1-1-2010

Simulator of IBM Mainframe Programming Environment

Chi Zhang

NORTHERN ILLINOIS UNIVERSITY

Simulator of IBM Mainframe Programming Environment

A Thesis Submitted to the
University Honors Program
In Partial Fulfillment of the
Requirements of the Baccalaureate Degree
With University Honors

Department of
Computer Science
By
Chi Zhang

DeKalb, Illinois
May 14, 2010

University Honors Program

Capstone Approval Page

Capstone Title (print or type):

Simulator of IBM Mainframe Programming Environment

Student Name (print or type): Chi Zhang

Faculty Supervisor (print or type): Georgia Brown

Faculty Approval Signature: Georgia Brown

Department of (print or type): Computer Science

Date of Approval (print or type): 5/5/2011

HONORS THESIS ABSTRACT

This project is to create a Mainframe Programming Environment Simulator running on PC Operation System (e.g. Linux, Max OS, Windows) that will compile / assemble source code intended for a Mainframe machine. The purpose of this simulator is to compiles / assembles the source code into PC program and tries to produce similar output as if the source code was compiled / assembled on a Mainframe machine. In that way, programmers will no longer need to work on Mainframe remotely while developing their programs.

The entire project is significantly large in scale, so this capstone project only touches a portion of it, well defined and well separated by the functionality -- the parser for the ASSIST assembler. Nevertheless, the current simulator can already fully function as an debugging tool for CSCI 360.

This project will be carried on to accomplish the entire design.

HONORS THESIS ABSTRACT
THESIS SUBMISSION FORM

AUTHOR: Chi Zhang

THESIS TITLE: Simulator of IBM Mainframe Programming Environment

ADVISOR: Georgia Brown ADVISOR'S DEPT: CSCI

DISCIPLINE: CSCI YEAR: 2011

PAGE LENGTH: 6 (+ a 4732-line program)

BIBLIOGRAPHY: No

ILLUSTRATED: No

PUBLISHED (YES OR NO): No LIST PUBLICATION: N/A

COPIES AVAILABLE (HARD COPY, MICROFILM, DISKETTE):

Downloadable Software Package (Linkage Host by Google)

ABSTRACT (100 - 200 WORDS):

See previous page.

Honors Capstone Project:

Simulator of IBM Mainframe Programming Environment

Overview:

This project is to create a Mainframe Programming Environment Simulator running on PC OS (Operation System) such as Linux, Max OS, Windows, that will compile / assemble source code intended for a Mainframe machine. The final goal is to let the developers work on their own PCs when developing and / or testing their programs without worrying about an internet connection to a Mainframe machine. It is also aimed at reduce the teaching cost of IBM Assembler, COBOL, and other Mainframe languages.

Purpose of the Project:

The purpose of this simulator is **not** to emulate the Mainframe hardware or its OS. Rather, it compiles / assembles the source code into PC programs and tries to produce similar outputs as if the source code were compiled / assembled on a Mainframe machine. That means programmers will no longer need to work on a Mainframe remotely while developing their programs. The problem of working remotely is that the internet connection can be slow or unstable sometimes (especially around the due date of an assignment, if we are talking about students doing course projects). Also, it is hard to recover files when the system (remotely or locally) or internet connection goes down. From my experience, when a 3270 terminal connection is lost, any unsaved contents will be lost as well. This simulator will, to a great degree, solve potential connectivity problems. This simulator also enables the programmers to work offline when an internet connection is unavailable.

This project is not intended to replace a remote 3270 terminal, since the online TSO/ISPF manipulation still has an important role in working in a Mainframe environment. It is a supplementary utility to simplify the developing and debugging processes of programming on a Mainframe. It really does not matter that much whether the source code was written using the TSO/ISPF editor on a 3270 terminal or it was developed and tested on a PC and uploaded to the Mainframe. What matters is the final product can run on the Mainframe successfully. Having said that, however, the final code should still be tested on a Mainframe machine before submitting as a finished product.

Scope of the Project:

Although the final goal is to simulate the entire Mainframe programming environment, to fit the project into a one-semester capstone project, the scope is limited to parsing, without going into the executing phase, a subset of IBM Assist Assembler instructions (listed below) with a CLI (command-line interface) driver. Due to the scope of the overall project, it is intended to be carried on either by me or by other people in future independent study projects to accomplish the entire design. A stand-alone GUI (graphic user interface) driver or web-based GUI driver will be offered after more features are introduced in the future.

