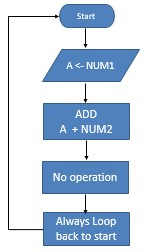
**Objective:**

This lab introduces simple computations to demonstrate how a program is written to perform math operations and alter program flow based on results of math operations. Students will learn to use assembly branch instructions to implement change in program flow. By writing code to read the CCR bits after a math operation, students will learn how to implement if-then conditional program flow execution. You will simulate, not need the training boards. **Complete Pre-Lab exercise before attempting this lab.**

**Instructions:**

**Part 1**:

**Practice reading the CCR**

1. Using the CodeWarrior IDE in Full Chip Simulation mode:
   * Create a project, name your project LastName\_Lab04
   * Write the program presented in the Lab 04 Pre-Lab Work question 1. After the nop instruction, write the instruction to branch always to the start label.
   * Program flow chart logic is shown to the right.
   * Once your program code is entered, Build your project

(Project | Make or F7) and then

* + Demonstrate and verbally explain:

o Your program to the Instructor or TA and o Show your completed (hand calculations and all) PreLab 04 work.

Check Point Ask Instructor to verify your work.

1. While still in the debugger, Reset your program (HCS12XE FCS | Reset) and then single**step through** the program up to the branch always instruction. As you single-step, observe the A accumulator register and the CCR in the Register window and compare your program’s results with your answers for **Pre-Lab 04 questions 1.a-f**.

* + 1. Get a screen shot of your current code (just use the [Snipping tool)](https://support.microsoft.com/en-us/help/4027213/windows-10-open-snipping-tool-and-take-a-screenshot). Be sure the screen shot that includes entire debugger window with the Register Window and your program path directory displayed in the top of the Source Window.

* + 1. Question: Are your Pre-Lab 04 predicted/calculated answers for 1.a-f the same or different than the values shown in the debugger windows? Explain.

You will include both a and b in your lab report.

Note:

The condition code register can be viewed in the Register window. The format is:

# ccr: S X H I N Z V C

The dark black NZVC above indicates the Negative (N), Zero (Z), Overflow (V), and Carry (C ) flags are each set (1 or true). If a flag is clear, (0 or false) the letter is light gray.

1. While still in the debugger, change NUM\_1 to $F6 and NUM\_2 to $EC. (Do you remember how to do this without closing the debugger? Lab 03 tells you how)

Then, single-**step through** the program up to the branch always instruction. As you single-step, observe the A register and CCR in the Register window and compare your program’s results with your Pre-Lab 04 answers for questions 2.a-f.

* 1. Get a screen shot of your current code (just use the [Snipping tool)](https://support.microsoft.com/en-us/help/4027213/windows-10-open-snipping-tool-and-take-a-screenshot). Be sure the screen shot includes entire debugger window with the Register Window and your program path directory displayed in the top of the Source Window.

* 1. Question: Are your predicted/calculated answers for 2.a-f, Pre-Lab 04 the same or different than the values shown in the debugger windows? Explain.

You will include both a and b in your lab report.

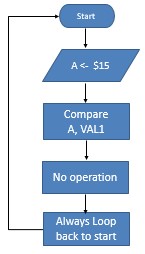
1. Close the debugger and return to your IDE code editor. Change both NUM\_1 and NUM\_2 to $3F and replace the ADDA instruction with SUBA instruction Build (Project | Make or F5) your program, step through the program and compare your program’s result with your PreLab 04 answers for questions 5.a-f.

* 1. Get a screen shot of your current code (just use the [Snipping tool)](https://support.microsoft.com/en-us/help/4027213/windows-10-open-snipping-tool-and-take-a-screenshot). Be sure the screen shot includes entire debugger window with the Register Window and your program path directory displayed in the top of the Source Window.

* 1. Question: Are your predicted/calculated answers for 2.a-f Pre-Lab 04 the same or different than the values shown in the debugger windows? Explain.

You will include both a and b in your lab report.

**Part II:**

1. **Write your own program**
   1. In CodeWarrior IDE editor, write the instructions that defines a variable VAL1 as a byte of storage initialized with 60 hex, loads Accumulator A with 15 hex and then use the CMPA instruction to compare the value in Accumulator A to VAL1. Write a nop instruction after the CMPA instruction and your last instruction should branch always to the start of your program.

Note: The CMPA instruction compares the two values by "subtracting" VAL1 from the value in Accumulator A, but does not change either value. (reference Reading material 4.10.3, and Lecture 4). Program flow-chart logic is shown to the right.

* 1. In CodeWarrior IDE debugger, Single-step to the branch always instruction, record observed register values in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Observed Values when the program reaches the branch always instruction | | | | |
| Accumulator A | N | Z | V | C |
| 15 | 1 | 0 | 0 | 1 |

* 1. Get a screen shot of your program (just use the [Snipping tool)](https://support.microsoft.com/en-us/help/4027213/windows-10-open-snipping-tool-and-take-a-screenshot).

