

CS 216

Laboratory 6:

IBCM Programming¹

Objective:

To become familiar with programming with IBCM, and understand how high-level language programs are represented at the machine level.

Background:

IBCM (Itty Bitty Computing Machine) is a simulated computer with a minimal instruction set. Despite its tiny RISC architecture, the IBCM can compute anything that a current “powerful” computer can.

Preparation required:

- Read slides on IBCM.
- Read IBCM *Principles of Operation* document (distributed in class, on the web, in the CS216 mailbox)
- Read the IBCM *Programming Examples* handout (distributed in class, on the web, in the CS216 mailbox)
- Run (install if necessary) the IBCM simulator. (Already installed on ITC lab machines. An installable version is on the labs webpage.)

¹ Note: Often in the past in CS216, this was Lab 5.

Using the IBCM Simulator

You will be jointly developing an IBCM program in the laboratory. So, as preparation, study the IBCM documentation and lecture notes. Additionally, you will be expected to come to class with two IBCM programs. You will not get much from the lab (nor many points from the lab) unless you are thoroughly familiar with the documentation and lecture notes — so study hard!

There are links from the web page to an IBCM simulator executable file. There are instructions for if you have Visual Basic installed on your machine, for if you do not. You can use the File->Open pull-down to make the simulator read in an IBCM program that you have typed into a text file

Writing IBCM Code

You must comment your IBCM code copiously. This means (practically) every line should have a comment describing what you are doing. The examples provided in the handouts given in class illustrate a good way of doing this, where there are columns for each of the following:

1. the hex values that will go into that memory location;
2. the address of that memory location;
3. labels for jumps or variable names.
4. the operation-name for the instruction on that line;
5. a symbolic name for the address the instruction references
6. comments (that explain what's happening in a higher-level form).

Note that together the operation-name and the address are sort of an assembly language version of the hex-version of the operations in the first column. You probably want to write those first, and then turn these into the hex instruction that will go into column 1.

You may find it useful to write your IBCM code in MS Excel or another spreadsheet program. This will make it easy to create programs as described above and have things line up nicely in columns. To run your program, you can copy the first column (memory contents) into a file and load it into the IBCM simulator. Use the File->Open to load the file having only the first column. Note that if you make a change to this file and the simulator is still running, hitting Reset in the simulator will re-load your program file. (This is very handy!)

Even though you will have your program well-thought out and written out in symbolic IBCM before you start typing it in, alas you may still find that you have forgotten an extra variable or an extra instruction or two that you need to add in. To make these corrections easier, be sure to leave a bit of extra space for variables at the top of your program. You may also want to leave extra nop (B) instructions in your code that could be replaced later with actual instructions if needed. Make use of labels in your symbolic IBCM code to aid readability.

Useful Information

Alt-PrintScrn will copy a screen shot, which can then be pasted into Word or another editor for printing.

Procedure

Pre-lab

0. Download the IBCM simulator from the class website.
 - It should run on any computer with Visual Basic installed.
 - Use it to test/validate your pre-lab programs.
1. “Addition” - Write a *single* IBCM program that does the following:
 - Gets three values from user input,
 - Stores the values into three variables,
 - Adds the variables together, and prints the sum (you may use additional memory if you wish).
 - If the sum is zero, it prints the three values and stops.
 - If the sum is not zero, it starts over (tries to get three values, etc).
2. “Array” - Write another IBCM program that finds the maximum value in an array of values.
 - The array is hard-coded into memory (you may code it after/past the IBCM program, or you may even use your IBCM program itself as the array!).
 - The beginning and end addresses of the array are hard-coded into memory at locations 1 and 2, respectively. You may also hard code other values, such as the number of elements in the array, into your program.
 - Before your program halts, it prints out the maximum value of the array.

In-lab

Come to class with a printed copy of the checkoff sheet, a completely working prelab, one page of paper to write on, and a writing utensil.

