The Queue Computer
Project

Instruction Set Architecture

© Abderazek Ben Abdallah
Graduate School of Information Systems
The University of Electro·communications, Tokyo
January 2003
COVOP

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>covop addr1</td>
<td>Convey an address</td>
<td>0</td>
<td>0</td>
<td>00000100</td>
</tr>
</tbody>
</table>

**Description:**
Convey an 8-bit address to a load or store instruction to extend the addressing bits from 8 to 16 bits.

Here, QH = 0 & QT = 0

Load & Store

**Instruction Format**

| 6 | 2 | 8 |
Load byte

**ldb**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldb d addr0</td>
<td>qtw←m((d)+addr1.addr0)</td>
<td>0</td>
<td>1</td>
<td>010000</td>
</tr>
</tbody>
</table>

**Description:**
Load byte to the operand queue pointed by the QT from memory address ((d)+addr0) or from ((d) + addr1.addr0) if the load instruction follows a convey instruction.
Here, QH = 0 & QT = 1

Load byte unsigned

**ldbu**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldbu d addr0</td>
<td>qtw←m((d)+addr1.addr0)</td>
<td>0</td>
<td>1</td>
<td>010001</td>
</tr>
</tbody>
</table>
**Description:**
Load byte unsigned to the operand queue pointed by the QT from memory address ((d) + addr0) or from ((d) + addr1.addr1) if the load instruction follows a convey instruction.
Here, QH = 0 & QT = 1

### Load string

<table>
<thead>
<tr>
<th>lds</th>
<th>d</th>
<th>addr0</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>lds addr0(d)</td>
<td>qtw←m((d)+addr1.addr0*2)</td>
<td>0</td>
<td>1</td>
<td>010010</td>
</tr>
</tbody>
</table>

**Description:**
Load string to the operand queue pointed by the QT from memory address ((d) + addr0) or from ((d) + addr1.addr1) if the load instruction follows a convey instruction.
Here, QH = 0 & QT = 1

### Load string unsigned

<table>
<thead>
<tr>
<th>ldsu</th>
<th>d</th>
<th>addr0</th>
</tr>
</thead>
<tbody>
<tr>
<td>010011</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldsu addr0(d)</td>
<td>qtw←m((d)+addr1.addr0*2)</td>
<td>0</td>
<td>1</td>
<td>010011</td>
</tr>
</tbody>
</table>
Description:
Load string unsigned to the operand queue pointed by the QT from memory address
((d)+ addr0) or from ((d) + addr1.addr1) if the load instruction follows a convey
instruction.
Here, QH = 0 & QT = 1

Load word

\[ \text{ldw} \]

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldw addr0(d)</td>
<td>qt_tw&lt;-m((d)+addr1.addr0*4)</td>
<td>0</td>
<td>1</td>
<td>010100</td>
</tr>
</tbody>
</table>

Description:
Load word to the operand queue pointed by the QT from memory address ((d)+ addr0)
or from ((d) + addr1.addr1) if the load instruction follows a convey instruction.
Here, QH = 0 & QT = 1

Store byte

\[ \text{stb} \]

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>stb addr(d)</td>
<td>qhw&lt;-m((d)+addr0.addr1)</td>
<td>1</td>
<td>0</td>
<td>010110</td>
</tr>
</tbody>
</table>

Description:
Store byte to the operand queue pointed by the memory address ((d)+ addr0) or
((d) + addr0.addr1) from QH if the store instruction follows a convey instruction.
So, here QH = 1 & QT = 0
### Store byte unsigned

*stbu*

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>stbu addr(d)</td>
<td>qhw→m((d)+addr1.addr0)</td>
<td>1</td>
<td>0</td>
<td>001110</td>
</tr>
</tbody>
</table>

**Description:**
Store byte unsigned to the operand queue pointed by the memory address ((d)+addr0) or ((d) + addr0.addr1) from QH if the store instruction follows a convey instruction.
So, here QH = 1 & QT = 0

### Store string

*sts*

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>sts addr(d)</td>
<td>qhw→m((d)+addr1.addr0*2)</td>
<td>1</td>
<td>0</td>
<td>010111</td>
</tr>
</tbody>
</table>

**Description:**
Store string to the operand queue pointed by the memory address ((d)+addr0) or ((d) + addr0.addr1) from QH if the store instruction follows a convey instruction.
So, here QH = 1 & QT = 0
Store string unsigned  \textit{stsu}

