Appendix D - Examples Chap 1 – Programming CARDIAC -

D.1 - LAB/HOME ASSIGNMENT #0:  Gauss - Sum the numbers  (+3 points)

Johann Karl Friedrich Gauss (1777 - 1855) had an immense influence in a wide variety of fields in both mathematics and physics including number theory, analysis, differential geometry, geodesy, magnetism, astronomy and optics.

As the story goes, in 1784, when Gauss was seven years old, his arithmetic teacher asked the pupils to add up the numbers from 1 through 100. No sooner had the teacher finished stating the exercise when young Gauss placed his slate on the teacher's desk with the answer to $1 + 2 + 3 + 4 + 5 + \ldots + 99 + 100$.

Alas, the answer is too big to easily compute using CARDIAC (requires double precision arithmetic… which CARDIAC can do… if you know how).

PADITM – program specification; analysis; design; implementation; testing; and maintenance.

Program specification:
Write an infinite looping CARDIAC program to get a number from the user. Compute and display the sum ($1+2+3+\ldots+n$).
Use CARDIAC machine language (just like step 12 in the table “D.7.4 -CARDIAC’s Programming Ideas” that starts on D-216).
Simplifying assumptions: $1 \leq n \leq 44$.

Analysis:
Darn! Simplifying assumptions make it too simple.

Design:  Code this algorithm. (Picture diagram and word description are the same algorithm!)

\[
\text{top:} \\
\quad \text{initialize sum to 0} \\
\quad \text{get target number from user} \\
\text{loop:} \\
\quad \text{add target to sum} \\
\quad \text{target} \ - \ = \ 1 \\
\quad \text{more to go?} \ \rightarrow \ (y) \ \text{goto loop} \\
\quad \quad (n) \\
\quad \text{display sum} \\
\quad \text{goto top}
\]

Implementation:
Translate into CARDIAC machine language, compile and load in CPU, then run it.

Testing:  Compute the answers by hand for 0, 1, 5, and 10. Now, test your program.

To earn +3 points credit:  Send an email to gmontante.dvc.edu@gmail.com. Use the subject line “CS260 - Lab #0 Gauss - <your name as it appears on the roster>”. Paste your program solution at the beginning of the email. Manually type the answers for 0, 1, 5, 10, and 44 at the end. Due at the beginning of class on day #3.

You must do the work yourself! Do NOT copy from anyone! NO CREDIT for late assignments.

So what the heck is CARDIAC?  Glad you asked.  Read on …
**D.2 - Introduction to CARDIAC**

CARDIAC is a miniature computer based on the Bell Labs’ "CARDboard Illustrative Aid to Computation" written in 1967 to show how a computer works.

Although made of cardboard, it had the heart of a “doped” silicon machine including -

- RAM (±999)
- ROM
- CPU
- data
- accumulator register (±9999)
- arithmetic
- logic
- I/O
- single-cycle CPU
- machine language
- instruction register
- instruction pointer register
- assembler (compiler)
- assembly language

(A bigger version of the picture is available from CARDIAC’s main menu Help → Pictures of CARDIAC…) Note: Since CARDIAC originally existed only in our heads, it was a base ten computer, just like our brains and our two hands. It doesn’t use transistors and doesn’t worry about base two.

**D.3 - Caution: challenges ahead…don’t judge a book by its cover!**

“Doesn’t look like much”, you say?

Well, *machine language is all about using the provided resources to solve the required problem.*

The above computer can act as a calculator, do recursive subroutines, sort, play games, etc. – limited only by your skill and imagination…

**D.4 - Download the CARDIAC program from my website**

Click [http://voyager2.dvc.edu/~gmontante/cs265/exe/CardiacInterpreter.exe](http://voyager2.dvc.edu/~gmontante/cs265/exe/CardiacInterpreter.exe) to get the electronic version of CARDIAC.
D.5 - CARDIAC’s hardware

After you download CARDIAC, click the heart to view the CPU and look for the ideas mentioned above.

D.5.1 - CARDIAC’s Memory
CARDIAC has 1 ROM cell (address 00), 98 general purpose RAM cells (addresses 01-98), and 1 special purpose cell (address 99 is half ROM and half RAM).