The list of assembler instructions that are currently implemented are as follows:

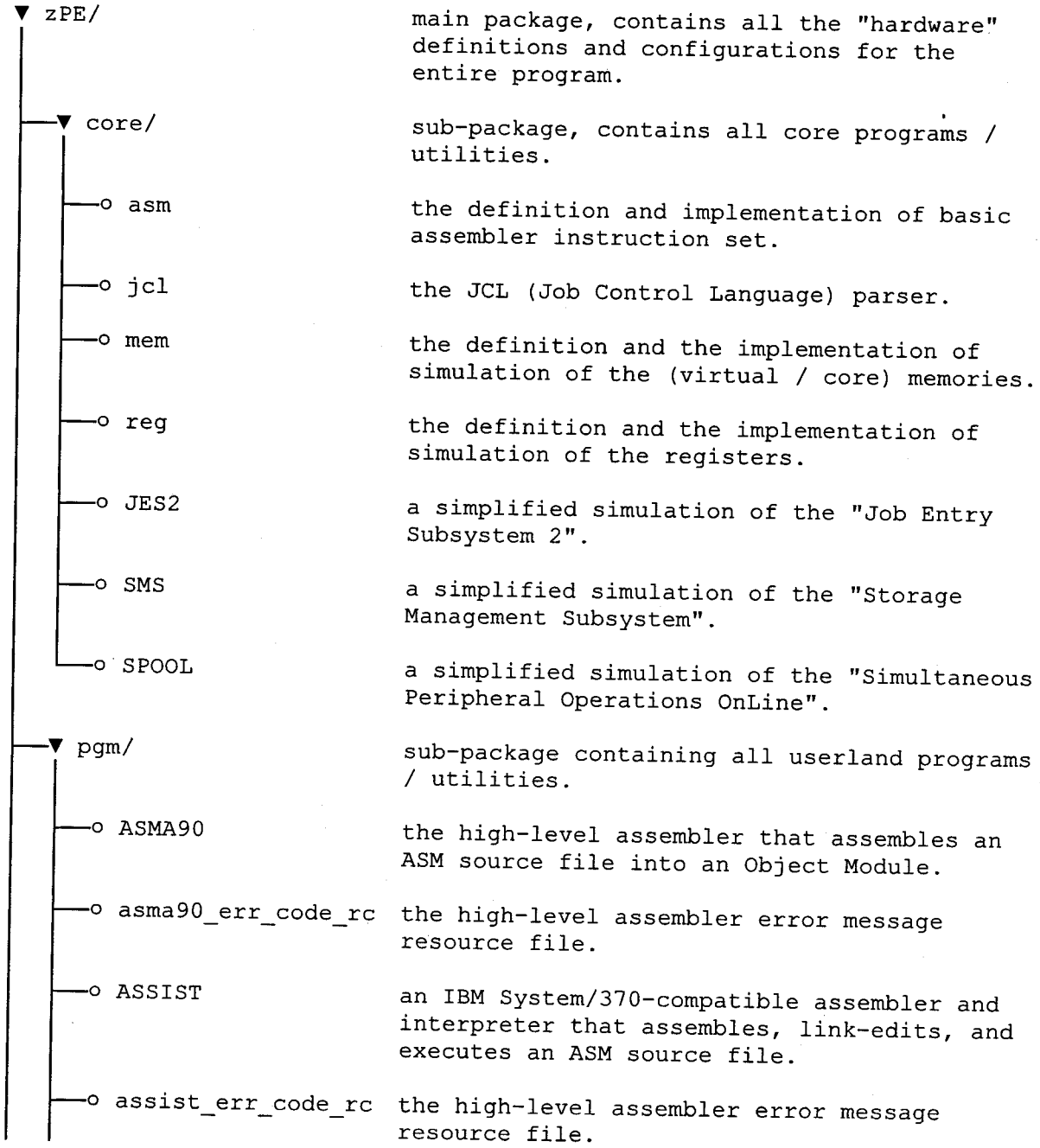
- Part of (Most Frequently Used) Built-in Instructions:
  - CSECT
  - DROP
  - END
  - LTORG
  - USING

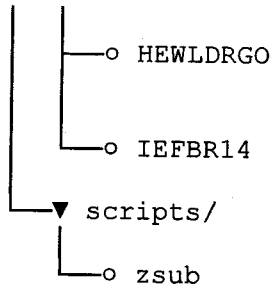
- All RR Instructions ever used in CSCI 360 and CSCI 465:
  - AR
  - BALR
  - BCR
  - BCTR
  - CLR
  - CR
  - DR
  - LCR
  - LR
  - LTR
  - MR
  - NR
  - OR
  - SR
  - XR

- All RX Instructions ever used in CSCI 360 and CSCI 465:
  - A
  - BAL
  - BC
  - BCT
  - C
  - CL
  - D
  - EX
  - IC
  - L
  - LA
  - M
  - N
  - O
  - S
  - ST
  - STC
  - X

Structure of the Software / Package:

This project is designed to be carried on by multiple people, so it requires a logical organizational structure. The software is organized with packages / sub-packages and modules. The modules with a name of all uppercase letters correspond to a program / utility on Mainframe. Otherwise, it is a module only for the simulator (no specific counterpart on Mainframe). The structure of the entire software (current version) is as follows:






```
the loader that link-edits and executes an
Object Module.

a system utility that DOES NOTHING.

sub-package containing all drivers.

a CLI driver that behaves like the "submit"
command on Mainframe.
```

This structure allows for additional modules to be developed and incorporated into the simulated programming environment easily. Say, if later another module is built (following established formatting and rules) to process COBOL source code, it just needs to be placed into the "zPE/pgm/" directory and the module will work as intended. Namely, "zPE/scripts/zsub" can now submit a COBOL source file and produce the right output(s).

