Chapter 7. Concurrent Processors

Problem 7.1

VADD VR3, VR1, VR2
VMPY VR5, VR3, VR4
Vector size = 64

Unchained time = 8 (add startup) + 64 (elements/VR) + 8 (mpy startup) + 64 (elements)
= 144 cycles.
Chained time = 8 (add startup) + 8 (mpy startup) + 64 (elements/VR) = 80 cycles.

a. Implied (average) instruction memory bandwidth

\[
\frac{2 \text{ inst} \times 4 \text{ bytes/inst}}{144 \text{ cycles}} = 0.0556 B/\text{cycle} \text{ for unchained}
\]

and

\[
\frac{2 \text{ inst} \times 4 \text{ bytes/inst}}{80 \text{ cycles}} = 0.1 B/\text{cycle} \text{ for chained}
\]

VLD VR1, add1
VLD VR2, add2
VLD VR4, add3
b. VADD VR3, VR1, VR2
VMPY VR5, VR3, VR4
VST VR5, add4

Unchained time = (8 + 8 + 8) (load startup) + 64 (load completion) + 8 (add startup)
+64 (add completion) + 8 (multiply startup) + 64 (multiply completion)
+8 (store startup) + 64 (store completion)
= 6 \times 8 + 4 \times 64 = 304 \text{ cycles}

A total of 64 registers \times 4 \text{ instructions} \times 8^3 = 2048 \text{ bytes} \text{ are moved.}

Implied (average) data bandwidth = \frac{2048 \text{ bytes}}{304 \text{ cycles}} = 6.7 B/\text{cycles}

Chained time = (8 + 8 + 8) (load startup) + 64 (load completion) + 8 (add startup)
+8 (multiply startup) + 64 (multiply completion) + 8 (store startup)
+64 (store completion)
= 6 \times 8 + 3 \times 64 = 240 \text{ cycles}

Implied (average) data bandwidth = \frac{2048 \text{ bytes}}{340 \text{ cycles}} = 5.95 B/\text{cycles}

Problem 7.2

a. \( F37B90_{16} = 330313232100_{4} \)
\[ r = (0 - 0 + 1 - 2 + 3 - 2 + 3 - 1 + 3 - 0 + 3 - 3) \mod 5 = 5 \mod 5 = 0 \]
So, module address = 0

\[
\begin{array}{c}
330313231004 \\
- 600230211004 \\
0300230211004 = 30B250_{16}
\end{array}
\]

So, address in module = B250_{16}

b. \( A43347_{16} = 222023031013_{4} \)
\[ r = (3 - 1 + 0 - 1 + 3 - 0 + 3 - 0 + 2 - 2 + 2 - 2) \mod 5 = 7 \mod 5 = 2 \]
So, module address = 0

\[
\begin{array}{c}
22203031013_{4} \\
- 202002210012_{4} \\
020200221001_{4} = 220A41_{16}
\end{array}
\]

So, address in module = 220A41_{16}

**Problem 7.3**

<table>
<thead>
<tr>
<th>( m_3 )</th>
<th>( m_2 )</th>
<th>( m_1 )</th>
<th>( m_0 )</th>
<th>( m_3' )</th>
<th>( m_2' )</th>
<th>( m_1' )</th>
<th>( m_0' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Since each original address maps to a unique hashed address, the scheme works.

b. \( F37B90_{16} \)
The 4 least significant bits are 0000\(_2\) which map to 0010\(_2\), thus the module address is 2.

\( A43347_{16} \)
The 4 least significant bits are 0111\(_2\) which map to 1001\(_2\), thus the module address is 9.

**Problem 7.4**

\( \Delta T = 8\text{ns} \)

Memory cycle time \( (T_c) = 64\text{ns} \)

Two requests per \( \Delta T \)
a. Offered memory bandwidth
   \[ n = 2 \times \frac{64}{8} = 16 \]
   \[ \lambda = \frac{n}{T_c} = \frac{16}{64 \times 10^{-9}} = 250 \text{ MAPS} \]
   Assume 64 bit word size
   \[ \lambda = 250 \text{ MAPS} \times 8 \text{ bytes} = 1907 \text{ MBps} \]
b. Achieved bandwidth
   Use \( M_B/D/1 \)
   \[ B(m, n) = 8 + 16 - \frac{1}{2} - \sqrt{(8 + 16 - .5)^2 - 2 \times 8 \times 16} = 6.288 \]
   \[ \lambda_n = \frac{B}{n} \times \lambda_{offered} \]
   \[ = \frac{6.288}{16} \times 1907 \text{ MBps} \]
   \[ = 750 \text{ MBps} \]
c. Mean queue size
   \[ Q_{c-t} = n - B = 16 - 6.288 = 9.712 \]