Cells contain 3 decimal digits and a plus or minus sign.
Each cell can hold numeric data or CARDIAC machine language numeric codes.

D.5.2 - CARDIAC’s accumulator register
Built into the CARDIAC CPU is special hardware, called a register, that can hold numbers and is used to perform special actions.

The accumulator register can hold a number with 4 decimal digits and a sign.
There are machine language instructions to move data into and out of the accumulator, and to perform arithmetic and logical operations.

Note: numbers in the accumulator may be too large to fit into RAM (e.g., 1234). The highest digit is lost (e.g., 1234 stores in a RAM cell as 234).

D.5.3 - CARDIAC’s instruction pointer register
The instruction pointer register contains the address of the data cell which contains the NEXT MACHINE LANGUAGE INSTRUCTION CODE for the CPU to perform.

After you put CARDIAC machine language codes and data into cells, you put the address of the cell which contains the first instruction of your program into CARDIAC’s instruction pointer.

D.5.4 - Other hardware
When we are ready to single-step CARDIAC through a program, we’ll learn about a single-cycle CPU and the instruction register.
D.6 - CARDIAC’s machine language

Each action that CARDIAC can perform has a numeric code. The set of these codes is CARDIAC’s machine language.

Each numeric code is two parts:
- the leading hundred’s digit is the opcode (operation code) and
- the ten’s and unit’s digits are the two digit operand, whose meaning depends on the opcode.

For example, the number $6 \ 2 \ 4$ has the opcode 6 and operand 24. It means "Store the contents of the accumulator into the memory cell whose address is 24".

From the diagram, you see that CARDIAC can

(1xx and 6xx) move data between memory cells and the accumulator;
(0xx and 5xx) perform input/output to/from memory cells;
(2xx and 7xx) add/subtract memory cells from the accumulator;
(8xx, 3xx, and 9xx) jump to a non-sequential part of a program, or jump conditionally, or halt execution;
(4xy) perform a left and right digit shift on the contents of the accumulator
    (that is, opcode 4xy does multiply by $10^x$, then divide by $10^y$).
D.7 - Software ideas

D.7.1 - Programming in General

Plan what you are going to do BEFORE you try to convince the CPU to do something!
What is PADITM?? (see page D-211)

D.7.2 - Load a program

Originally, programs were loading by toggling switches on a computer’s front panel or shuffling wires. Here, we just use CARDIAC’s monitor program.

(1) Load a machine language program directly into RAM.

(A) Click the Heart to start the CPU.
(B) Right click a memory cell to bring up the “Edit Memory Cell” dialog.
(C) Enter the machine language code which describes what you want to do.

(2) Alternately, compile and load a program into RAM.

(A) Put assembler codes into the text window.
(B) Click “Compile and Start” button (or press F5)

Remember - you are in complete control of what numbers you put into which cells, and what those numbers stand for.

D.7.3 - Run a program

To run a program, put the address of the first instruction into the “Instruction pointer” register, then click “Single Step” button to trace through your program step by step, or click “RUN” button to go full-speed ahead.

CARDIAC is a single-cycle CPU, which means it repeats the same sequence of steps for each instruction in the program. Use the “Single Step” button to see this sequence.
### D.7.4 - CARDIAC's Programming Ideas

Practice the following ideas, which are useful for writing your first program, by loading codes directly to RAM (see "Load a program" on page D-215, loading directly into RAM).