1. **Analyze a Program**

* 1. Consider the following assembly-language program: do not write this code yet!

; in the constants section

Num1: dc.b $C0

Num2: dc.b $05

Cnt: dc.b $3F

; in the code section

Again: LDAB Num1 LDAA Num2

ADDB Cnt

INCB

SUBA #$04

DECA

BRA Again

1. Using your list of opcodes: Fill in the table with the machine language for the program. Make sure you include the constants at the end. Values are in hex, I did the first instruction *ldab Num1* for you.

Note: Program Code and Constants are stored in Rom Memory starting at address C000 hex

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C000 | F6 | C0 | 0F | B6 | C0 | 10 | FB | C0 |
| C008 | 11 | 52 | 80 | 04 | 43 | 20 | F1 | C0 |
| C010 | 05 | 3F |  |  |  |  |  |  |

1. What is the machine code for the instruction INCB? Answer: 52

1. In the table below, without running the program, predict the hex contents of Accumulator A and Accumulator B **after** each instruction is executed. Assume that initially both of these registers hold zero. Also predict the values of the N, Z, V and C bits in the Condition Code Register **after** each instruction is executed, again assuming that initially they’re all zero. I have filled in the first row for you.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction | Predicted Values After Instruction Executes | | | | | |
| Accumulator A | Accumulator B | N | Z | V | C |
| LDAB Num1 | CB or 0 | $C0 | 1 | 0 | 0 | 0 |
| LDAA Num2 | 5 | C0 | 0 | 0 | 0 | 0 |
| ADDB Cnt | 5 | FF | 1 | 0 | 0 | 0 |
| INCB | 5 | 0 | 0 | 1 | 0 | 0 |
| SUBA #$04 | 1 | 0 | 0 | 0 | 0 | 0 |
| DECA | 0 | 0 | 0 | 1 | 0 | 0 |
| BRA Again | 0 | 0 | 0 | 1 | 0 | 0 |

1. In CodeWarrior IDE editor, enter the program, make sure to use the proper sections. Make your program. Then single-step through the program, recording observed register values in the table below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction | Predicted Values After Instruction Executes | | | | | |
| Accumulator A | Accumulator B | N | Z | V | C |
| LDAB Num1 | CB | C0 | 1 | 0 | 0 | 0 |
| LDAA Num2 | 5 | C0 | 0 | 0 | 0 | 0 |
| ADDB Cnt | 5 | FF | 1 | 0 | 0 | 0 |
| INCB | 5 | 0 | 0 | 1 | 0 | 0 |
| SUBA #$04 | 1 | 0 | 0 | 0 | 0 | 0 |
| DECA | 0 | 0 | 0 | 1 | 0 | 0 |
| BRA Again | 0 | 0 | 0 | 1 | 0 | 0 |

1. Do the observed values agree with your predictions? (Yes or No?) \_\_\_\_YES\_\_\_\_\_\_ If not, make sure you understand why not.

1. Get a screen shot of your program in the debugger (just use the [Snipping tool)](https://support.microsoft.com/en-us/help/4027213/windows-10-open-snipping-tool-and-take-a-screenshot).

**3. Two More Programs of your Own**

1. Based on your understanding of the bits in the condition code register, and using only the instructions discussed so far, use the table below to plan a program that results in the Z bit and the C bit both being set to 1 when the program ends. In the table below, write the assembly language for your program (one instruction per line, add more lines if needed):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Program  Instruction | Predicted Values After Instruction Executes | | | | | |
| Accumulator A | Accumulator B | N | Z | V | C |
| Ldaa #$FF | FF | CB | 1 | 0 | 0 | 0 |
| Adda #$01 | 00 | CB | 0 | 1 | 0 | 1 |
| NOP | 0 | CB | 0 | 1 | 0 | 1 |
|  |  |  |  |  |  |  |

1. In the table below, write the machine language for your program.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C000 | 86 | FF | 8B | 01 | A7 |  |  |  |
| C008 |  |  |  |  |  |  |  |  |
| C010 |  |  |  |  |  |  |  |  |

1. In CodeWarrior IDE editor, clear your editor of previous code and declarations. Then enter your program and single-step through it. When you are sure that the program works correctly, ask Instruction or TA to check it.