**Problem 7.8**

\( \gamma = .5, m = 17 \)
\( n = 16 \) and \( T_c = 64 \text{ ns} \), from problem 7.4
Assume 64 bit word size.
\[ B(m, n, \gamma) = 17 + 16(1 + .5) - .5 - \sqrt{(17 + 16(1 + .5) - .5)^2 - 2 \times 16 \times 17(1.5)} = 11.79 \]
\[ \lambda_{sch} = \frac{B}{T_c} = \frac{11.79}{64 \times 10^{-9}} \approx 184 \text{ MAPS} = 1405 \text{ MBps} \]

**Problem 7.10**

4 memory requests/\( \Delta T \)
\( \Delta T = 8 \text{ ns} \)
\( T_c = 42 \text{ ns} \)

a. Minimum number of interleaving for no contention
   \[ m > n \]
   \[ m > 4 \times \frac{42}{8} = 21, \text{ or } m = 32 \text{ if only powers of 2 are allowed.} \]
b. \( m = 64 \)
   \[ \gamma_{opt} = \frac{n - 1}{2m - 2n} = \frac{21 - 1}{2 \times 64 - 2 \times 21} = .233 \]
   Mean TBF \[ = n \times \gamma_{opt} = 21 \times .233 = 4.893 \]
   \[ B(m, n, \gamma) = B(64, 21, .233) = 20.567 \]
   Relative perf \[ = \frac{20.567}{21} = .979 \]
**Problem 7.11**

Assume that both the vector processor and the uniprocessor have the same cycle time. Assume that the uniprocessor can execute one instruction every cycle. Assume that the vector processor can do chaining.

The vector processor can load 3 operands concurrently in advance of using them since there are 3 read ports to memory. It can also store a result concurrently since there is a write port to memory. With chaining, the vector processor can perform 2 arithmetic operations concurrently if there are enough functional units.

Thus, the vector processor can perform 6 operations per cycle and the maximum speedup over a uniprocessor is 6.

**Problem 7.14**

\[
\begin{align*}
I_1 & \text{ DIV.F R1,R2,R3} \\
I_2 & \text{ MPY.F R1,R4,R5} \\
I_3 & \text{ ADD.F R4,R5,R6} \\
I_4 & \text{ ADD.F R5,R4,R7} \\
I_5 & \text{ ST.F ALPHA,R5}
\end{align*}
\]

a. Improved control flow.

Assume there is no reorder buffer.

Assume floating point divide, multiply, and add take 8,4, and 3 separately.

Cycle 1: Decoder Issues $I_1 \rightarrow$DIV unit

$R2,R3 \rightarrow$DIV Res Stn

TAG_DIV $\rightarrow R1$

Cycle 2: Divide begins $DIV.F$

Cycle 9: Divide completes

Divide requests permission to broadcast result in the next cycle

Cycle 10: Divide Result $\rightarrow R1$

Cycle 11: Decoder Issues $I_2 \rightarrow$MPY unit

$R4,R5 \rightarrow$MPY unit

TAG_MPY $\rightarrow R1$

Cycle 12: MPY begins

Decoder issues $I_3 \rightarrow$ADD unit

$R5,R6 \rightarrow$ADD unit

ADD_TAG1 $\rightarrow R4$

Cycle 13: ADD begins

Decoder issues $I_4 \rightarrow$ADD unit

ADD_TAG1 $\rightarrow$ADD unit

$R7 \rightarrow$ADD unit

ADD_TAG2 $\rightarrow R5$

Cycle 14: ADD2 waits

Decoder issues $I_5 \rightarrow$store unit

ADD_TAG2 $\rightarrow$store units

Cycle 15: $I_2,I_3$ completes, requests for broadcast

($I_3$ is granted, $I_3$ is not granted)

Cycle 16: MPY $\rightarrow R1$

Cycle 17: ADD $\rightarrow R4$, ADDER2's Res Stn
Cycle 18: ADD \(I_4\) begins
Cycle 20: \(I_4\) completes and requests for broadcast (granted)
Cycle 21: ADD \(\rightarrow R5\), store units
Cycle 22: \(I_5\) begins

b. Data flow with value holding reservation station.

Assume reorder buffer.