<table>
<thead>
<tr>
<th>IDEA</th>
<th>C++ translation</th>
<th>CARDIAC translation</th>
<th>Codes</th>
</tr>
</thead>
</table>
| 1 | You “create” a variable by mentally reserving some cell number for that purpose... | int x, y; (C++ compiler picks the locations for you...) | Pick cells to hold values. E.g.,  
- cell 12 will hold x.  
- cell 13 will hold y. |
| 2 | Put a constant into a cell... | x = 4; | Right-click cell 12, then enter the value 4. |
| 3 | Display the contents of a cell... | cout << x; | Pick a cell to hold this instruction (e.g. cell 20). Right-click 20, enter code ____ (display value in cell 12) |
| 4 | How do you run the CPU from a given starting point? | Initialize the i________ p________, click Run or Single Step. What address do you use to run “cout << x;” ??? |
| 5 | Oops, how do you exit or stop a program? | return; | ____ halt and reset IP to address of the cell containing the “cout << x;” |
| 6 | How do you move data to/from the accumulator? | What the heck’s an accumulator? | ____ copy x to accumulator (acc ← x)  
____ copy accumulator to y (y ← acc) |
| 7 | How do you move data around? | x = y; | hint: similar to step 6... |
| 8 | Put user data into a cell at run time... | cin >> x; | ____ input to x |
| 9 | Do some arithmetic.  
Is x*100/10 == x*10? |  
{x + y; x-y; x*10; x/100; x*100/10; x % 10} | ____ copy x to accumulator (acc ← x)  
____ add y to accumulator (acc += y)  
____ multiply acc by 10 (acc *= 10) |
| 10 | Translate  
“put the result of x-10*y into z”.  
(note: multiplication BEFORE subtraction, so must compute 10*y first!) | int z =x-10*y; | E.g., use cell 14 to hold the value of Z...  
____ copy y to accumulator (acc ← y)  
____ multiply acc by 10 (acc *= 10)  
____ save result in progress (z ← acc)  
____ copy x to acc (acc ← x)  
____ sub z from acc (acc -= z)  
____ copy acc to z (z ← acc) |
11 Translate the C++ program into Cardiac machine language.

```cpp
void main()
{
  int a = 20;
  int b;
  cin >> b;
  int sum = a + b;
  cout << sum;
  int boo = sum / 10;
  cout << boo;
  return;
}
```

Let data begin at cell 8:
- Cell 8: value for A; right click 8, enter 20
- Cell 9: value for B
- Cell 10: value for sum
- Cell 11: value for boo

Let program start at cell 15:
- Input \( \rightarrow \) B
- Acc \( \leftarrow \) A
- Acc += B
- Sum \( \leftarrow \) acc
- Display sum
- Acc \( \leftarrow \) acc / 10
- Boo \( \leftarrow \) acc
- Display boo
- Halt, reset to beginning

12 To simplify changing a program, create a machine language version in the editor, compile, and load it.

Copy and paste to Cardiac text editor window

```cardiac
;Module: Demo.ci
;Author: (GSM)Montante, Gary
;Purpose: Machine language coding in edit window
;History: 01/19/2008 (GSM)- in course notes
00: cel 001 ;ROM cell
08: cel 20 ;variable A
09: cel ? ;variable B
10: cel ? ;value for sum
11: cel ? ;value for boo
15: cel 009 ;input \( \rightarrow \) B
16: cel 108 ;acc \( \leftarrow \) A
17: cel 209 ;acc += B
18: cel 610 ;sum \( \leftarrow \) acc
19: cel 510 ;display sum
20: cel 401 ;acc \( \leftarrow \) acc / 10
21: cel 611 ;boo \( \leftarrow \) acc
22: cel 511 ;display boo
23: cel 915 ;halt, reset to start
;specify start point
```

13 How can you put zero into the accumulator without having to manually initialize a cell to 0?

What the heck is an accumulator?

```
"acc = 0;"
```

Three ways!

```
_____ acc \( \leftarrow \) any cell
_____ acc -= that cell
or
_____ acc /= 10000
or
_____ acc *= 10000
```

14 How do you jump to a non-sequential part of your program (that is, how do you loop)?

```
label:
x=x + b;
goto label;
```