Check Point

1. Next, using only the instructions discussed so far, Use the table below to plan a new Assembly language program that results in the N bit and the C bit both being set to 1 when the program ends. (one instruction per line, add more lines if needed):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction | Predicted Values After Instruction Executes | | | | | |
| Accumulator A | Accumulator B | N | Z | V | C |
| LDAA #$C1 | C1 | CB | 1 | 0 | 0 | 0 |
| ADDA #$C0 | 81 | CB | 1 | 0 | 0 | 1 |
| NOP | 81 | CB | 1 | 0 | 0 | 1 |
|  |  |  |  |  |  |  |

1. In the table below, write the assembly language for your program.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C000 | 86 | C1 | 8B | C0 | A7 |  |  |  |
| C008 |  |  |  |  |  |  |  |  |
| C010 |  |  |  |  |  |  |  |  |

1. In CodeWarrior IDE editor, comment out your previous code and declarations and enter your program. Make (F7) your program and single-step through it. When you are sure that the program works correctly, continue to next step.
2. Why is it impossible to write a program that results in the N bit, the Z bit, and the C bit all being set to 1 when the program ends? Write your answer below.

Answer: You can’t have a negative number be zero. There is no possible way for you to have a negative number and for it to be zero.

**Part III**

**1. Analyzing Another Program**

1. Consider the following assembly-language program:

ENTRY:

LDAA #13

INCA

LDAB #%00100011

ABA

SUBA #$10

BRA ENTRY

In the table below, write the machine language for the program.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C000 | 86 | 0D | 42 | C6 | 23 | 18 | 06 | 80 |
| C008 | 10 | 20 | F5 |  |  |  |  |  |
| C010 |  |  |  |  |  |  |  |  |

1. In the table below, without running the program, predict the hex contents of the registers after each instruction is executed. Assume that initially these registers hold zero, except the program counter, which holds $C000. The first row is done for you. Enter values in Hex.

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Predicted Values After Instruction Executes | | |
| Program Counter | Accumulator A | Accumulator B |
| LDAA #13 | C002 | 0D | 0 |
| INCA | C003 | E | 0 |
| LDAB #%00100011 | C005 | E | 23 |
| ABA | C007 | 31 | 23 |
| SUBA #$10 | C009 | 21 | 23 |
| BRA ENTRY | C000 | 21 | 23 |

1. In CodeWarrior IDE editor, comment out the previous code and declarations and enter the program. Then single-step, recording your observed values below.

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Predicted Values After Instruction Executes | | |
| Program Counter | Accumulator A | Accumulator B |
| LDAA #13 | C002 | 0D | 0 |
| INCA | C003 | E | CB |
| LDAB #%00100011 | C005 | E | 23 |
| ABA | C007 | 32 | 23 |
| SUBA #$10 | C009 | 32 | 23 |
| BRA ENTRY | C000 | 21 | 23 |

1. Do the observed values agree with your predictions? (Yes or no?) \_\_\_\_\_\_YES\_\_\_\_\_\_ If not, make sure you understand why not

**Report:**

Your report should be well written and include properly formatted and clearly marked questions and answers. Make sure to include the question with your answer, include all tables and work. Include the required screen shots clearly marked with responses. Follow the order in which the lab questions are presented and mark each section.

For your discussion and observation remarks to questions in **Part I and Part II**: include your **Hand calculation verification from the Pre-Lab 04, and screen shots of your Lab 04 program code Part I and Part II:** Show the hex math in binary as well (indicate all carry/borrows). Compare your program code and recorded results with your manual calculations and answers. Then compare program execution results with your hand calculated results. Did you calculate the same as your program results? Discuss if your hand calculations support the results determined in your recorded program execution.

**For the program code in Part III:** discussion and observation remarks to questions should include, whether your hand calculations support the results determined in your recorded program execution.

**Write your Report:**

* Word document no more than a few pages, well formatted with: Your name, lab section #, lab assignment #, and date, (save your .docx as YourLastName\_Lab##). (wherer ## is the lab number)
* Include sections:
  + Objective (in your own words), o Procedure (one short concise paragraph),
  + Answers to questions (clearly identify the question and your answer. *Ensure your answers are indicated in highlight, red, or some means to indicate the answer from the instructions*),
  + Observations: Hand calculation verifications and a Conclusion (discuss points of learning).
  + Conclusion: Summarize what you have learned and observed. Include discussion on the CCR, Write a brief summary of your observations for Parts I, II, and III.

**Report:**

Your report should be well written and include properly formatted and clearly marked questions and answers. Make sure to include the question with your answer, include all tables and work. Include the required screen shots clearly marked with responses. Follow the order in which the lab questions are presented and mark each section.

For your discussion and observation remarks to questions in **Part I and Part II**: include your **Hand calculation verification from the Pre-Lab 04, and screen shots of your Lab 04 program code Part I and Part II:** Show the hex math in binary as well (indicate all carry/borrows). Compare your program code and recorded results with your manual calculations and answers. Then compare program execution results with your hand calculated results. Did you calculate the same as your program results? Discuss if your hand calculations support the results determined in your recorded program execution.

**For the program code in Part III:** discussion and observation remarks to questions should include, whether your hand calculations support the results determined in your recorded program execution.

**Submit by due date:** Your completed report and your completed program, just the main.asm file for Part III. Be sure your code is well organized, includes a commented header, and logical comments and Part III clearly commented.