<table>
<thead>
<tr>
<th>\textbf{Format}</th>
<th>\textbf{Action}</th>
<th>\textbf{QH}</th>
<th>\textbf{QT}</th>
<th>\textbf{Binary}</th>
</tr>
</thead>
<tbody>
<tr>
<td>stsu addr(d)</td>
<td>qhw→m((d)+addr1.addr0*2)</td>
<td>1</td>
<td>0</td>
<td>001111</td>
</tr>
</tbody>
</table>

**Description:**
Store string unsigned to the operand queue pointed by the memory address ((d)+addr0) or ((d) + addr0.addr1) from QH if the store instruction follows a convey instruction.
So, here QH = 1 & QT = 0

Store word  \textit{stw}

<table>
<thead>
<tr>
<th>\textbf{Format}</th>
<th>\textbf{Action}</th>
<th>\textbf{QH}</th>
<th>\textbf{QT}</th>
<th>\textbf{Binary}</th>
</tr>
</thead>
<tbody>
<tr>
<td>stw addr(d)</td>
<td>qhw→m((d)+addr1.addr0*4)</td>
<td>1</td>
<td>0</td>
<td>011000</td>
</tr>
</tbody>
</table>

**Description:**
Store word to the operand queue pointed by the memory address ((d)+addr0) or ((d) + addr0.addr1) from QH if the store instruction follows a convey instruction.
So, here QH = 1 & QT = 0

**Immediate**
**Instruction Format:**

```
8     8
```

**Load immediate value**

Load immediate value to the operand queue pointed by the QT from memory address 
((d)+ addr0) or from ((d) + addr0.addr1) if the load instruction follows a convey instruction.
Here, QH = 0 & QT = 1

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldil</td>
<td>qtw(0-7bit)---value</td>
<td>0</td>
<td>1</td>
<td>00100010</td>
</tr>
</tbody>
</table>

**Description:**
Load immediate value to the operand queue pointed by the QT from memory address 
((d)+ addr0) or from ((d) + addr0.addr1) if the load instruction follows a convey instruction.
Here, QH = 0 & QT = 1

**Load immediate address**

```
1a
```
Description:
Load immediate address to the operand queue pointed by the QT from memory address ((d)+ addr0) or from ((d) + addr0.addr1) if the load instruction follows a convey instruction.
Here, QH = 0 & QT = 1

Control
Instruction Format

Branch

\[
\begin{array}{c|c}
8 & 8 \\
\end{array}
\]

Jump

\[
\begin{array}{c|c|c}
6 & 2 & 8 \\
\end{array}
\]

Interrupt, Barrier

\[
\begin{array}{c|c}
8 & 0 \\
\end{array}
\]

Queue Control

\[
\begin{array}{c|c}
8 & 8 \\
\end{array}
\]

Branch target

\[b \ t\]
Description:
This instruction to the operand queue pointed by the program counter \(((\text{pc}) + t)\) or \(((\text{pc}) + \text{addr1} \cdot t)\) to FC if the branch instruction follows a convey instruction.
Here QH = 0 & QT = 0

Branch equal beq \textit{t}

Description:
This instruction to the operand queue pointed by the program counter \(((\text{pc}) + t)\) or \(((\text{pc}) + \text{addr1} \cdot t)\) to FC if the branch instruction follows a convey instruction.
Here QH = 0 & QT = 0, 'CC' means conditional code.

Branch not equal bne
**Description:**
Branch not equal to target the operand queue pointed by the program counter ((pc)+ t) or ((pc) + addr1.t) to FC if the branch instruction follows a convey instruction.
Here QH = 0 & QT = 0 'CC' means conditional code

**Branch less than**

<table>
<thead>
<tr>
<th>bne</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>bne t</td>
<td>fc←(pc)+2(addr1.t),if cc=ne</td>
</tr>
<tr>
<td>QH</td>
<td>QT</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to target the operand queue pointed by the program counter ((pc)+ t) or ((pc) + addr1.t) to FC if the branch instruction follows a convey instruction.
Here QH = 0 & QT = 0 'CC' means conditional code