Accomplishments:

So far, I have accomplished the following tasks successfully:

- I have been able to correctly parse the JCL (Job Control Language) containing a subset of instructions, which includes all necessary ones that is taught in CSCI 360 and most ones in CSCI 465.
- I have designed and implemented the "virtual memory" implementation and other simulated hardware like General Purpose Registers, PSW (Program Status Word), SPOOL (Simultaneous Peripheral Operations OnLine), etc.
- I have simulated the basic functionality of some important system level subsystems, such as JES (Job Entry Subsystem) and SMS (Storage Management Subsystem).
- I have almost fully finished the High-Level Assembler parser that parse the assembler code with all the features (although only a few of them were touched in CSCI 360 and CSCI 465) supported.
- I have implemented the ASSIST parser to use the result of High-Level Assembler parser so it can support more recent instructions yet produce a "programmer-friendly" report. (This can be viewed as an update of the ASSIST currently on the Mainframe.)

Challenges:

The challenges I encountered so far were:

- On the Mainframe systems like z/OS, files are organized differently in comparison to that on a PC. There is no "folder" in z/OS, for example, as PC OS organized files are based mainly on a tree-structure directory / folder organization. On Mainframe systems, files are organized by nodes (an example of a four-node file name would be "A.BCD.EF.G"). The PDS/PDSE (Partitioned Data Set [Extended]) on z/OS is often referred as a "directory," but what it really looks like is similar to a "tar" file on PC (or you may think it as a "zip" file without compression). Currently, I treat each node as one level of a directory, and did not implement the PDS/PDSE yet. I am planning, though, to implement PDS/PDSE as "zip" files.
- IBM Mainframe systems uses big-endian byte order, while the mainstream PC systems (Windows, Linux) are using little-endian byte order. I came across this problem when I was designing the registers and memories, and finally solved it by manually reversing all the bytes in the correct order.
- Mainframe systems support multiple addressing modes (24 bit, 31 bit, 64 bit, or even larger) simultaneously. Since that depends highly on the hardware design, it is hard to simulate that feature at the software level. Currently, I am using the 24 bit addressing mode, because both ASSIST and High-Level Assembler use it by default. However, I have left APIs for supporting different addressing modes for a later time.
- In parsing the argument for the Assembler, the grammar is so complicated that no simple regular expression (re) can parse the argument line directly. As I went through the parsing process, I gradually developed a set of re-splitting functions that perform the re-splitting based on not only match-pattern but skip-pattern(s). For example, a skip-pattern of "'" + "'" (single quote) will cause the re-searching to ignore anything in between two single quotes, and a skip-pattern of "(" + ")" will cause it to ignore anything in between left and right parentheses.
- When doing implicit addressing using labels, the assembler on the Mainframe does what is called "active using." This concept, although having plenty of documentation, is significantly hard to understand. Even if I have overcome the difficulties in implementing the "active using," it will still take time for me to recall how it works. If later it needs to be modified for some reason, I would expect to have no less struggle than I have already had.

- All the calculations of expressions in the High-Level Assembler on the Mainframe is done at the bit level, and the length of the result is critical. Every number has a length that depends only on the magnitude of the number itself. On PC OS, however, every number has a pre-defined length that aligned on a "full-word" boundary, meaning that any number are 4 byte (32 bit) long. It took me weeks to figure out a way to calculate and store length information during the calculation of expressions, and several days to actually implement it. The up side is that a bunch of cool features never touched in CSCI 360 were made available automatically.

The potential challenges that I expect to encounter in the future are:

- Currently, only simple instructions are considered, but when stepping into I/O instructions/pseudo-instructions, many issues need to be solved. The biggest challenge in this part should be the method to synchronize real files on the system with the virtual files in this software. I have already designed and implemented a part of it, but it is still problematic at this stage.
- Once I have stepped into the executing phase, interacting with PSW (program status word) and exception handling will be problematic. I have already designed and implemented the APIs for PSW, but never tested it with real programs.
- When executing an object module, all books / documentation I have read so far suggest using a big switch to simulate the decoder. This is hard to read and hard to maintain or modify. I plan to embedded the actual action in the instruction map itself so no switch is needed. I have started to practice that technique already, but there are still several difficulties that need to be solved in order to get this technique working.

Availability:

Downloading Page:

http://code.google.com/p/mainframe-env-simulator/downloads/detail?name=Mainframe_PE_v.0.0.1.zip&can=2&q=

Direct Downloading Link:

http://mainframe-env-simulator.googlecode.com/files/Mainframe_PE_v.0.0.1.zip

All historical and future version(s) can be found at:

http://code.google.com/p/mainframe-env-simulator/downloads/list