<table>
<thead>
<tr>
<th>Cycle 1:</th>
<th>Decoder issues</th>
<th>(I_1 \rightarrow \text{DIV unit})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(R3,R2 \rightarrow \text{DIV Res Stn})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{TAG_DIV} \rightarrow R1)</td>
</tr>
<tr>
<td>Cycle 2:</td>
<td>\text{Divides begins}</td>
<td>(\text{DIV_F})</td>
</tr>
<tr>
<td></td>
<td>Decoder issues</td>
<td>(I_2 \rightarrow \text{MPY unit})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(R4,R5 \rightarrow \text{MPY Res Stn})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{TAG_MPY} \rightarrow R1)</td>
</tr>
<tr>
<td>Cycle 3:</td>
<td>\text{Begin}</td>
<td>(\text{MPY_F})</td>
</tr>
<tr>
<td></td>
<td>Decoder issues</td>
<td>(I_3 \rightarrow \text{ADD unit})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(R5,R6 \rightarrow \text{ADD Res Stn})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{TAG_ADD1} \rightarrow R4)</td>
</tr>
<tr>
<td>Cycle 4:</td>
<td>\text{Begin}</td>
<td>(\text{ADD_F})</td>
</tr>
<tr>
<td></td>
<td>Decoder issues</td>
<td>(I_4 \rightarrow \text{ADD unit})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{TAG_ADD1} \rightarrow \text{ADD Res Stn})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(R7 \rightarrow \text{ADD Res Stn})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{TAG_ADD2} \rightarrow R5)</td>
</tr>
<tr>
<td>Cycle 5:</td>
<td>ADDER for (I_4) waits</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decoder issues</td>
<td>(I_5 \rightarrow \text{Store unit})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{TAG_ADD2} \rightarrow \text{store buffer})</td>
</tr>
<tr>
<td>Cycle 6:</td>
<td>MPY completes and requests for broadcast (granted)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD completes and requests for broadcast (not granted)</td>
<td></td>
</tr>
<tr>
<td>Cycle 7:</td>
<td>Multiply unit (\rightarrow R1)</td>
<td></td>
</tr>
<tr>
<td>Cycle 8:</td>
<td>ADD unit (1st unit) (\rightarrow R4), 2nd Res Stn of Adder</td>
<td></td>
</tr>
<tr>
<td>Cycle 9:</td>
<td>ADD (I_4) begins</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{DIV completes and requests for broadcast (granted)})</td>
<td></td>
</tr>
<tr>
<td>Cycle 10:</td>
<td>DIV_F broadcasts but ignored</td>
<td></td>
</tr>
<tr>
<td>Cycle 11:</td>
<td>ADD completes and request for broadcast (granted)</td>
<td></td>
</tr>
<tr>
<td>Cycle 12:</td>
<td>ADD unit (\rightarrow R5), store buffer</td>
<td></td>
</tr>
<tr>
<td>Cycle 13:</td>
<td>STORE begins</td>
<td></td>
</tr>
</tbody>
</table>
c. Control flow with shadow register.

Assume reorder buffer.

In this case, it would be better to rewrite the code using register renaming.

DIV, F R1, R2, R3 renamed
MPY, F R9, R4, R5 R1 → R9
ADD, F R10, R5, R6 R4 → R10
ADD, F R11, R10, R7 R5 → R11
ST, F ALPHA, R11 R5 → R11

Cycle 1: Decoder issues $l_1 \rightarrow$ DIV unit
   R3, R2 → DIV Res Stn
   TAG_DIV → R1
Cycle 2: Divides begins DIV, F
   Decoder issues $l_2 \rightarrow$ MPY unit
   R4, R5 → MPY Res Stn
   TAG_MPY → R9
Cycle 3: Begin MPY, F
   Decoder issues $l_3 \rightarrow$ ADD unit
   R5, R6 → ADD Res Stn
   TAG_ADD1 → R10
Cycle 4: Begin ADD, F
   Decoder issues $l_4 \rightarrow$ ADD unit
   TAG_ADD1 → ADD Res Stn
   R7 → ADD Res Stn
   TAG_ADD2 → R11
Cycle 5: ADDER for $l_4$ waits
   Decoder issues $l_5 \rightarrow$ Store unit
   TAG_ADD2 → store buffer
Cycle 6: MPY completes and requests for broadcast (granted)
   ADD completes and requests for broadcast (not granted)
Cycle 7: Multiply unit → R9
Cycle 8: ADD unit (1st unit) → R10, 2nd Res Stn of Adder
Cycle 9: ADD ($l_4$) begins
   DIV completes and requests for broadcast (granted)
Cycle 10: DIV, F → R1
Cycle 11: ADD completes and request for broadcast (granted)
Cycle 12: ADD unit → R11, store buffer
Cycle 13: STORE begins