A GENERAL-PURPOSE CUSTOM-DESIGNED ASSEMBLER IN C

Bo Hatfield[1], *Mei Zhang*[2], *Lan Jin*[3]

Abstract *- In this paper, we introduced the design and implementation of a general-purpose assembler that can be written and modified to be adaptive for any given assembly language to generate the target machine code and carry out the execution. The system consists of a parsing module and an execution module. The former generates the target machine code from an inputted assembly language code, and the latter simulates the hardware operation of a computer and executes the instructions. Due to its versatility and flexibility, this assembler allows easy modification to reconfigure itself for implementing any given instruction set architecture. As the major objective of this project, the assembler can help students to learn assembly-language programming and develop an in-depth understanding of its close relationship with the instruction set architecture.*

Index Terms – General-Purpose Assembler, Simulation, Fetch-Execute Cycle, Data Structures.

The assembly-language programming is one of the important knowledge areas of Computer Science curriculum. Since assembly languages are machine-dependent, we may need to use assembler of a specific type of machine for students' laboratory activities. This approach has the advantage of a closer relationship to reality, but may lead to some problems, such as (1) high cost of buying and upgrading the computer and its assembler of a specific type, and (2) additional time for teaching students machine-specific knowledge about the computer and its assembler. These problems have motivated us to take another approach, i.e., to provide students a general-purpose assembler that has no association with a specific type of machine, but can be custom-designed to fit the special demand of an assembly-language programming course. This approach has the following advantages:

- It is possible to choose a fictitious educational computer as the target machine for writing a customized assembler that emphasizes the basic principle of its instruction set architecture rather than the details of the complex structure of a real machine.
- The simple code of the assembler allows students to not only use it in programming, but also study the control process of an instruction set. Students can even modify it to implement any special instructions that we may wish to add into the assembler.
- With the support of this assembler, the assembly-language programming course may be taught on a more flexible and wider basis of instruction set architecture than teaching a single type of machine with a fixed instruction set. The new teaching methodology helps to avoid the possible phenomenon that some students may have learned the writing of assembly-language programs on a specific type of machine, but still find trouble in writing programs on a new given instruction set with a different architecture.

Motivated by the aforementioned objectives, we developed a new methodology of writing an assembler that can be reconfigured to adapt any given instruction set architecture. As a case study, we developed a general-purpose custom-designed assembler named GPCDA, using the DLX processor [1] as its target machine. DLX is a well-known educational processor with a typical RISC instruction set.

The execute module of GPCDA simulates a complete Fetch-Execute cycle of an instruction [3] described as follows:

```
haltFlag=FALSE;       /* haltFlag=0 allows execution */
PC=0;                 /* assume start address 0 */
while (!haltFlag){    /* while no exception to halt CPU */
  IR=memory[PC];      /* fetch the instruction into IR */
  PC=PC+sizeOfInstr.; /* update the PC for next fetch */
  execute(IR);}       /* execute the current instr. in IR */
```

where *execute(IR)* is the function which is heavily dependent on the instruction set, and we need to choose the most suitable data structures for describing various types of data and instruction formats. The data types of *pointer* and *struct* in C are used most frequently in this assembler.

The parsing module goes through two passes of the source code, performing syntax checking, symbol table creation, machine code conversion, and program preprocessing.

Besides ordinary DLX programs, three classes of complex arithmetic operations, multiply, divide, and floating-point add, have been programmed for testing the assembler [2, 4].

REFERENCES

[1] Hennessy, J, L. and D, A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan-Kauffmann Publishers, Inc., 1995.

[2] Jin, L. and B, Hatfield, *Computer Organization: Principles, Analysis, and Design*, Tsinghua University Press, 2003.

[3] Nutt, G, *Operating Systems*, Addison Wesley Longman, Inc., 2000.

[1] Bo Hatfield, Dept. of Computer Science, Salem State College, 352 Lafayette Street, Salem, MA 01970, bo.hatfield@salemstate.edu
[2] Mei Zhang, Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, meiz@cadence.com
[3] Lan Jin, Dept. of Computer Science, California State University, Fresno, CA 93740, jin@satyrs.csufresno.edu

[4] Zhang, M, "A General-Purpose Custom-Designed Assembler in C – A Case Study of the DLX Instruction Set Architecture", *MS Project Report*, California State University, Fresno, 2000.