**Branch less than or equal**

<table>
<thead>
<tr>
<th>blt</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>blt</td>
<td>fc←(pc)+2(addr1.t),if cc=lt</td>
</tr>
<tr>
<td>QH</td>
<td>QT</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**ble t**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ble t</td>
<td>fc←(pc)+2(addr1.t),if cc=lte</td>
<td>0</td>
<td>0</td>
<td>00001001</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to target the operand queue pointed by the program counter ((pc)+ t) or ((pc) + addr1.t) to FC if the branch instruction follows a convey instruction. Here QH = 0 & QT = 0 'CC' means conditional code

**bgt t**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>bgt t</td>
<td>fc←(pc)+2(addr1.t),if cc=gt</td>
<td>0</td>
<td>0</td>
<td>00001010</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to target the operand queue pointed by the program counter ((pc)+ t) or ((pc) + addr1.t) to FC if the branch instruction follows a convey instruction. Here QH = 0 & QT = 0 'CC' means conditional code

**bge t**

**Branch greater than or equal**
Description:
This instruction to target the operand queue pointed by the program counter ((pc) + t) or ((pc) + addr1.t) to FC if the branch instruction follows a convey instruction.

Here QH = 0 & QT = 0 'CC' means conditional code

Stop QH move

<table>
<thead>
<tr>
<th>bge</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001011</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>bge t</td>
<td>fc←(pc)+2(addr1.t),if cc=gt</td>
<td>0</td>
<td>0</td>
<td>00001011</td>
</tr>
</tbody>
</table>

Stop LQH move

<table>
<thead>
<tr>
<th>stplqh n</th>
</tr>
</thead>
<tbody>
<tr>
<td>010111100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>stplqh n</td>
<td>Stop LQH moving</td>
<td>9</td>
<td>0</td>
<td>010111100</td>
</tr>
</tbody>
</table>
**Description:**
Stop LQH move to target the operand queue pointed that the LQH stop to move from $n^{th}$ position.
Here QH = 0 & QT = 0

**Fixed QH automatically**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>stplqh n</td>
<td>Stop LQH moving</td>
<td>0</td>
<td>0</td>
<td>00000101</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to target the operand queue pointed that the QH will be fixed in the current $n^{th}$ position.
Here QH = 9 & QT = 0

**Fixed LQH automatically**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>autqh n</td>
<td>Fixed QH automatically</td>
<td>0</td>
<td>0</td>
<td>01011101</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to target the operand queue pointed that the LQH will be fixed in the current $n^{th}$ position.
Here QH = 0 & QT = 0
autlqh
00000110

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>autlqh n</td>
<td>Fixed LQH automatically</td>
<td>0</td>
<td>0</td>
<td>00000110</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to target the operand queue pointed that the LQH will be fixed in the current n\textsuperscript{th} position.
Here QH = 9 & QT = 0

---

**Jump**

jump t(a)

<table>
<thead>
<tr>
<th>jump</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001100</td>
<td>15</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump t(a)</td>
<td>fc←(a)+2(addr1.t)</td>
<td>0</td>
<td>0</td>
<td>00001100</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to target the operand queue pointed by the memory address ((a)+ t) or ((a) + addr1.t) to FC if the jump instruction follows a convey instruction.
Here QH = 0 & QT = 0 ‘(a)’ means the content of address register ‘a’

---

**Call**

call t(a)
Description:
This instruction to target the operand queue pointed by the memory address ((a) + t) or ((a) + addr1.t) to FC if the call instruction follows a convey instruction.
Here QH = 0 & QT = 0     ‘(a)’ means the content of address register ‘a’

Return from call

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>call t</td>
<td>fc ← (a) + 2(addr1.t)</td>
<td>0</td>
<td>0</td>
<td>00001101</td>
</tr>
</tbody>
</table>

Description:
This instruction to target the operand queue pointed by the current return address to FC.
Here QH = 0 & QT = 0

Return from interrupt

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>rfc</td>
<td>fc ← cra</td>
<td>0</td>
<td>0</td>
<td>01070101</td>
</tr>
</tbody>
</table>
Description:
This instruction to target the operand queue pointed by the interrupt return address (ira) to FC.
Here QH = 0 & QT = 0

No operation

Description:
nop is a dummy instruction that has no effect. It can be useful as an explicit 'do nothing' instruction.

