
</tbody>
</table>

- Indexed with automatic inc. and dec.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 A6 29</td>
<td>ldax, 7.X</td>
</tr>
<tr>
<td>0002 A6 3E</td>
<td>ldax, 2.X</td>
</tr>
<tr>
<td>0004 A6 20</td>
<td>ldax, 1.X</td>
</tr>
<tr>
<td>0006 A6 30</td>
<td>ldax, 1.X</td>
</tr>
</tbody>
</table>

Indexed Indirect Addressing

- Indexed addressing used first to find address of the data
  - That address then used to find the data (i.e. go to memory to find address, then to memory again to get data)
  - Two forms:
    - Operation [Offset, Index_Reg] : for constant, 16-bit offset.
    - Operation [D, Index_Reg] : to use D reg. for offset.

Examples

- Indexed-indirect w/ 16-bit constant offset
  
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 CE 5000</td>
<td>ldx #$5000 ; $5000 -&gt; X, initialize X</td>
</tr>
<tr>
<td>0003 CE 0064</td>
<td>ldax [B4.X]; (B$064) -&gt; A</td>
</tr>
<tr>
<td>0007 6A E3 FFFF</td>
<td>stax [1.X] ; A -&gt; ($4800)</td>
</tr>
</tbody>
</table>

- Index-indirect w/ D offset
  
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 CE 5000</td>
<td>ldx #$5000 ; $5000 -&gt; X, initialize X</td>
</tr>
<tr>
<td>0003 CC 0064</td>
<td>ldd #D64 ; D -&gt; D, initialize D</td>
</tr>
<tr>
<td>0006 EF 87</td>
<td>lds [D,X] ; ($5064) -&gt; SP</td>
</tr>
</tbody>
</table>

Relative Addressing

- Branch instructions use short (8-bit offset) or long (16-bit offset) relative addressing
  - By contrast, jump instructions use extended or indexed addressing
  - Also, loop primitive instructions that use 9-bit offsets (i.e. range –256 to +255)
  - NOTE: w/ relative addressing, offset calculated from location of next instruction in line
**Relative Addressing: Example**

<table>
<thead>
<tr>
<th>No.</th>
<th>Address</th>
<th>Instruction</th>
<th>Op Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000 00 02</td>
<td>bra</td>
<td>00</td>
<td>Forward branch (+2)</td>
</tr>
<tr>
<td>2</td>
<td>0002 A7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0003 A7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0004 22 FA</td>
<td>bhi</td>
<td>0000</td>
<td>Cond. branch back (+4)</td>
</tr>
<tr>
<td>5</td>
<td>0005 1500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0006 A+0100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>010A</td>
<td></td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>010A A7</td>
<td></td>
<td>01</td>
<td></td>
</tr>
</tbody>
</table>

**Reset Action on the 68HC12**

- When RESET* pin asserted
  - Some internal regs. and control bits forced to initial state
  - All regs. in pgm. model are indeterminate except I, X, and S bits in CC reg (they must be initialized!!)
  - Bits I and X set to mask interrupts (i.e. interrupts cannot occur until unmasked) since interrupt handling must be programmed before use, and S bit set to disable STOP mode
  - CPU fetches vector from $FFFFFF as address of 1st instr. to execute, stores to PC, and F&E begins

- Other consequences of reset
  - Allocates first 512B in memory map to control registers and 1kB RAM to $0000-$00FF
    - Both relocatable to any 2kB boundary if certain instructions executed after reset
    - EEPROM relocatable to any 4kB boundary depending upon mode
  - Bidirectional I/O lines configured as high impedance inputs (default)
  - BKGD, MODA, and MODB pins read to select operating mode
  - Timer system is reset (some regs. set to initial values, some indeterminate)
  - Serial I/O and A/D capabilities disabled (default)

- Causes of reset:
  - RESET* signal applied to device by external manual reset or low-voltage sensing circuit
  - Active-high reset signal ARST; uses same vector as RESET*
  - On-chip system for sensing clock oscillator stop or too slow, resets as before except using vector $FFFC:FFFD to allow special code for this event to execute
  - CPU Operating Properly (COP) watchdog timer generates reset with vector from $FFFF:FFFF if program does not keep timer from timing out; purpose to regain control if something goes wrong