Q1.

Yes. Availablility is a must relation while reaching is a may relation, thus any definition that is available will always be reaching.

Q2.

False. Not if it is conditionally executed and might cause an exception. Or, if the instruction conditionally over writes a live out, then LICM may also not be possible.

Q3.

Yes. The Lstart time represents the latest an instruction can be scheduled and still obtain the "best" or infinite resource schedule length. But, schedules are often not the best due to resource constraints, thus instructions will be scheduled after their Lstart time.

Q4.

For loop count N

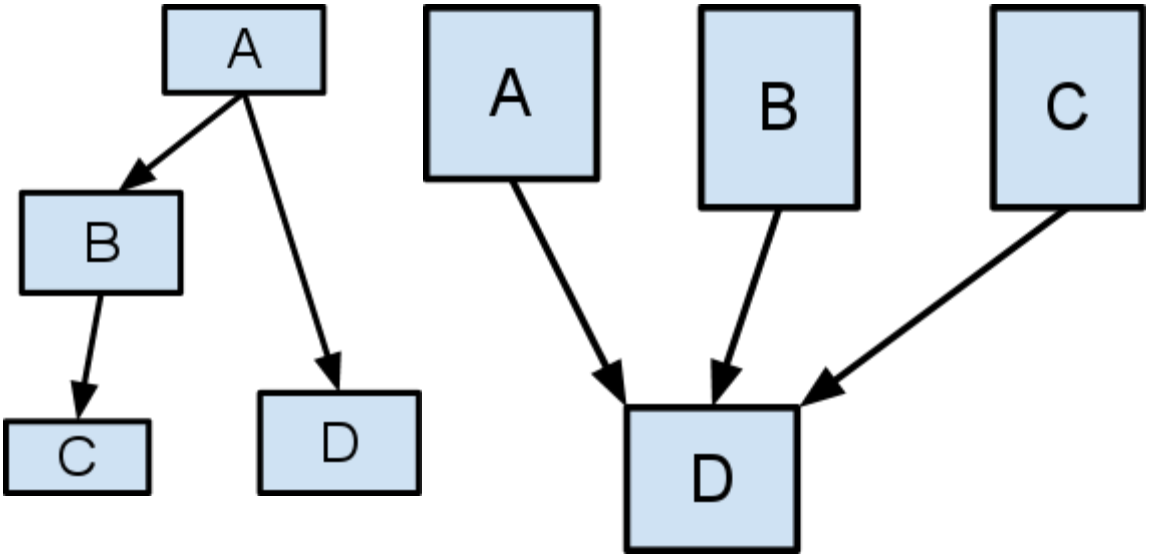
total cycles = N*II + (SC-1)*II

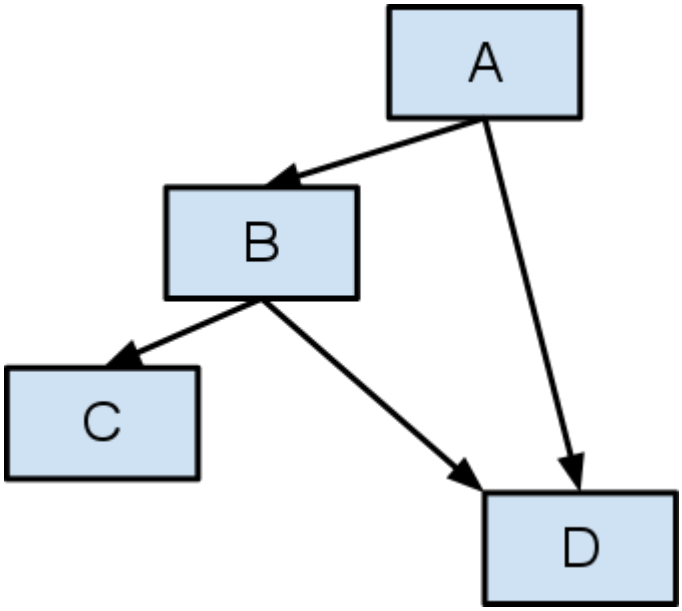
For large N, loops with smaller II will finish in fewer cycles, thus (a) will require fewer cycles.

Q5.

Memory instructions, particularly loads, because these take longer time and initiating these earlier enables more compact schedules to be achieved.

Q6:



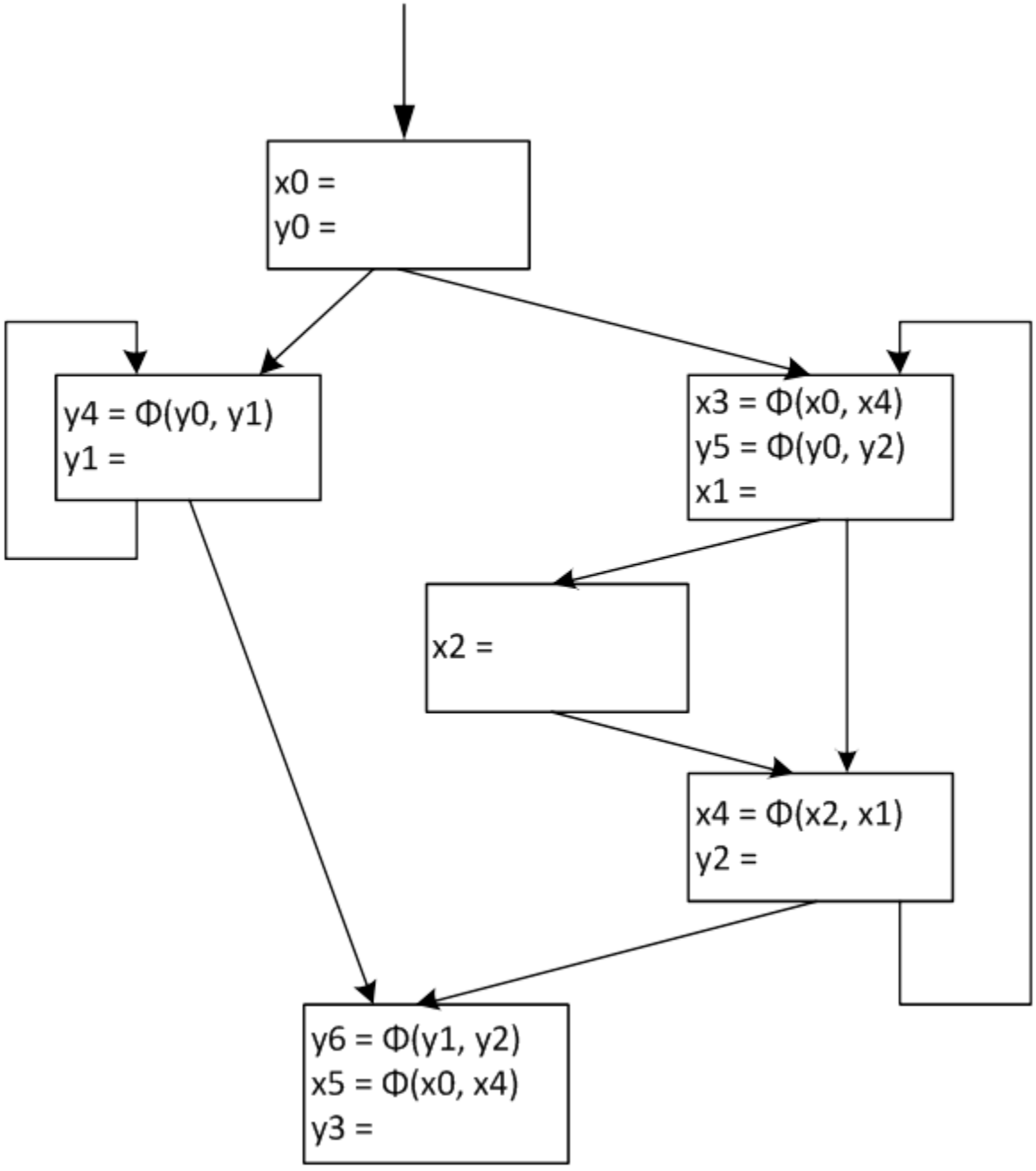


Q7:

K = {-1, +1, -3, -4, {+3, +4}} → 5 unique control dependences so 5 predicates are required

p1 = cmpp.UN(cond1_bar) if T
p2 = cmpp.UN(cond_3) if p1
p3 = cmpp.UN(cond_4) if p2

Q.8



Q.9

Since the rolled schedule has two cycles, II = 2

There are 4 predicates so there would be 4 stages in unrolled schedule.

Cycle 0:Load
Cycle 1:Sub

Cycle 2: Add, Mpy
Cycle 3:

Cycle 4:
Cycle 5:Or

Cycle 6:
Cycle 7:Branch

Q10:

5 can be removed
3 has least cost (100/5). spill 3
remove 2 or 6 and the other one
remove 1 and 4

Stack:
1
4
6
2
3 (spilled)
5

Q11:
ALU used by 3 ops => [3/2]
ResMII = 2
MEM used by 2 ops => [2/1]
ResMII = 2
BR used by 1 ops => [1/1]
ResMII = 1

RecMII:
1 - 3 => delay/distance => 4/1 = 4
2 - 3 - 4 - 5 => 7/3 => 3
4 -5 => 2/1 => 2
5 - 5 => 1/1 => 1

MII = max(ResMII, RecMII) = 4

Q.12:
This is backward must data flow analysis. Intersection should be used and the Gen/Kill calculations should be performed on exprs.

for each basic block in the procedure, X, do
    GEN(X) = 0

```
        KILL(X) = 0
        for each operation in reverse sequential order in X, op, do
                G = expr of op
                K = exprs in GEN(X) that use dest of op as operand
                GEN(X) = G + (GEN(X) - K)
                KILL(X) = K + (KILL(X) - G)
        endfor
endfor
```

or the following simple definition of GEN/KILL is also accepted

Kill(X) = { *A op B* | either *A or B* defined before use of *A op B* in X}
Gen(X) = {*A op B* | *A op B* used in X before any definition of *A or B*}

IN/OUT calculation

Kill(Exit) = all expressions
Gen(Exit) = Phi

```
FOR each block in the procedure, X, do
  Out(X) := all expressions
  In(X) := (Out(X) - Kill(X)) ∪ Gen(X))
ENDFOR
WHILE there are changes DO
        FOR each block in the procedure X DO
                Out(X) = Intersect(IN(Y)) for all successors Y of X
                In(X) = (Out(X) - Kill(X)) ∪ Gen(X)
        ENDFOR
ENDWHILE
```

Q13: sample solution

There are many other solutions.

```
r2 = 50
r3 = r2*2
r4 =  r2*2
brz, r3, L1
L1: store(r1, r4)
```

```
r2 = 50
r3 = r2*2
r5 = r3
r4 = r5
brz r3, L1
L1: store(r1, r4)
```

```
r2 = 50
r3 = r2*2
r4 = r3
brz, r3, L1
L1: store(r1, r4)
```

```
r2 = 50
r3 = 50*2
r4 = r3
brz, r3, L1
L1: store(r1, r4)
```

```
r2 = 50
r3 = 100
r4 = r3
brz, r3, L1
L1: store(r1, r4)
```

```
r2 = 50
r3 = 100
r4 = 100
brz, 100, L1
L1: store(r1, 100)
```

```
store(r1, 100)
```