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
EEL 4744C: Microprocessor Applications

Lecture 2

Part 1

CPU Registers and Control Codes
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?
• **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:** $10010011 \,(147_{10}) \, +/- \, 10110011 \,(179_{10})$
  – ADD gives *overflow*, SUB gives *underflow*, setting the carry (or borrow) bit to 1
• 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\textsubscript{10}) +/- 10110011 (-77\textsubscript{10})
  – Give 1 01000110 (-186\textsubscript{10}) and 1 11100000 (-32\textsubscript{10})
  – The ADD results in overflow, the SUB has no problem
• **Conclusion**: carry/borrow bit cannot 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
**Conditional Code Register (5)**

- **Example: 10010011 (-109_{10}) +/- 10110011 (-77_{10})**
  - Produce 1 01000110 (-186_{10}) and 1 11100000 (-32_{10})
  - In *2’s-complement* number encoding, the ADD had an overflow, the SUB was OK

- **Examples: 10010011 (147_{10}) +/- 10110011 (179_{10})**
  - Produce 1 01000110 (326_{10}) and 1 11100000 (-32_{10})
  - 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
**Condition Code Register (7)**

- **Symbols for flags**
  - 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
**Condition Code Register (8)**

- **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
EEL 4744C: Microprocessor Applications

Lecture 2

Part 2

Addressing Modes
Addressing Terminology

• **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
Addressing Terminology (2)

• **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
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
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

Segmented addressing memory map – Intel 8086
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

• Immediate addressing
  – Used for constant values know when the program is written
  – Data (i.e. constant values) can immediately follow the instruction
• **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**

---

**Diagram:**
- **Full address**:
  - Op Code
  - Data Adr H
  - Data Adr L
  - Data

- **Offset**:
  - CPU adds these
  - Address: $0000
  - The instruction specifies these
  - Address: $00FF

---

*256 Byte Base Page*
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 in-/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

Dr. Tao Li

• 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
• 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

Fixed offset between the 2 data storage locations

---

Dr. Tao Li
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.
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.
EEL 4744C: Microprocessor Applications

Lecture 2

Part 3

Introduction to M68HC12 Hardware
An Overview of M68HC12

• HCMOS microcontroller family

• 16-bit machine
  – 16-bit data and address buses ⇒ 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

- 32KB FLUSH EPROM
- A/D Converter
- Serial I/O Port
- Timer I/O Port
- Pulse Width Modulator
- Data Link Controller
- Interrupts & Control Signals
- General Purpose I/O or Multiplexed Address/Data Bus

Dr. Tao Li
• **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
• 16-bit program counter (PC)
  – usually not used directly by programmer
  – can be used as base register for some indexed addressing modes

<table>
<thead>
<tr>
<th>7</th>
<th>ACCUM. A</th>
<th>0</th>
<th>7</th>
<th>ACCUM. B</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>DOUBLE ACCUMULATOR D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>INDEX REGISTER X</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>INDEX REGISTER Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>STACK POINTER</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>PROGRAM COUNTER</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CONDITION CODES REGISTER
S X H I N Z V C

- CARRY
- OVERFLOW
- ZERO
- NEGATIVE
- I - INTERRUPT MASK
- HALF CARRY
- X - INTERRUPT MASK
- STOP
- DISABLE
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
• **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**
  
  1: 0000 1806 aba ; A + B -> A
  2: 0002 08 inx ; X+1 -> X
  3: 0003 B7 81 exg a, b ; A <-> B

- **Immediate**
  
  1: 0000 86 40 ldaa #64 ; Decimal 64 -> A
  2: 0002 86 64 ldaa #$64 ; Hexadecimal 64 -> A
  3: 0004 CE 1234 ldx #$1234 ; Hexadecimal 1234 -> X

- **Direct**
  
  1: 0000 96 64 ldaa $64 ; ($0064) -> A
  2: 0002 5B FF stab 255 ; B -> ($00FF)
  3: 0004 DE 0A ldx 10 ; ($000A:000B) -> X

