The Queue Computer Project

Qasm

Technical Report, Ref. TR12010

ASL-Ben Abdallah Group
The University of Aizu,
Adaptive Systems Laboratory, School of Computer Science and Engineering

Feb. 2010
Background (1)

• What is Queue Computation?
  – The intermediate data is written into a circular queue register (QREG)
  – A given instruction implicitly reads data from a head of the queue register (QH)
  – The executed result is written into a tail of the queue register (QT).
Background (2)

- Queue computation features
  - High instruction parallelism
  - Small program size
  - No false data dependency
• In our lab, there are two types of Queue processors
  – Queue Core (QC) model
  – Dual Execution Processor (DEP) model

• Increase the these Queue processors usage by to develop Queue Assembler (Qasm)
Qasm features

• Development of Queue Assembler(Qasm)
  – User friendly
  – Support two computing models
    • QC and DEP model
  – Support Queue compiler output
    • With preprocessor
Assembler structure

Qasm

Compiler output

Preprocessor

Intermediate file

Syntax analysis

Generation

Machine language

Converting compiler output file into Intermediate file

Parsing Intermediate file

Converting Intermediate file into Machine language file
Assembler development

- Preprocessor -

1. Read in one line
   \[\text{sub iws, qt, qh, qh+1}\]

2. Divide into tokens
   \[\text{sub iws, qt, qh, qh+1}\]
   Not needed

3. Clean not needed tokens
   \[\text{sub qh+1}\]

4. Clean not need characters
   \[\text{sub qh+1}\]
   \[1\]

5. Combine tokens
   \[\text{sub 1}\]
Assembler development

-Syntax analysis-

1. Syntax parsing Intermediate file and divide into tokens
   
   ```
   main:   add    1    # hoge
   ```

2. Divided tokens into category and substitute to structure
   
   ```
   main:   add    1    #hoge
   ```
   
   Label    Mnemonic    Operand    Comment
Assembler development

- Generation(1) -

1. Replace mnemonic code by opcode

main: `add 1 #hoge`

2. Convert integer into hexadecimal on operand

main: `0x30 1 #hoge`
Assembler development -Generation(2)-

3. Paste address into label

```
.text
setd0  10  # N
setd1  2   # counter
setd2  1   # arg(0)
seld3  1   # arg(1)
seta0  $main
call  0(a0)
hlt

main:
ldw   0(d0)
ldil  2
```
Assembler development

-Generation (2)-

3. Paste address into label

```
.text
setd0 10  # N
setd1 2   # counter
setd2 1   # arg(0)
setd3 1   # arg(1)
seta0
call
halt

.main:
  lw     0(a0)
  ldil

 Find a Label
```
3. Paste address into label

```
.text
setd0 10  # N
setd1 2  # counter
gtext 1  # arg(0)
setd3 1  # arg(1)
seta0
call
hlen

main:  
ldw 0\(a0)\)
ldil 0\(d0)\)
```

address = 0x0c

Search for jump address
Assembler development

-Generation(2)-

3. Paste address into label

```
.text
setd0 10 # N
setd1 2 # counter
setd2 1 # arg(0)
setd3 1 # arg(1)
seta0
call hlt

main:
ldw
ldil 0(d0) 2
```

address = 0x0c

Change the label to address
Design result(1)

Qasm

- qasm.c, qasm.h, target.h

Preprocessor

- preprocessor.c, preprocessor.pl

Syntax analysis

- parser.y, scanner.l

Generation

- pqp_codegen.c
  - QC.h, DEP.h
- conversion.c
- symtab.c, backpatch.c
- codegen.c

MD file
## Design result (2)

<table>
<thead>
<tr>
<th>File</th>
<th>Number of lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>qasm.c</td>
<td>400</td>
</tr>
<tr>
<td>qasm.h</td>
<td>148</td>
</tr>
<tr>
<td>target.h</td>
<td>37</td>
</tr>
<tr>
<td>preprocessor.c</td>
<td>62</td>
</tr>
<tr>
<td>preprocessor.pl</td>
<td>99</td>
</tr>
<tr>
<td>parser.y</td>
<td>226</td>
</tr>
<tr>
<td>scanner.l</td>
<td>348</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File</th>
<th>Number of lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>pqp_codegen.c</td>
<td>579</td>
</tr>
<tr>
<td>QC.h</td>
<td>420</td>
</tr>
<tr>
<td>DEP.h</td>
<td>205</td>
</tr>
<tr>
<td>conversion.c</td>
<td>86</td>
</tr>
<tr>
<td>symtab.c</td>
<td>332</td>
</tr>
<tr>
<td>backpatch.c</td>
<td>105</td>
</tr>
<tr>
<td>codegen.c</td>
<td>171</td>
</tr>
</tbody>
</table>

Total number of lines: 3218
Evaluation results (1)

- Qasm executes
  - Reading and writing file assignment
  - Select using compiler output or handmade assembly
  - Decide file type of input and output
  - With user interface

```
std4dc17{s1140196}155: ./qasm

Queue Computer Project
University of Aizu

Queue Assembler
Optimized by Reo Honjoya, Hiroki Hoshino
Last update: Feb 04, 2010

# Do you use compiler output file?
# ( 1:Yes 2:No )
> 2
# Enter a assembly file name
> ./Assembly/TEMP/optimize.s
# Enter ISA ( 1:QC model 2:DEP model )
> 1
# Enter a output file name
> ./Out/com_out.hex
# Enter output file type ( 1:Binary 2:Hexadecimal )
> 2
14 bytes written to file: ./Out/com_out.hex
```

std4dc17{s1140196}156: 

---

16
Evaluation results(2)

Queue compiler generated code for \( \frac{a^2}{(b-a)-a} \)

\[
\begin{align*}
\text{.text} \\
\text{main:} \\
\text{ld} & iws, qt, (fp)0 \\
\text{ld} & iws, qt, (fp)4 \\
\text{sub} & iws, qt, qh, qh+1 \\
\text{mul} & iws, qt, qh-1, qh-1 \\
\text{sub} & iws, qt, qh, qh+1 \\
\text{div} & iws, qt, qh, (fp)8 \\
\text{st} & iws, qh, (fp)8
\end{align*}
\]

Intermediate file

```
main:
ldw  0(d0)
ldw  0(d1)
sub  1, 1
mul  1, 0
sub  1, -1
div  1, 1
stw  0(d2)
```

Assemble

```
60 00  // ldw
61 00  // ldw
82 81  // sub
b8 80  // mul
82 ff  // sub
ba 81  // div
7a 00  // stw
```