for(i = 0; i < n; i++) {
    X[i+1] = X[i] * 2
    Y[i+1] = X[i+1] + Y[i]
}

1: LD R2 *(R0++)
2: MUL R3, R2, $2
3: ST *(R0), R3
4: LD R4, *(R0)
5: LD R5, *(R1++)
6: ADD R6, R4, R5
7: ST *(R1) R6

For this problem, assume that ADD/MUL takes 2 clocks, while
LD/ST takes 1 clock. The machine has two MEM units that can
execute a LD and ST, and two ALU units. The machine can
auto-increment address registers.

Given the above instructions corresponding to the snippet of
code, draw the data dependence graph.
Software Pipelining with Resource Constraints
2. Global instruction scheduling

Assume you have a statically scheduled machine that can only issue one operation every clock. All operations have a latency of one clock cycle, with the exception of its memory load operation, which has a latency of three clock cycles. Consider the following locally scheduled program:

```
L0
w = x;
h = y;
m = w + h;
if (m >= 0) go to L1;
```

```
L1
f = m + 2;
g = *f;
nop;
nop;
h = g + g;
```

```
L2
if (m == 0) go to L3;
```

```
L3
w = *h;
nop;
nop;
h = 2 * w + 1;
```

```
L4
r = h + w;
```

Assume that only r is live at the end of the program. Each branch in the flow graph is labeled with the probability that it is taken dynamically. To answer the following, you may apply any of the code motions discussed in class, but no other optimizations.

1. Is this the best globally scheduled code that can be generated given that p = 0.1? If not, provide the improved code along with its expected execution time.

L0 -> L2 => 0.1
L0 -> L1 => 0.9
2. Global instruction scheduling

Assume you have a statically scheduled machine that can only issue one operation every clock. All operations have a latency of one clock cycle, with the exception of its memory load operation, which has a latency of three clock cycles. Consider the following locally scheduled program:

PATHS:

L0 -> L2 -> L3 -> L4 => 0.01
L0 -> L2 -> L4 => 0.09
L0 -> L1 -> L4 => 0.9

Expected number of cycles:

L0 => 4 cycles
L1 => 5 cycles
L2 => 1 cycle
L3 => 4 cycles
L4 => 1 cycle

Expected number of cycles for each path, multiply by the probability:

4 + 1 + 4 + 1 = 10 => * 0.01 = 0.1 cycle (average)

Final result = 9.64 cycles.
L0:
W = x;
H = y;
M = w + h;

F = m+2;
G = *f
If (m >= 0) goto L1;

L1:
Nop;
H = g + g;

L0 -> L2 -> L3 -> L4 => 0.01
L0 -> L2 -> L4 => 0.09
L0 -> L1 -> L4 => 0.9
L0 = 6
L1 = 2

Total result = 8.94 cycles < 9.64 cycles.
2 Dependency Analysis

```
for(i = 1; i < n; i++) {
    for(j = i; j < 2 * i; j++) {
    }
}
```

Draw the iteration space for this loop.
i = 1 .. n-1
j = 1 .. 2*(n-1)-1 = 1 .. 2n - 3
What are all the candidates for data dependencies in this loop? A data dependency is something of the form “read/write A[i], read/write B[j]”.

WAW
WAR
RAW

Formulate the data dependence tests for the given loop nest.

Consider the following program.

L1:
for (int i = 1; i < n; i++) {
    for (int j = 0; j < n; j++) {
        A[i, j] = c * A[i, j-1];
    }
}

L2:
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
    }
}

Assume A and B are two non-overlapping n × n matrices. Both matrices are stored in row-major layout.
1. Draw the iteration space for the program. Use arrows to mark data-dependencies between iterations.

L1:
for (int i = 1; i < n; i++) {
    for (int j = 0; j < n; j++) {
        A[i, j] = c * A[i, j-1];
    }
}

2. **Parallelize** each loop nest individually. Show the transformed code, and describe any transformation you performed.

```c
#pragma omp parallel for
for (int p = 1; p < n; i++) {
    for (int j = 0; j < n; j++) {
    }
}
```