Halt (stop)
**Description:**
This barrier instruction to target the operand queue pointed to stop fetching, to stop decoding.
Here QH = 0 & QT = 0

**Barrier**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>hlt</td>
<td>Stop fetch, decode</td>
<td>0</td>
<td>0</td>
<td>00000010</td>
</tr>
</tbody>
</table>

**Description:**
This barrier instruction to target the operand queue pointed that 'wait' until all the previous instructions are executed.
Here QH = 0 & QT = 0

**Serial on/off**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>hlt</td>
<td>Stop fetch, decode</td>
<td>0</td>
<td>0</td>
<td>00000010</td>
</tr>
</tbody>
</table>
**Description:**
This barrier instruction to target the operand queue pointed to execute serially on for serial on and multiple out of order for serial off.
Here $QH = 0$ & $QT = 0$

Switch (Program mode selector)  

<table>
<thead>
<tr>
<th>sonf</th>
<th>on/off</th>
</tr>
</thead>
<tbody>
<tr>
<td>01100001</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>$QH$</th>
<th>$QT$</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>sonf</td>
<td>Serial On/Off Begin &amp; end serial execution</td>
<td>0</td>
<td>0</td>
<td>01100001</td>
</tr>
</tbody>
</table>

**Description:**
Switch the execution mode by following the “mode” bit (0 indicates the Queue program and 1 indicates the Stack program).
ALU for Single Word

<table>
<thead>
<tr>
<th>opcode</th>
<th>offset (n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Add

add n
Add two operands to the operand queue pointed by the QT from \((qhw0+(qh+n)w)\). Here \(QH = 1\) & \(QT = 1\)

Add Unsigned \texttt{addu n}

Description:
Add two unsigned operands to the operand queue pointed by the QT from \((qhw0+(qh+n)w)\). Here \(QH = 1\) & \(QT = 1\)

Sub \texttt{sub n}
Description:
Subtract two operands to the operand queue pointed by the QT from \((qh^0-(qh+n)w)\).
Here \(QH = 1\) & \(QT = 1\)

Sub unsigned \hspace{1cm} \textbf{subu} \hspace{1cm} \textbf{n}

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Format} & \textbf{Action} & \textbf{QH} & \textbf{QT} & \textbf{Binary} \\
\hline
\textbf{subu} \hspace{0.2cm} n & qtw0\rightarrow qhw0-(qh+n)w & 1 & 1 & \text{00110011} \\
\hline
\end{tabular}
\end{center}

Description:
Subtract two unsigned operands to the operand queue pointed by the QT from \((qh^0-(qh+n)w)\).
Here \(QH = 1\) & \(QT = 1\)

Sub by order \hspace{1cm} \textbf{subo} \hspace{1cm} \textbf{n}

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Format} & \textbf{Action} & \textbf{QH} & \textbf{QT} & \textbf{Binary} \\
\hline
\textbf{sub} \hspace{0.2cm} \textbf{n} & qtw0\rightarrow qhw0-(qh+n)w & 1 & 1 & \text{00110010} \\
\hline
\end{tabular}
\end{center}
This instruction operands to the operand queue pointed by the QT subtract \((qh+n)w\) from \(qhw0\)w.

Here \(QH = 1\) & \(QT = 1\)

### Description:

This instruction operands to the operand queue pointed by the QT subtract \((qh+n)w\) from \(qhw0\)w.

Here \(QH = 1\) & \(QT = 1\)

### Sub unsigned by order

#### subuo n

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>subuo</td>
<td>((qh+n)w) - (qhw0)w</td>
<td>1</td>
<td>1</td>
<td>00110101</td>
</tr>
</tbody>
</table>

### Description:

This instruction operands to the operand queue pointed by the QT subtract from \((qh+n)w\) from \(qhw0\)w.

Here \(QH = 1\) & \(QT = 1\)

### Multiply

#### mul n

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>mul</td>
<td>((qh+n)w) - (qhw0)w</td>
<td>1</td>
<td>1</td>
<td>00110101</td>
</tr>
</tbody>
</table>
### Description:
Multiply two operands to the operand queue pointed by the QT from \((\text{qhw0}*(\text{qh}+\text{n})\text{w})\).

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>(QH)</th>
<th>(QT)</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>mul n</td>
<td>qtw0←qhw0*(qh+n)w</td>
<td>1</td>
<td>1</td>
<td>00111011</td>
</tr>
</tbody>
</table>

### Multiply unsigned \(\text{mulu n}\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>(QH)</th>
<th>(QT)</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>mulu n</td>
<td>qtw0←qhw0+(qh+n)w</td>
<td>1</td>
<td>1</td>
<td>00111000</td>
</tr>
</tbody>
</table>

### Description:
Multiply two unsigned operands to the operand queue pointed by the QT from \((\text{qhw0}*(\text{qh}+\text{n})\text{w})\).

