1. Systolic arrays

a) Create a fully-pipelined datapath for the following code. Do not perform any optimizations (loop unrolling, etc.).

```c
short a[100], b[101];
for (i=0; i < 100; i++)
  a[i] = 3*b[i] + 5*b[i+1] + 11*b[i+2] + 4*b[i+3];
```

b) Calculate speedup of the fully-pipelined 300 MHz circuit compared to software execution on a microprocessor (assuming 20 instructions for each iteration, a CPI of 1.5, and a clock frequency of 3 GHz). Assume memory bandwidth is sufficient for full pipelining.

Latency = 5 cycles, iteration 0 takes 5 cycles, iteration 1-99 takes 1 cycle.

Total HW cycles = 5 + 99 × 1 = 104 cycles.

Total SW cycles = total instructions × CPI = (20 × 100) × 1.5 = 3000 cycles.

Speedup = \( \frac{SW\, time}{HW\, time} = \frac{3000}{104} \times \frac{300\, MHz}{3000\, MHz} = 2.8 \)
c) Assuming that memory can deliver 128 bits per cycle, how many iterations of the loop can be performed in parallel? (Ignore any creative buffering, i.e. assume that data from previous memory fetches is not reused)

First iteration requires $16 \times 4 = 64$ bits, each unrolled iteration requires 16 bits.

Bits available for unrolling = 128 - 64 = 64 bits

Amount of possible unrolling = $\frac{64}{16} = 4$

Total parallel iterations = $4 + 1 = 5$

d) Calculate the speedup of the circuit when utilizing the amount of loop unrolling determined in part c.

Total HW cycles = $5 + \frac{95}{5} = 24$ cycles

Speedup = $\frac{3000}{24} \times \frac{300}{3000} = \underline{12.5}$

e) Draw a block diagram of the entire circuit (not just the datapath).

2. Create a fully-pipelined datapath for the following code. Do not unroll the loop.

```c
short a[100], b[100];
for (i=0; i < 100; i++) {
    if (b[i] < 0)
        a[i] = b[i] + 1;
    else
        a[i] = b[i] - 1;
}
```

![Diagram of the fully-pipelined datapath]
3. Create a fully-pipelined datapath for the following code. Do not perform any optimizations. Explain why without any optimizations, the pipeline may frequently stall. Be specific.

```c
short a[100], b[100], c[100], d[100], e[100];
for (i=0; i < 100; i++) {
    a[i] = b[i]+c[i]+d[i]+e[i];
}
```

![Diagram](image)

The pipeline would likely stall because the 4 arrays may be stored in different parts of memory.

4. Create a fully-pipelined datapath for the following code. List any optimizations needed to eliminate loop-carried dependencies. Show that such optimizations enable a fully pipelined datapath, assuming memory delivers 128 bits of data every cycle.

```c
short a[100], b[104];
for (i=0; i < 100; i++) {
    for (j=i; j < i+4; j++) {
        val += b[j];
    }
    a[i] = val;
}
```

1) unroll inner loop

```
val += b[i];
val += b[i+1];
val += b[i+2];
val += b[i+3];
```

2) tree-height reduction

```
val = val + b[i] + b[i+4] + b[i+8] + b[i+12] + b[i+16];
```

3) constant propagation

```
val = b[i] + b[i+4] + b[i+8] + b[i+12] + b[i+16];
```

For unrolling to not cause stalls, the circuit requires 4 inputs each cycle.

```
required bandwidth = 4 x 16 = 64 bits/cycle
actual bandwidth = 128 bits/cycle
```

Unrolling inner loop does not cause stalls.
5. For the following code:

```c
int f( short a[100] ) {
    int x,y,z;
    x=0; y=0; z=50;
    for (i=0; i < 100; i++) {
        if (a[i] < z) {
            x += a[i];
        } else {
            y += a[i];
        }
    }
    return x*y;
}
```

(a) Manually create a FSM representing the controller, using the methodology discussed in the lecture slides. (Do not create a systolic array) *Ignore details of memory accesses*

(b) Manually create a datapath that works with the controller from part a. Show all inputs/outputs, and control signals. *Ignore details of memory accesses*
(c) Calculate the execution time in terms of cycles. List assumptions:

- Assume 1 cycle per stage.
- 1 iteration requires 4 cycles, 100 iterations = 400 cycles for the loop.

Total cycles = 400 + 1 + 1 = 402 cycles

(d) Explain why this circuit is likely much slower than a systolic array-based circuit, assuming a systolic array is possible.

- Not pipelined, each iteration takes 4 cycles.
- Systolic array can do 1 or more iterations per cycle.