0. You will be assigned into groups of two by your TAs.
 - Introduce yourself to your laboratory partner, and exchange email addresses.
1. Both partners should demonstrate their pre-lab code during the first 15 minutes of class.
2. Your group will be given an algorithm to implement in IBCM. Do so:
 - a) Write up high-level pseudo-code for your design on paper (make sure that it is absolutely correct, or you will regret it later).
 - b) Refine this pseudo-code, making it closer to the assembly code level.
 - c) Alter your code into IBCM assembly code with labels instead of addresses.
 - d) Run through a sufficient number of test cases by hand of your IBCM code from step (c) to convince yourself that it is correct.
 - e) Encode into actual hex IBCM code and addresses, and test it using the simulator.
 - f) Identify and fix the errors that you did not pick up in the previous steps.
3. Demonstrate your working programs to a TA for in-lab check-offs.
 - Demonstrate the full functionality of your in-lab using the TA’s test sequence. Show and explain your in-lab implementation process and your algorithm(s).
 - Every member of the group will need a copy of the in-lab code to turn in, so be sure to have everyone save a copy before leaving lab.
4. Turn-in your check-off sheet **before** leaving lab.

Post-lab

0. Implement a “quine” in IBCM *individually* (see “*What is a Quine?*” section).
 - You may discuss this question with your classmates, but apply the “Gilligan’s Island” principle (see lab #3) for writing up and submitting your final answer. (i.e. what you turn in should be your own work.)
1. For this lab you will be submitting your code electronically via the toolkit. Your fully functional code is **due at 11:59 pm Thursday October 27, 2005**. *Your code will be graded for correctness AND style.* Be sure to include: your name, the date, and the name of the file in a comment at the beginning of each file you submit. If you have edited them in excel please save this output to a text file. Be sure that your text file is readable when printed out on a (normal) 8.5” x 11” piece of paper. Please turn in:
 - A text file listing of your COMMENTED pre-lab “addition” code.
 - A text file listing of your COMMENTED pre-lab “array” code.
 - A text file listing of your COMMENTED in-lab code.
 - A text file listing of your COMMENTED post-lab “quine” code.
 - For examples of commented code, see the two examples handed out in class and on the course web page.
 - All IBCM code needs to be commented extensively.

*If you did not get elements of the in-lab test sequence checked off during lab, you may attempt to receive partial credit for these by successfully implementing those elements after lab and submitting your corrected code at turn-in. In order to receive these points you must submit in addition to your code, a paragraph (**text file only** please) describing what difficulties you encountered getting your code working and what you did to solve them.*

What is a Quine?

Based on the experience in lab, you should now be able to write an IBCM program on your own. For the postlab, you should individually write an IBCM program that prints itself. This type of program is known as a “quine.”

quine: /kwi:n/ /n./ [from the name of the logician Willard van Orman Quine, via Douglas Hofstadter] A program that generates a copy of its own source text as its complete output. Devising the shortest possible quine in some given programming language is a common hackish amusement.

While at first this idea may sound like a serious mind-bender, in reality it is a rather short program that is not too tough to do in IBCM. This is not a fully general program, it is a carefully crafted program that will only print itself out. The program may contain very specific information such as a variable that is initialized to contain the length of the program. For example, if your quine is 25 lines long (data and instructions), then when it runs, it will print out 25 lines where each line consists of four hex digits. The 25 lines you print out may differ from the original file read into the IBCM simulator in a couple of places – these may be variables and instructions which you have modified between the time the program was loaded and the time that particular line is printed. It is possible to write this program in as few as 9 lines of IBCM code, but most likely you will have closer to 20 lines.

CS 216: Program and Data Representation Checkoff

Date:	Circle: Your 1 Tue 7:30 – 9:15 PM Section: 2 Tue 12:15 – 2:00 PM
My <u>Email</u> Address: (UVa ID please, no aliases)	My Name:
Partner's <u>Email</u>:	My Partner's Name:

(One check-off sheet per person)

<u>Lab 6: IBCM Programming</u>	
Demonstrated pre-lab “addition”	
Demonstrated pre-lab “array”	
Demonstrated in-lab functionality	
Explained and exhibited in-lab implementation artifacts	

Pledge:

TO BE TURNED IN BEFORE LEAVING LAB