Structured Programming Process

1. Define Problem
   - what exactly should your program do?
2. Plan Solution = Develop Algorithm
   - sequence of computational or logical step to transform given inputs to desired outputs
   Options
   - top-down: start with goals, work down to required outputs
   - bottom-up: start with required outputs, work up to goals
   - divide & conquer
     - use top-down to break into tasks
     - use bottom-up to solve each task
     - best method for large programs
   Approaches for algorithm planning
   - Pseudocode: non-code list of tasks (example will follow)
   - Flowcharts: graphical map of algorithm flow (example follow)
3. Code Algorithm
   - ASM, C, higher level language
4. Simulate & Debug
5. Test in Hardware & Debug
**Program Structures**

**sequential**  
- start  
- \( ? \)  
- if true  
- else  
- end  

**conditional**  
- start  
- \( ? \)  
- \( \text{if true} \)  
- \( \text{else} \)  
- end  

**looping**  
- start  
- \( \text{repeat until} \)  
- end

**Assembly Implementations**

- start  
- \( ? \)  
- conditional branch  
- \( \text{always branch} \)  
- jump  
- subroutine  
- return  
- end

---

**Looping Concept**

**CONCEPT:** Consider need to sum 10 HW grades

**Option 1:** "Brute Force"

**pseudo code**

```
set sum = 0  
sum = sum + score 1  
sum = sum + score 2  
. . .  
sum = sum + score 10
```

**ASM code**

```
CLA  ; set sum to zero  
ADDA$1000 ; assume scores saved starting at mem addr $1000  
ADDA$1001  
. . .  
ADDA$1009  
STAA SUM  ; assume SUM = mem addr to store the sum
```

How many bytes in above ASM code? Approximately 3 x number_of_scores = \(~30\)

Code gets longer if we want to add more scores
Looping Concept

Option 2: Use a Loop

**pseudo code**

set sum = 0  
set count = 0  
while count < 10  
  sum = sum + score(count)  
  increment count  
(loop back to “while”)

**ASM code**

See “Simple Looping Code”

Can you identify the loop in this program?  
How many bytes of code in the loop?  
21, for an infinite number of scores

Simple Looping Code

**ASM code file, text**

; Loop example for Ch. 3 notes by A. Mason. Mar 09  
; Sums 10 values from memory & store result in SUM. Assumes 10  
; values to sum are stored at $1000  
; assumes prior sum stored at SUM

ORG $4000  
LIDX #$1000  ; set x to first memory address  
LDAB SUM  ; load starting sum into accB  
LDAA #$00  ; initialize counter to 0  
CHECK CMPA #$0A  ; ? added all 10?  
BEQ DONE  ; if yes, done  
ADDB 0,X  ; if no, add # to SUM  
INX  ; increment IX  
INCA  ; increment counter  
BRA CHECK  ; repeat loop  
DONE STAB SUM  ; store result  
SUM EQU $4400

**LST compiled file, text**

<table>
<thead>
<tr>
<th>Line</th>
<th>Addr</th>
<th>Op Code</th>
<th>Label</th>
<th>Mnemonic</th>
<th>Operand</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>=00004000</td>
<td>ORG</td>
<td>$4000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>4000 CE 1000</td>
<td>LDX</td>
<td>#$1000</td>
<td>load x to first mem addr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>4003 FE 4400</td>
<td>LDAB</td>
<td>SUM</td>
<td>load starting sum into accB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8:</td>
<td>4006 86 00</td>
<td>LDA X</td>
<td>#$00</td>
<td>set counter to 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>4008 81 0A</td>
<td>CHECK</td>
<td>CMPA</td>
<td>#$0A</td>
<td>if added all 10?</td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>400A 27 06</td>
<td>BRQ</td>
<td>DONE</td>
<td>if yes, done</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>400C EB 00</td>
<td>ADD X</td>
<td>$X</td>
<td>if no, add # to SUM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>400E 08</td>
<td>INX</td>
<td></td>
<td>increment IX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>400F 42</td>
<td>INCA</td>
<td></td>
<td>increment counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>4010 20 FE</td>
<td>BRA</td>
<td>CHECK</td>
<td>repeat loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>4012 7B 4400</td>
<td>DONE</td>
<td>STAB</td>
<td>SUM</td>
<td>store result</td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>=00004400</td>
<td>SUM</td>
<td>EQU</td>
<td>$4400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17:</td>
<td></td>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.S19 machine code record, binary**

S0030000FC
S1134000CE1000F644000FE0080010A2706EB00084221
S10925126FE8BF4000D2
S9030000FC

Looping Constructs

**For Loops**

- execute S | i2-i1 times, S = any set of instructions, I = counter
- see flowchart below
  