(c) Explain the difficulty in creating a systolic array-based circuit for this code.
6. Optimize the following code, by showing the resulting code after each applied technique.

\[
\begin{align*}
  &x = 0; \\
  &y = a + b; \\
  &\text{if } (x < 15) \\
  &\quad z = a + b - c; \\
  &\text{else} \\
  &\quad z = x + 12; \\
  &o = z * 12;
\end{align*}
\]

1) const. prop. \( \Rightarrow x = 0; \quad y = a + b; \)

\[
\begin{align*}
  &\text{if } (0 < 15) \\
  &\quad z = a + b - c; \\
  &\text{else} \\
  &\quad z = 12; \\
  &o = z * 12;
\end{align*}
\]

2) dead code el. m. \( x = 0; \quad y = a + b; \)

\[
\begin{align*}
  &x = 0; \\
  &y = a + b; \\
  &z = y - c; \\
  &o = z * 12;
\end{align*}
\]

3) common subexpression el. m. \( x = 0; \quad y = a + b; \)

\[
\begin{align*}
  &x = 0; \\
  &y = a + b; \\
  &z = y - c; \\
  &o = z * 12;
\end{align*}
\]

4) strength reduction \( x = 0; \quad y = a + b; \)

\[
\begin{align*}
  &x = 0; \\
  &y = a + b; \\
  &z = y - c; \\
  &o = (z < 3) + (z < 2);
\end{align*}
\]
7. Schedule the following circuit: (using minimum latency scheduling)

(a) Using ASAP scheduling

(b) Using ALAP scheduling
c) What is the minimum number of resources required for the ASAP schedule? Assuming $\div$ for ALU, $\times$ for Mult

$3$ ALUs, $2$ Mult

d) What is the minimum number of resources required for the ALAP schedule?

$2$ ALUs, $1$ Mult

e) How would the resource requirements change for a fully-pipelined implementation?

$5$ ALUs, $3$ Mult
8. Trace the steps of *minimum-latency resource-constrained* list scheduling on the following DFG. Draw lightly next to each node.

Resource Constraints: 2 ALU(+/-), 1 mult

<table>
<thead>
<tr>
<th>Cycle</th>
<th>ALUs</th>
<th>Mults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1</td>
<td>2, 4, 7</td>
<td>1</td>
</tr>
<tr>
<td>Cycle 2</td>
<td>6</td>
<td>3, 6</td>
</tr>
<tr>
<td>Cycle 3</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Cycle 4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Cycle 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9. Trace the steps of *minimum-latency resource-constrained* list scheduling on the following DFG, assuming that multiplications take 2 cycles and divides take 5 cycles. Place priority next to each node.

Resource Constraints: 2 ALU(+/-), 2 mults, 1 div

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALUs</td>
</tr>
<tr>
<td>Cycle 1</td>
<td>x, 2, 4</td>
</tr>
<tr>
<td>Cycle 2</td>
<td>x</td>
</tr>
<tr>
<td>Cycle 3</td>
<td></td>
</tr>
<tr>
<td>Cycle 4</td>
<td></td>
</tr>
<tr>
<td>Cycle 5</td>
<td></td>
</tr>
<tr>
<td>Cycle 6</td>
<td></td>
</tr>
<tr>
<td>Cycle 7</td>
<td></td>
</tr>
<tr>
<td>Cycle 8</td>
<td></td>
</tr>
<tr>
<td>Cycle 9</td>
<td></td>
</tr>
<tr>
<td>Cycle 10</td>
<td></td>
</tr>
</tbody>
</table>
10. Trace the steps of minimum-resource latency-constrained list scheduling on the following DFG. Show the last possible cycle next to each node in the original graph. Show the slack for each candidate in parentheses. Show the resource requirements of each step, and the final requirements. Latency constraint = 4 cycles

<table>
<thead>
<tr>
<th>Cycle</th>
<th>ALUs</th>
<th>Mults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle1</td>
<td>1</td>
<td>2(3)</td>
</tr>
<tr>
<td>Cycle2</td>
<td>2(3)</td>
<td>4(3)</td>
</tr>
<tr>
<td>Cycle3</td>
<td>5(3)</td>
<td>6(3)</td>
</tr>
<tr>
<td>Cycle4</td>
<td>8</td>
<td>7(3)</td>
</tr>
<tr>
<td>Cycle5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Candidates

<table>
<thead>
<tr>
<th>ALUs</th>
<th>Mults</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a(0), b(1), c(3))</td>
<td>(x(0))</td>
</tr>
<tr>
<td>(y(0), z(1))</td>
<td>(b(0))</td>
</tr>
<tr>
<td>(b(0), y(1))</td>
<td>(b(0))</td>
</tr>
<tr>
<td>(b(0), y(1))</td>
<td>(b(0))</td>
</tr>
</tbody>
</table>

Required resources

2 ALUs, 1 Mult
11. Bind the following scheduled DFG using clique partitioning. Show the compatibility graph and cliques. Assume that the circuit uses 2 multipliers and 2 ALUs.
12. Bind the following scheduled DFG using the left edge algorithm. Assume that the circuit uses 2 multipliers and 2 ALUs. Work top to bottom if multiple candidates.

---

\[ ALU_1 = 3, 7, 8 \]
\[ ALU_2 = 2 \]
\[ Mul1 = 1, 4 \]
\[ Mul2 = 6, 5 \]
13. Translate the binding from the previous problem into a datapath.