4N-1

A General Purpose Assembler for Queue Computers

CANEDO Arquimedes†, ABDERAZEK Ben†,
YOSHINAGA Tsutomu†, SOWA Masahiro†

1. Introduction

Different types and implementations of queue computers have been developed and researched at The Advanced Distributed and Parallel Computer Systems Laboratory[1]. QASM is a general purpose two-pass assembler designed to generate object code for multiple queue-based microprocessors and to provide a suitable tool for both programmers and compiler assembly output. It is capable to perform macro expansion, pseudoinstructions assembly and machine dependent optimizations.

2. QASM

The assembly process takes place in two passes. The first pass reads the input file statement by statement and fills the symbol table and program image. Forward references are solved using a backpatch list once the input file has been completely read. Program image is an intermediate language form based on generalized quadruples. The second pass reads the machine description file of the target architecture and performs pattern matching for every statement found in the program image. All machine dependent optimizations and type checking are achieved in the second pass. QASM is entirely written in C language using flex and bison to build lexical analizer and parser.

2.1. Preprocessor

In case that macro expansion capability has to be used, the programmer provides QASM a library file which contains all macro expansion definitions for a given program. Preprocessor adds an extra pass over the input program. Preprocessor (QKK) default behavior generates a expanded assembly file for every input program and it is capable to merge all input files into one single output assembly file.

† Graduate School of Information Systems,
University of Electro-Communications, Tokyo

2.2. Intermediate Representation

The intermediate representation of program was chosen according to two main functions the assembler must provide: (1) to satisfy all queue-computer architecture features and (2) to be easily retargetable to new queue-based computer architectures [4,5]. Each instruction is represented in a quadruple with at least opcode field. Up to three operands can be encoded into each quadruple: destination operand and two sources. This intermediate form allows the QASM to perform efficient pattern matching and translation to binary code.

2.4. Porting Mechanism

For each target architecture supported by QASM a machine description file has to be defined. The machine description file contains one by one all existing instructions in the target architecture in a form of extended quadruple described in section 2.3. The opcode field defines the mnemonic, binary value, and offset that specifies the starting position of opcode field within every particular instruction. As for the destination and source fields, the data type expected (e.g. General Purpose Register, Queue Register, Immediate), the offset defining the position of field, and bit size of field is provided in the definition.

Second pass of assembler is machine dependent and can be controlled at runtime by a special directive (.syntax TARGET-ARCHITECTURE) in the source program. This feature is particularly usefull for assembling programs for hybrid queue processors.

The second pass over the program image attempts to match every instruction (quadruple) found in the program image with instruction definitions in the machine description files (extended quadruples). If a match is found the following steps are bounds checking, type checking and machine dependent optimizations if available for the matched instruction.

2.2. Optimizations

QASM distinguishes between two types of optimizations: user defined and machine dependent optimizations. User defined optimizations are activated by programmer when invoking the assembler and are not critical for assembly process. Machine dependent optimizations are transformations the assembler must do to source program to guarantee correctness of assembled program.

2.3. Special Features

All design features and decisions in QASM were made acording the following keypoints:

- Assemble multiple input syntaxes.
- Provide a tool suitable for compiler and a intuitive and simple framework for human programmers.
- Easy retargetable and portable assembler for queue computers and hybrid queue-register computers.
- Multiple output for different target architectures.
- Multiple object file formats for hardware and software simulators and linker.

In order to generate usefull data for hardware and software simulators QASM provides simulator dependent object file formats, hexadecimal object file, and raw binary object file. In a near future QASM will support ELF[2] object file format.

Figure 1. depicts QASM general structure in block diagram.

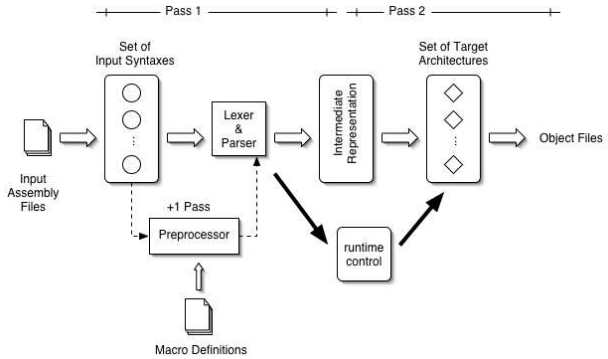

Figure 1. QASM block diagram.

3. Conclusions and Future Work

QASM has been finely tuned for three different queue architectures and supports both high-level human programmer input assembly syntax and compiler generated assembly output. It has been successfully interfaced with GCC port for PQP and Queue Java Compiler[3]. QASM is a custom made, general purpose, portable assembler developed to fulfill all queue computer architecture features and needs.

QASM has been successfully compiled on MacOSX, FreeBSD, OpenBSD, Linux, Solaris operating systems using GCC 3.4 compiler.

A fragment of quicksort benchmark assembled program for multithreaded PQP is shown in Figure 2.

```
3e 00 // Optimized
2c 24 // Optimized
1b 00 // call0
07 00 // hlt
ac 0c // incr 10
78 00 // stw0
42 01 // mvq 1
...
42 00 // mvq 0
40 04 // ldil 4
80 00 // add
43 00 // mvr
60 00 // ldw0
```

Figure 2. Object file output fragment of quicksort benchmark for MTPQP.

4. References

[1] Advanced Distributed and Parallel Computer Systems Laboratory, University of Electro-Communications, Tokyo

[2] ELF: Executable and Linkable Format, Tool Interface Standards (TIS), Portable Formats Specification, v1.1

[3] 生産順序キューマシン命令コード生成手法の提案, 川島祐介, 繁田聡一, 吉永努, 曽和将容, 第66回情報処理学会全国大会 論文集 (1)pp81-82, March 2004.

[4] Parallel Queue Processor Architecture Based on Produced Order Computation Model, (to appear in the International Journal of Supercomputing 2005)

[5] 曽和将容, 省電力大並列高速並列キュープロセッサ PQP, Technical Report, SLL030331, 電気通信大学情報システム学研究科, June 2003