Here \(QH = 1\) & \(QT = 1\)

### Divide \(\text{div n}\)
Description:
Divide two operands to the operand queue pointed by the QT from (qhw0/(qh+n)w).
Here QH = 1 & QT = 1

Divide unsigned

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>divu n</td>
<td>qtw0←qhw0/(qh+n)w</td>
<td>1</td>
<td>1</td>
<td>00111110</td>
</tr>
</tbody>
</table>

Description:
Divide two unsigned operands to the operand queue pointed by the QT from (qhw0/(qh+n)w).
Here QH = 1 & QT = 1

Divide by order

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>divo n</td>
<td>qtw0←qhw0/(qh+n)w</td>
<td>1</td>
<td>1</td>
<td>00111110</td>
</tr>
</tbody>
</table>
Description:
This instruction to the operand queue pointed by the QT that divide \(qhw1\) by \(qhw0\) for consumed order instruction or \((qhw1 + n) / qhw0\).

Here \(QH = 1\) & \(QT = 1\)

Divide unsigned by order \(\text{divuo } n\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>divuo n</td>
<td>(qtw0 ← (qhw1 + n) / qhw0)</td>
<td>1</td>
<td>1</td>
<td>00111111</td>
</tr>
</tbody>
</table>

Description:
This instruction to the operand queue pointed by the QT that divide \((qhw1 + n)w\) by \(qhw0\).

Here \(QH = 1\) & \(QT = 1\)

Modular \(\text{mod } n\)
This instruction operands to the operand queue pointed that reminder of two operands to the QT from \((\text{qhw0}/(\text{qh}+\text{n}) \text{w})\).

Here \(\text{QH} = 1\) & \(\text{QT} = 1\)

**Modular by order**

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{mod n} & \text{n} \\
01000001 & \text{rem(qhw0/(qh+n)w)} & 1 & 1 & 01000001 \\
\hline
\end{array}
\]

**Description:**

This instruction operands to the operand queue pointed that reminder of two operands to the QT from \((\text{qhw0}/(\text{qh}+\text{n}) \text{w})\).

Here \(\text{QH} = 1\) & \(\text{QT} = 1\)

**Modular unsigned**

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{modo n} & \text{n} \\
01000011 & \text{rem((qh+n)w/qhw0)} & 1 & 1 & 01000011 \\
\hline
\end{array}
\]

**Description:**

This instruction operands to the operand queue pointed that reminder of two operands to the QT \(( (\text{qh}+\text{n})\text{w}/ \text{qhw0})\).

Here \(\text{QH} = 1\) & \(\text{QT} = 1\)
Description:
This instruction operates to the operand queue pointed that remainder of two unsigned operands to the QT from \( (qhw0/(qh+n)w) \).
Here \( QH = 1 \) & \( QT = 1 \)

**Modular unsigned by order**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{modu} p,n</td>
<td>rem(qhw0/(qh+n)w)</td>
<td>1</td>
<td>1</td>
<td>01000010</td>
</tr>
</tbody>
</table>

\[ \texttt{modu} p,n \quad \text{rem}(qhw0/(qh+n)w) \quad 1 \quad 1 \quad 01000010 \]

Description:
This instruction operates to the operand queue pointed that remainder of two unsigned operands to the QT from \( (qh+n)w/qhw0 \).
Here \( QH = 1 \) & \( QT = 1 \)

**moduo n**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{moduo} n</td>
<td>rem((qh+n)w/qhw0)</td>
<td>1</td>
<td>1</td>
<td>01000010</td>
</tr>
</tbody>
</table>

\[ \texttt{moduo} n \quad \text{rem}((qh+n)w/qhw0) \quad 1 \quad 1 \quad 01000010 \]

And

\[ \text{and } n \]
Description:
And two operands to the operand queue pointed by the QT from (qhw0 and (qh+n)w. Here QH = 2 & QT = 1

Or

Description:
Or two operands to the operand queue pointed by the QT from (qhw0 or (qh+n)w. Here QH = 1 & QT = 1