Suppose you want to jump to the instruction at address 20:

```
_____ Change IP to cell 20
```
<table>
<thead>
<tr>
<th></th>
<th>IDEA</th>
<th>C++ translation</th>
<th>CARDIAC translation</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>How do you write an infinite loop to count 3, 2, 1, 0, -1, -2, etc.?</td>
<td><code>int cnt = 3;</code>&lt;br&gt;<code>repeat:</code>&lt;br&gt;<code>    cnt = cnt - 1;</code>&lt;br&gt;<code>    goto repeat;</code>&lt;br&gt;<code>or</code>&lt;br&gt;<code>while(true)</code>&lt;br&gt;<code>    cnt -=1;</code></td>
<td><code>org 14</code>&lt;br&gt;<code>cel 3 ;cnt variable (right click &amp; enter 3)</code>&lt;br&gt;<code>org 20</code>&lt;br&gt;<code>    ______ acc ← cnt ; top of loop</code>&lt;br&gt;<code>    ______ acc -= cell 0 ;(always has 1!!!)</code>&lt;br&gt;<code>    ______ cnt ← acc</code>&lt;br&gt;<code>    ______ jmp to top of loop</code></td>
<td>114, 706, 614, 820</td>
</tr>
<tr>
<td>16</td>
<td>How do you test if the value of a variable is less than 0?</td>
<td><code>if( x &lt; 0 )</code>&lt;br&gt;<code>goto addr 20;</code></td>
<td><code>org 14</code>&lt;br&gt;<code>cel ? ;variable X</code>&lt;br&gt;<code>    ______ acc ← x</code>&lt;br&gt;<code>    ______ jump to cell 20 if acc &lt; 0</code></td>
<td>114, 230</td>
</tr>
<tr>
<td>17</td>
<td>How do you test if the value of a variable is greater than or equal to zero 0?</td>
<td><code>if( x &gt;= 0)</code>&lt;br&gt;<code>goto addr 20;</code>&lt;br&gt;<code>next:</code></td>
<td><code>org 8</code>&lt;br&gt;<code>cel ? ;variable X</code>&lt;br&gt;<code>org 10</code>&lt;br&gt;<code>    10: ______ acc ← x</code>&lt;br&gt;<code>    11: ______ jump to cell 13 (acc &lt; 0)</code>&lt;br&gt;<code>    12: ______ jump to cell 20 (acc &gt;= 0)</code>&lt;br&gt;<code>    13:     ---- next ---</code></td>
<td>108, 313, 820</td>
</tr>
<tr>
<td>18</td>
<td>How do you write a loop to display 3,2,1,0 then stop?</td>
<td><code>int a = 3;</code>&lt;br&gt;<code>do</code>&lt;br&gt;<code>    {</code>&lt;br&gt;<code>    cout &lt;&lt; a;</code>&lt;br&gt;<code>    a--;</code>&lt;br&gt;<code>}while(a &gt;= 0);</code>&lt;br&gt;<code>cout &lt;&lt; a;</code></td>
<td><code>org 8</code>&lt;br&gt;<code>cel ? ;loop count A</code>&lt;br&gt;<code>org 10</code>&lt;br&gt;<code>    ______ display value in A</code>&lt;br&gt;<code>    ______ acc ← A</code>&lt;br&gt;<code>    ______ acc -= 1 (ROM cell 0 == 1!)</code>&lt;br&gt;<code>    ______ A ← acc</code>&lt;br&gt;<code>    ______ stop loop if acc &lt; 0</code>&lt;br&gt;<code>    ______ continue looping</code>&lt;br&gt;<code>    ______ display value in A</code></td>
<td>508, 108, 706, 616, 810, 508</td>
</tr>
<tr>
<td>19</td>
<td>How do you compare two numbers in general?</td>
<td></td>
<td><code>org 8</code>&lt;br&gt;<code>cel ? ;variable X</code>&lt;br&gt;<code>cel ? ;variable Y</code>&lt;br&gt;<code>org 10</code>&lt;br&gt;<code>    10: 108 ; acc ← X</code>&lt;br&gt;<code>    11: 709 ; acc = x-y</code>&lt;br&gt;<code>    ______ ;jump acc &lt; 0 (x-y &lt; 0)</code>&lt;br&gt;<code>    ; So x-y &gt;= 0 ....</code>&lt;br&gt;<code>    700 ; acc -1 = acc</code>&lt;br&gt;<code>    ______ ;jump acc &lt; 0 (x-y == 0)</code>&lt;br&gt;<code>    ______ ;jump (x-y &gt; 0)</code></td>
<td></td>
</tr>
</tbody>
</table>
Are you tired of writing a bunch of numbers to program in machine language? Well, use *assembly language* and an assembler (compiler) instead...the assembler translates op codes and operands into numbers for you and keeps track of actual cell locations. See the problem Demo.ci in step 12 above.

