CS 216

Laboratory 7

IBCM Programming

Week of 17 March 2008

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 small instruction set, the IBCM can compute anything that a modern "powerful" computer can compute.

Reading(s):

1. Read the slides on IBCM.
2. Read IBCM *Principles of Operation* document (posted on the Collab site)
3. Read the IBCM *Programming Examples* handout (posted on the Collab site)
4. Run (install if necessary) the IBCM simulator. (Already installed on ITC lab machines. An installable version is on the Collab site.)

Procedure

Pre-lab

1. The pre-lab is **NOT** a partner exercise. Sorry!
2. Download and install the IBCM simulator from the Collab site, if desired. We only have a Microsoft Windows version; however, it is installed on all the ITC machines.
3. Write the two IBCM programs described in the pre-lab section: addition.ibcm and array.ibcm.
4. Files to download: the IBCM simulator (vb-ibcm.exe and/or vb-ibcm-setup.exe), as well as the other IBCM files (sample programs, documentation, etc.)
5. Files to submit: addition.ibcm, array.ibcm

In-lab

1. On the machines in Olsson 001, there should be a shortcut to the IBCM simulator on the desktop. If not, you can locate the simulator at C:\Program Files\vb-ibcm.exe.
2. Find a partner to work with. It should be somebody you have never worked with before.
3. Implement the bubble sort algorithm in IBCM from the C++ code. See the in-lab section for details.
4. Discuss any problems getting your programs to work with a TA.
5. If you are unable to finish the lab during the in-lab time, submit an extension request. You are allowed to work on the in-lab (only!) with your partner after the lab, of course.
6. Files to download: bubblesort.cpp
7. Files to submit: bubblesort.ibcm

Post-lab

1. The post-lab is an individual project, not a group project.
2. Complete the rest of the shell scripting tutorial found at http://steve-parker.org/sh/sh.shtml (through case statements).
3. Complete the shell script described in the post-lab section.
4. Implement a "quine" in IBCM *individually* (see "*What is a Quine?*" section, below) into a quine.ibcm file.
5. You may discuss this question with your classmates, as long as it is high-level design issues, not IBCM-specific implementation issues (i.e. what you turn in should be your own work).
6. Submit a report, called postlab7.doc (a Word file, NOT a text file), that contains your thoughts on IBCM. What did you think? How easy was it to use? Would modifications to the simulator make life easier for you? How confident do you feel in writing IBCM code? A quarter to half a page is fine.
7. Files to download: none
8. Files to submit: someshellscript.sh, quine.ibcm, postlab7.doc

Pre-lab

Using the IBCM Simulator

You will be developing a somewhat more complicated IBCM program in the in-lab. So, as preparation, study the IBCM documentation and lecture notes. Additionally, you will be expected to come to the in-lab with two IBCM programs. You will not get much from the lab unless you are thoroughly familiar with the documentation and lecture notes — so make sure you are familiar with the material!

The Collab site has the IBCM simulator executable file. There are instructions for both if you have Visual Basic installed on your machine, and 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 posted on the course website and discussed in class illustrate a good way of doing this, where there are columns for each of the following:

1. the hexadecimal 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 hexadecimal 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.

The IBCM Simulator

Download the IBCM simulator from the Collab site (vb-ibcm.exe). It should run on any computer with Visual Basic installed. If not, install the vb-ibcm-setup.exe file. At some point, it will mention that a DLL on your computer is newer than the one it wants to install – you want to keep the newer version. You should use the IBCM simulator to test and validate your pre-lab programs.

Input is funny in the simulator – you will have to hit 'enter' a few extra times after entering input for it to 'take'.

For the pre-lab, you will need to write two IBCM programs, as described below. Note that the programs will need to have an .ibcm extension when submitting, but they are text files, so you can still edit them in Emacs.

addition.ibcm

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.)

array.ibcm

Write a second 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

Submitting your code

Like the previous code that you have submitted, you can also compile IBCM code through the submission system. The compile command will only look at the left-most four characters for each command, and will ignore the remainder of the line. Make sure your code compiles on the submission system!

It MUST have comments in the file so that the TAs can grade it. No comments will earn a zero for the grade.

In-lab

Pick a partner for the in-lab, but somebody that you have NOT worked with before.

Bubble sort

Download and look at the bubblesort.cpp algorithm from the Collab site. This algorithm is what needs to be implemented in IBCM, although you should NOT implement the output in the IBCM version.

To encode this program, follow these steps:

1. Write up high-level pseudo-code for your design on paper (make sure that it is absolutely correct, or you will probably regret it later)
2. Refine this pseudo-code, making it closer to the assembly code level
3. Alter your code into IBCM assembly code with labels instead of addresses
4. Run through a sufficient number of test cases by hand of your IBCM code from step (c) to convince yourself that it is correct
5. Encode into actual hex IBCM code and addresses, and test it using the simulator
6. Identify and fix the errors that you did not pick up in the previous steps

The file should be called bubblesort.ibcm. It MUST have comments in the file so that the TAs can grade it. No comments will earn a zero for the grade.

Post-lab

What is a quine

Based on the experience from the 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.

That them there shell script

You will need to write a shell script for this post-lab. However, in order to get the lab out in time, we haven't written this part yet – but it will be completed by the time of the in-lab (and thus well in time for the post-lab).