Negative
### Description:
This instruction to the operand queue pointed by the QT from \((qhw0 - (qh+n)w)\).
Here \(QH = 1\) & \(QT = 1\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>neg n</td>
<td>qtw0←-(qh+n)w</td>
<td>1</td>
<td>1</td>
<td>00110110</td>
</tr>
</tbody>
</table>

### Xor

<table>
<thead>
<tr>
<th>xor</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor n</td>
<td>qtw0←qhw0 xor (qh+n)w</td>
</tr>
</tbody>
</table>

### Description:
Xor two operands to the operand queue pointed by the QT from \((qhw0 xor (qh+n)w)\).
Here \(QH = 1\) & \(QT = 1\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor n</td>
<td>qtw0←qhw0 xor (qh+n)w</td>
<td>1</td>
<td>1</td>
<td>00111001</td>
</tr>
</tbody>
</table>

### Not

| not n |

| not n |
Description:
This instruction to the operand queue pointed by the QT from (qhw0←not((qh+n)w)).
Here QH = 1 & QT = 1

### Shift and rotate Instruction

**Shift Instruction Format:**

| 8 | 4 | 4 |

**Rotate Instruction Format:**

| 8 | 8 |

Right Shift  

sru s,n
**Description:**
This instruction to the operand queue pointed by the QT that logically right shift (qhw0←(qh+n)w).
Here QH = 1 & QT = 1

**Left Shift**

```
 15 7 3 0
  
  su 01001011 s n

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>sru p,s,n</td>
<td>qtw0←logical shift(qh+n)w</td>
<td>1</td>
<td>1</td>
<td>01001011</td>
</tr>
</tbody>
</table>
```

**Description:**
This instruction to the operand queue pointed by the QT that logically left shift from (qhw0←(qh+n)w).
Here QH = 1 & QT = 1

**Right Shift (Arithmetically)**

```
 15 7 3 0
  
  r 01001100 s n

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>sru p,s,n</td>
<td>qtw0←logical shift(qh+n)w</td>
<td>1</td>
<td>1</td>
<td>01001100</td>
</tr>
</tbody>
</table>
```
**Description:**
This instruction to the operand queue pointed by the QT that arithmetically from
$(qhw0 ← (qh+n)w$.
Here $QH = 1$ & $QT = 1$

**Rotate left** $\texttt{rol } n$

<table>
<thead>
<tr>
<th>$\texttt{sr}$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>01001101</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>$QH$</th>
<th>$QT$</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\texttt{sr } n$</td>
<td>$\texttt{qtw0 ← Arithmetic right shift(qh+n)w}$</td>
<td>1</td>
<td>1</td>
<td>01001101</td>
</tr>
</tbody>
</table>

**Description:**
This instruction to the operand queue pointed by the QT that rotates left.
Here $QH = 1$ & $QT = 1$

**Rotate Right** $\texttt{ror } n$

<table>
<thead>
<tr>
<th>$\texttt{rol}$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>01001111</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>$QH$</th>
<th>$QT$</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\texttt{rot } n$</td>
<td>$\texttt{Rotate left}$</td>
<td>1</td>
<td>1</td>
<td>01001111</td>
</tr>
</tbody>
</table>
### Description:
This instruction to the operand queue pointed by the QT that rotates right.
Here QH = 1 & QT = 1

### Duplicate
**dup n**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>dup n</td>
<td>qtw0,...,qtw(2c-1) ← q(qh+n)</td>
<td>1</td>
<td>1</td>
<td>00100111</td>
</tr>
</tbody>
</table>

### Description:
This instruction to the operand queue pointed by the QT that rotate & duplicate from (qhw0,...,qtw(2c-1) ← q(qh+n)).
Here QH = 1 & QT = 1

### Move
**mov n**
**Description:**
This instruction to the operand queue pointed by the QT that move from \((qh+n)\) for produced order instruction.
Here \(QH = 1\) & \(QT = 1\)
ALU for Double Word

<table>
<thead>
<tr>
<th>opcode</th>
<th>offset (n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Add double  

addd n
**Description:**
Add two double operands to the operand queue pointed by the QT from $(qhd0+(qh+n)d)$.
Here $QH = 2$ & $QT = 2$

Add double Unsigned\hspace{2cm} \text{adddu} \ n

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>$QH$</th>
<th>$QT$</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>adddu n</td>
<td>$qtd0\leftarrow qhd0+(qh+n)d$</td>
<td>2</td>
<td>2</td>
<td>11100001</td>
</tr>
</tbody>
</table>