  For I = i1 to i2  
  Do S

**Do While Loops**

- execute S as long as C is TRUE, C = condition that will be false when done with S
- see flowchart below
  
  While C = true  
  Do S

**Repeat Until Loops**

- execute S until C is FALSE, C = condition that will be false when done with S
- functionally similar to Do While but with slightly different flow
- see flowchart below
  
  Repeat S,  
  Until C = false
ASM Examples

Discuss ASM Examples in HO_5, pg. 4-6

• Simple Arithmetic ASM Program Examples
  - Example 1: Write a program to add the numbers stored at memory locations $800, $801, and $802, and store the sum at memory location $900.
  - Example 2: Write a program to subtract the contents of the memory location at $805 from the sum of the memory locations at $800 and $802, and store the result at the memory location $900.
  - Example 3: Write a program to add two 16-bit numbers that are stored at $800~$801 and $802~$803, and store the sum at $900~$901.
  - Example 4: Write a program to subtract 5 from four memory locations at $800, $801, $802, and $803.

• Assembly and Execution Example (Chapter 2)

More ASM Examples

See “Example ASM Code.txt” for more examples
Branch Instructions

• Function of branch instructions
  – implement loops; IF-THEN-ELSE constructs
  – alter execution order of linear instructions

• 68HC12 has 2 types of branches
  – conditional
  – unconditional
  – both use Relative Addr. Mode

• Unconditional Branches
  – does not depend on any conditions; will always occur
  - BRA ; branch always
    • 8-bit signed offset
    • distance to next instr. +127 to -128
  – EXAMPLE:
    – best to use labels & let assembler determine relative offset

Branch Instructions II

• More unconditional branches
  - LBRA ; long branch always, 16-bit signed offset
  - JMP ; jump, 16-bit extended/indexed addr.
    • not relative like BRA; specific 16-bit addr.
    • less compact & more clocks than BRA

• Conditional Branch
  – branch only if a condition is true
    • conditions derived from CCR flags
    • previous instr. must set proper CCR flags
      – can use a Test/Compare instruction to set flags without changing memory values
  - Bxx ; conditional branches with 8-bit signed offset
  - LBxx ; conditional long branches with 16-bit signed offset
  – all conditional branches use Relative Addr. Mode
  – best to use labels & let assembler determine relative offset operand
AMS Branch Instructions

**Unconditional Branches**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRN</td>
<td>branch never</td>
<td>useful as delay</td>
</tr>
<tr>
<td>BRA</td>
<td>branch always</td>
<td>8-bit signed offset</td>
</tr>
</tbody>
</table>

**Simple Conditional Branches**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>&quot;if&quot; Function</th>
<th>Condition = Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCC</td>
<td>carry clear</td>
<td>C=0</td>
</tr>
<tr>
<td>BCS</td>
<td>carry set</td>
<td>C=1</td>
</tr>
<tr>
<td>BEQ</td>
<td>equal</td>
<td>Z=1</td>
</tr>
<tr>
<td>BNE</td>
<td>not equal</td>
<td>Z=0</td>
</tr>
<tr>
<td>BMI</td>
<td>minus</td>
<td>N=1</td>
</tr>
<tr>
<td>BPI</td>
<td>plus</td>
<td>N=0</td>
</tr>
<tr>
<td>BVS</td>
<td>overflow set</td>
<td>V=1</td>
</tr>
<tr>
<td>BVC</td>
<td>overflow clear</td>
<td>V=0</td>
</tr>
</tbody>
</table>

**Unsigned Conditional Branches**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>&quot;if&quot; Function</th>
<th>Condition</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHS</td>
<td>higher or same</td>
<td>r ≥ 0</td>
<td>C=0</td>
</tr>
<tr>
<td>BHI</td>
<td>higher than</td>
<td>r &gt; 0</td>
<td>C+Z=0</td>
</tr>
<tr>
<td>BLS</td>
<td>lower or same</td>
<td>r ≤ 0</td>
<td>C+Z=1</td>
</tr>
<tr>
<td>BLO</td>
<td>lower than</td>
<td>r &lt; 0</td>
<td>C=1</td>
</tr>
</tbody>
</table>

**Signed Conditional Branches**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>&quot;if&quot; Function</th>
<th>Condition</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGE</td>
<td>greater or equal</td>
<td>r ≥ 0</td>
<td>N⊕V=0</td>
</tr>
<tr>
<td>BGT</td>
<td>greater than</td>
<td>r &gt; 0</td>
<td>Z+N⊕V=0</td>
</tr>
<tr>
<td>BLE</td>
<td>less or equal</td>
<td>r ≤ 0</td>
<td>Z+N⊕V=1</td>
</tr>
<tr>
<td>BLT</td>
<td>less than</td>
<td>r &lt; 0</td>
<td>N⊕V=1</td>
</tr>
</tbody>
</table>

Example:
If all flags = 0 after an instruction, which branches would be taken?