The assembler makes a “symbol table” as it reads the code in Demo2.ci (at the right) to note which cells have “label” names:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>08</td>
</tr>
<tr>
<td>B</td>
<td>09</td>
</tr>
<tr>
<td>SUM</td>
<td>10</td>
</tr>
<tr>
<td>BOO</td>
<td>11</td>
</tr>
<tr>
<td>MAIN</td>
<td>15</td>
</tr>
</tbody>
</table>

“org 8” means use cell 8 for the address of the next cell.

“A  cel 20” means use next available RAM cell to store the number 20. “A” is put into the symbol table with the address of this cell (08).

“B  cel ?” means the next cell (09) has an unknown value. “B” is put into the symbol table with the address of this cell (09).

The label “main” is put into the symbol table with the current address (15). Since it isn’t a CARDIAC instruction, the assembler keeps looking for an instruction to put into cell 15.

And so on.

“inp B” means create an instruction:

“inp” means op code 0

“B” means use value of B from symbol table.

Thus, 0 09 is put into cell 15.

---

(a) Copy and paste to CARDIAC text editor window

Same program as Demo.ci above in step 12

```plaintext
;===================================
;Module: Demo2.ci
;Author: (GSM) Montante, Gary
;Purpose: Assembly language coding in edit window
;History: 01/19/2008 (GSM) - in course notes
;===================================

or 00

  cel 001 ;ROM cell

  -------- data

  org 08

  A  cel 20 ;variable A
  B  cel ? ;variable B
  SUM cel ? ;value for sum
  BOO cel ? ;value for boo

  -------- program starting point

  org 15

  main:

  inp B ;input --> B
  lda A ;acc <-- A
  add B ;acc += B
  sto sum ;sum <-- acc
  out sum ;display sum
  salr 01 ;acc <-- acc / 10
  sto boo ;boo <-- acc
  out boo ;display boo
  hrs main ;halt, reset to start

  end main
```

(b) Then click,

Assembler symbol table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>08</td>
</tr>
<tr>
<td>B</td>
<td>09</td>
</tr>
<tr>
<td>SUM</td>
<td>10</td>
</tr>
<tr>
<td>BOO</td>
<td>11</td>
</tr>
<tr>
<td>main</td>
<td>15</td>
</tr>
</tbody>
</table>

Assembler translates this indentically to Demo.ci in step 12.
D.8 - LAB/HOME ASSIGNMENT extra credit: Guess a number (+1 point e.c.)

Write a number guessing game program in CARDIAC assembly language (like step 20 above).

A human thinks of a “target” number from 1 to 99, inclusive. The computer will try to guess the target number by asking the human questions. For example, the computer will ask “Is the number 23?” The human will answer “-1” if 23 is less than the target, “0” if 23 IS the target, or “1” if 23 is greater than the target. The computer keeps asking questions until it guesses the number. Assume that the human always truthfully answers the computer’s questions. Do NOT use brute force (your program always guesses the answer in 7 questions or less)!

Take a moment to design this program… Then compare it with the design given below.

A sample test run – human picks 44.
Click “Compile and Run”, then click "Run"

- Guess = 50. Human enters 1 (too high)
- Guess = 25. Human enters -1 (too low)
- Guess = 37. Human enters -1 (too low)
- Guess = 43. Human enters -1 (too low)
- Guess = 46. Human enters 1 (too high)
- Guess = 44. Human enters 0 (done!)

Program stops (to restart, close CPU, restart);

Test procedure:

What test procedure would you use? Sample test procedure: the highest legal numbers (97,98,99); the lowest legal numbers (1,2,3); a random lower-range even number; and a random upper-range odd number.

Hint 1: Using “add” and "salr 01", which divides by ten, how do you do compute \( \frac{(a+b)}{2} \) ???

Hint 2: How can you change the above “division by 2” into a “division by 10” problem which CARDIAC can do (i.e., a fraction with denominator 10)?

Hint 3: How can you multiply by doing a series of additions instead?

To earn +1 points credit credit: Send an email to gmontante.dvc.edu@gmail.com. Use the subject line “CS260 – Guessing game extra credit - <your name as it appears on the roster>”. Paste your program solution at the beginning of the email. To receive credit, I must be able to copy and paste your submission into CARDIAC’s editor, click “Compile and RUN”, then click RUN. The program runs in an infinite loop so I can immediately start a new game at the end of the previous game. You must do the work yourself! Do NOT copy from anyone!