**Description:**
Add two double unsigned operands to the operand queue pointed by the QT from $(qhd0+(qh+n)d)$.
Here $QH = 2$ & $QT = 2$

Sub double\hspace{2cm} \text{subd} \ n
Subtract two double operands to the operand queue pointed by the QT from (qhd0-(qh+n)d).
Here QH = 2 & QT = 2

**Description:**

Sub double unsigned  

subdu n

Sub double by order  

subdo n
**Description:**
This instruction operands to the operand queue pointed by the QT subtract from \((qh+n)d\) - qhd0).
Here QH = 2 & QT = 2

**Sub unsigned double by order**

```
subduo n
```

```
<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>subduo n</td>
<td>qtd0←(qh+n)d - qhd0</td>
<td>2</td>
<td>2</td>
<td>11100101</td>
</tr>
</tbody>
</table>
```

**Description:**
This instruction operands to the operand queue pointed by the QT subtract from \((qh+n)d\) - qhd0).
Here QH = 2 & QT = 2

**Multiply double**

```
muld n
```
Description:
Multiply two operands to the operand queue pointed by the QT from \((qhd0*(qh+n)d)\) for produced order instruction.
Here \(QH = 2\) & \(QT = 2\)

Multiply unsigned double \(\text{muldu } n\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>muldu n</td>
<td>qtd0(\leftarrow)qhd0((qh+n)d)</td>
<td>2</td>
<td>2</td>
<td>11100111</td>
</tr>
</tbody>
</table>

Description:
Multiply two unsigned double operands to the operand queue pointed by the QT from \((qhd0*(qh+n)d)\).
Here \(QH = 2\) & \(QT = 2\)

Divide double \(\text{divd } n\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>divd n</td>
<td>qtd0(\leftarrow)qhd0((qh+n)d)</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Description:
Divide two operands to the operand queue pointed by the QT from \((qhd0/(qh+n)d)\).

Here \(QH = 2\) & \(QT = 2\)

Divide double unsigned \(\text{divdu n}\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>(QH)</th>
<th>(QT)</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>divdu n</td>
<td>qtd0←qhd0/(qh+n)d</td>
<td>2</td>
<td>2</td>
<td>11100000</td>
</tr>
</tbody>
</table>

Description:
Divide two unsigned double operands to the operand queue pointed by the QT from \((qhd0/qhd1)\) for consumed order instruction or \((qhd0/(qh+n)d)\) for produced order instruction.
Here \(QH = 2\) & \(QT = 2\)

Divide double by order \(\text{divdo n}\)


**Description:**  
This instruction to the operand queue pointed by the QT that divide \((qh+n)d\) by \(qhd0\).

Here \(QH = 2\) & \(QT = 2\)

**Divide unsigned double by order**  
\[ \text{divduo } n \]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Format} & \text{Action} & QH & QT & \text{Binary} \\
\hline
\text{divduo } n & \text{qtd0} \leftarrow (qh+n)d/qhd0 & 2 & 2 & 11101011 \\
\hline
\end{array}
\]

**Description:**  
This instruction to the operand queue pointed by the QT that divide \((qh+n)d\) by \(qhd0\).

Here \(QH = 2\) & \(QT = 2\)

**Modular double**  
\[ \text{modd } n \]
**Description:**
This instruction operates on the operand queue pointed that the remainder of two operands to the QT from \( (qh+n)/qhd0 \). 
Here \( QH = 2 \) & \( QT = 2 \)

**Modular double by order**  
\[ \text{moddo} \ n \]

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>( QH )</th>
<th>( QT )</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{moddo} n</td>
<td>rem((qh+n)/qhd0)</td>
<td>2</td>
<td>2</td>
<td>11101110</td>
</tr>
</tbody>
</table>

**Description:**
This instruction operates on the operand queue pointed that the remainder of two operands to the QT from \( (qh+n)/qhd0 \). 
Here \( QH = 4 \) & \( QT = 2 \)

**Modular double unsigned**  
\[ \text{moddu} \ n \]
**Description:**
This instruction operands to the operand queue pointed that reminder of two unsigned operands to the QT from \((qhd0/(qh+n)d)\).
Here \(QH = 2\) & \(QT = 2\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>moddu n</td>
<td>rem(qhd0/(qh+n)d)</td>
<td>2</td>
<td>2</td>
<td>11101101</td>
</tr>
</tbody>
</table>