---

**“Relative” Concept**

- How can we define the address of the red house?

- Absolute Address
  - 3rd House on the left from Main Street

- Relative Address
  - locate by offset from reference
  - references offsets offset + reference
    - Green House
    - Pink House
    - ‘X’

---
Relative Address Mode

• Function
  – adjust PC by value in operand
  – operand can be set by program or by assembler
    • assembler can calculate “distance” to a Label on line to branch to
  – only used in Branch instr.

• EXAMPLES

Counting Relative Offsets

• Offset in Relative Addr. Mode specifies how many instruction bytes (forward or backward) the PC must be moved to get from current PC value to new PC value
  – PC(next) = PC(current) + Offset ;offset is signed (+/-)

• Number of Bytes per Instruction
  – sum of instr. op-code and operand bytes
  – easy to see in .LST file

• EXAMPLE: What is the hex value of ??

<table>
<thead>
<tr>
<th>Program</th>
<th>op-codes</th>
<th>#bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDDA</td>
<td>#10</td>
<td>2</td>
</tr>
<tr>
<td>STUK</td>
<td>INCA</td>
<td>1</td>
</tr>
<tr>
<td>STAA</td>
<td>$10B5</td>
<td>3</td>
</tr>
<tr>
<td>TSTA</td>
<td>97</td>
<td>1</td>
</tr>
<tr>
<td>BNE</td>
<td>STUK</td>
<td>2</td>
</tr>
</tbody>
</table>

??
**BRCLR / BRSET**

- **Function**
  - combine test and branch instr.
  - can use direct, extended, or indexed addr. mode

- **Format**
  - BRCLR/SET mem mask label
  - mem = addr. of data to test
  - mask = bits to check (set to 1 if checked)
  - label = branch location
  - BRCLR: are 1 bits in mask = 0 @ mem?
    - if yes, branch to label
  - BRSET: are 1 bits in mask = 1 @ mem?
    - if yes, branch to label

- **EXAMPLE** BRCLR 0,X %11110000, NEXT
  - compare value @ <IX> with $F0
  - if bits 7-4 are ‘0’, then branch to NEXT
  - else, advance to next instr.

---

**Instruction Timing**

- All ASM instructions require a specific amount of time to execute
  - can use this information for program timing, e.g., delay loops

- \( T_{instr} = \text{Instruction execution time} \)

  \[ T_{instr} = N \cdot t_{cycle} \]

  - \( N \) = # clock cycles per instruction
    - listed under ~ in textbook Appendix A (\text{HO}_4)
    - \( N \) depends on address mode
      - **EXAMPLE:**

- \( t_{cycle} \) = time of one clock cycle = \( 1/f_{clk} \), where \( f_{clk} \) = clock frequency
  - **EXAMPLE:**
Creating Time Delays

- How can a program generate a specific time delay?
  1. Use a hardware timer – will be discussed later
  2. Construct a software delay loop

- Software delay loop
  - EXAMPLE:

  - What is the loop delay?

Creating Time Delays II

- EXAMPLE continued:
  - What is the loop delay?

  - What is ‘count’ for $T_D = 2\text{ms}$?

  - What is the maximum delay for this loop code @ $f_{\text{clk}} = 10\text{MHz}$?
Creating Time Delays III

- How can we make a longer loop for longer delay?
  1. More instructions (clocks) within the loop
  2. Use nested loops

- Nested Loop: Loop within a loop
  - EXAMPLE

- Take home exercise:
  - How long is a nested loop delay if CNTX & CNTY = $FFFF?

ASM Loop Examples

Discuss ASM Loop Examples in HO_5, pg. 4-5

- Branches & Reading Assembled List File (Chapter 2), pg. 4
  1. Where is the program stored in memory (what addresses)?
  2. What is the op-code for the ASM instruction INCB?
  3. What value is loaded into index in line 9?
  4. What address mode is used to store data to memory in line 12?
  5. What is the value of the relative_address_mode offset byte for BEQ in line 15? Forward or backward?
  6. What is the value of the relative_address_mode offset byte for BRA in line 18? Forward or backward?
  7. What does the program do? Where is the main loop (from what line to what line)?
  8. What is the purpose of line 14?
  9. How many times does copy loop execute? Does the value $16 get copied?
  10. Could you explain the purpose and operation of each line in this code?

- Example Loop using For looping structure (pg. 5)

- Also, Simple Looping Code (slide 6)