**Address range: first 256 bytes**

Dr. Tao Li
# Examples

- **Extended**

1: 0000 B6 1234 ldaa $1234 ; ($1234) -> A
2: 0003 FC 1234 ldd $1234 ; ($1234:1235) -> D
3: 0006 7E C000 stx $c000 ; X -> ($C000:C001)

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], [Offset]):
    
    • 5-bit offset fits in postbyte $\Rightarrow$ 2-byte instruction
    
    • 9-bit offset requires an extra byte $\Rightarrow$ 3-byte instruction
    
    • 16-bit offset requires 2 offset bytes $\Rightarrow$ 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

1: 0000 A6 00      ldaa ,x
   ; (X+0) 5-bit offset -> A
2: 0002 A6 00      ldaa 0,x
   ; (X+0) 5-bit offset -> A
3: 0004 A6 E0 40    ldaa 64,x
   ; (X+64) 9-bit offset -> A
4: 0007 A6 E9 C0    ldaa -64,y
   ; (Y-64) 9-bit offset -> A
5: 000A 6A 9F       staa -1,SP
   ; A -> (SP-1) 5-bit offset
6: 000C A6 FA 1388  ldaa 5000,PC
   ; (PC+5000) 16-bit offset -> A

• Indexed with automatic inc. and dec.

   Opcode Operation_Value, +/- Index_Register +/-

1: 0000 A6 29      ldaa 7,-X
   ; X-7 -> X, (X) -> A ; Pre-decrement
2: 0002 A6 3E      ldaa 2,X-
   ; (X) -> A, X-2 -> X ; Post-decrement
3: 0004 A6 20      ldaa 1,+X
   ; X+1 -> X, (X) -> A ; Pre-increment
4: 0006 A6 30      ldaa 1,X+
   ; (X) -> A, X+1 -> X ; Post-increment
Examples

- Indexed with accumulator offset

1: 0000 A6 E5  ldaa B,X ; (X+B) -> A
2: 0002 E6 EC  ldab A,Y ; (Y+A) -> B
3: 0004 ED E6  ldy D,X  ; (X+D:X+D+1) -> Y

Replace constant with accumulator
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.

Indicate indirect addressing
Examples

- **Indexed-Indirect w/ 16-bit constant offset**
  
  1:  0000 CE 5000  \( \text{ldx} \)  \#$5000  ; $5000 -> X, Initialize X  
  2:  0003 A6 E3 0064  \( \text{ldaa} \)  \[\$64,X\]  ; ((\$5064)) -> A  
  3:  0007 6A E3 FFFF  \( \text{staa} \)  [-1,X]  ; A -> ((\$4fff))

- **Index-Indirect w/ D offset**
  
  1:  0000 CE 5000  \( \text{ldx} \)  \#$5000  ; $5000 -> X, initialize X  
  2:  0003 CC 0064  \( \text{ldd} \)  \#064  ; $0064 -> D, initialize D  
  3:  0006 EF E7  \( \text{lds} \)  [D,X]  ; ((\$5064)) -> SP

Content of X is not changed
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**

1: 0000 20 02  
   **THERE:**  
   bra

2: 0002 A7  
   **WHERE:**  
   nop

3: 0003 A7  
   nop

4: 0004 22 FA  
   **WHERE:**  
   bhi

5: 0006 1826 0100  
   **THERE:**  
   lbne

6: 000A +0100  
   **LONG_BRANCH:**  
   DS.B 256 ; Simulate space for instrs.

7: 010A
8: 010A A7

**BRA:** Branch Always

**BHI:** Branch if Higher

**LBNE:** Branch if Not Equal
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 $FFFF:FFFF$ as address of 1st instr. to execute, stores to PC, and F&E begins
Reset Action on the 68HC12

• Other consequences of reset

  – Allocates first 512B in memory map to control registers and 1KB RAM to $0800-$0BFF
    • 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)
Reset Action on the 68HC12

• 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 **$FFF:FFF** if program does not keep timer from timing out; purpose to regain control if something goes wrong