**Modular unsigned double by order modduo n**

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>modduo n</td>
<td>rem((qh+n)d/qhd0)</td>
<td>2</td>
<td>2</td>
<td>11101111</td>
</tr>
</tbody>
</table>

**Description:**
This instruction operands to the operand queue pointed that reminder of two unsigned operands to the QT from \((qh+n)d/qhd0)\).
Here \(QH = 4\) & \(QT = 2\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>modduo n</td>
<td>rem((qh+n)d/qhd0)</td>
<td>2</td>
<td>2</td>
<td>11101111</td>
</tr>
</tbody>
</table>

**And double andd n**
And two double operands to the operand queue pointed by the QT from \((q_h+n)d\).
Here \(Q_H = 2\) & \(Q_T = 2\)

\[
\text{Or double} \quad \text{ord} \ n
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Format} & \text{Action} & \text{QH} & \text{QT} & \text{Binary} \\
\hline
\text{ord} \ n & qtd0←qhd0 or (qh+n)d & 2 & 2 & 11110001 \\
\hline
\end{array}
\]

\text{Description:}
Or two double operands to the operand queue pointed by the QT from \((qh+n)d\).
Here \(Q_H = 2\) & \(Q_T = 2\)

\[
\text{Negative double} \quad \text{negd} \ n
\]
Description:
This instruction to the operand queue pointed by the QT from \((qhd_0 ← -(qh+n)d)\).

Here \(QH = 2 \& QT = 2\)

Xor double \(xord\ n\)

Description:
Xor two double operands to the operand queue pointed by the QT from \((qhw_0 \ xor \ (qh+n)w)\).
Here \(QH = 2 \& QT = 2\)

Not double \(notd\ n\)
**Description:**
This instruction to the operand queue pointed by the QT from \((\text{qhw}0 \leftarrow \text{not}((\text{qh}+n)\text{w}))\).
Here \(\text{QH} = 2\) & \(\text{QT} = 2\)

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>notd n</td>
<td>qtd0←not((qh+n)d )</td>
<td>2</td>
<td>2</td>
<td>10110001</td>
</tr>
</tbody>
</table>
Shift Instruction

Instruction Format:

\[
\begin{array}{ccc}
8 & 4 & 4 \\
\end{array}
\]

Right Shift for double \texttt{srdu s,n}
Description:
This instruction to the operand queue pointed by the QT that logically right shift from \((qhd_0 \leftarrow (qh+n)d)\).
Here \(QH = 2\) & \(QT = 2\)

**Right Shift (Arithmetically) for double** \(srd\ s,n\)

\[
\begin{array}{cccc}
\text{Format} & \text{Action} & QH & QT \\
\text{srd s,n} & qtd_0 \leftarrow \text{logical right shift}(qh+n)d & 2 & 2 \\
\end{array}
\]

Description:
This instruction to the operand queue pointed by the QT that arithmetically from \((qhd_0 \leftarrow (qh+n)d)\).
Here \(QH = 2\) & \(QT = 2\)

**Rotate left for double** \(rold\ n\)
Description:
This instruction to the operand queue pointed by the QT that rotate left.
Here QH = 2 & QT = 2

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>rold n</td>
<td>Rotate left</td>
<td>2</td>
<td>2</td>
<td>10110101</td>
</tr>
</tbody>
</table>

Rotate Right for double  
**rord n**

Description:
This instruction to the operand queue pointed by the QT that rotate right.
Here QH = 2 & QT = 2

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>QH</th>
<th>QT</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>rord n</td>
<td>Rotate right</td>
<td>2</td>
<td>2</td>
<td>10111010</td>
</tr>
</tbody>
</table>

Rotate & duplicate for double  
**dupd n**
Here $QH = 2$ & $QT = 2$.

This instruction to the operand queue pointed by the QT that rotate & duplicate from $(qhd_0,...,qtd_{2c-1}) \rightarrow (q_{qhd+n})$.

<table>
<thead>
<tr>
<th>Format</th>
<th>Action</th>
<th>Binary</th>
<th>QD</th>
<th>QH</th>
<th>dupd n</th>
<th>qtd_0,..,qtd_{2c-1}...(q_{qhd+n})</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>dup</td>
<td>10011010</td>
<td>7</td>
<td>7</td>
<td>16</td>
<td>10011010</td>
</tr>
</tbody>
</table>