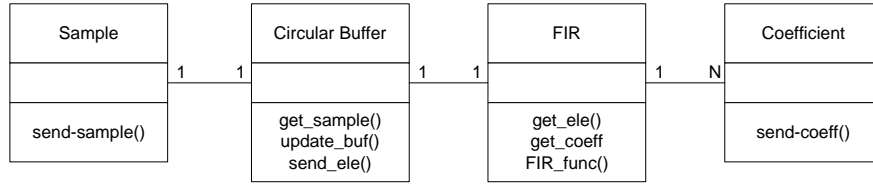


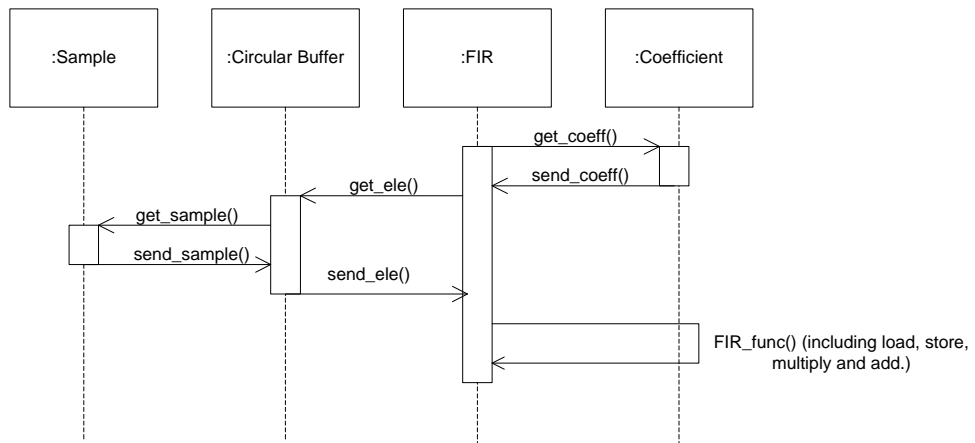
Answers

Q5-1.

a.

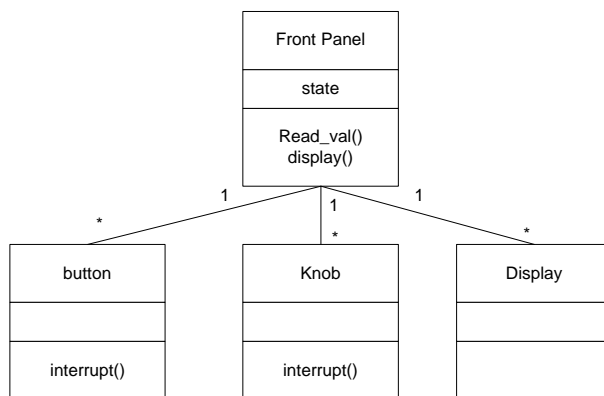


b.

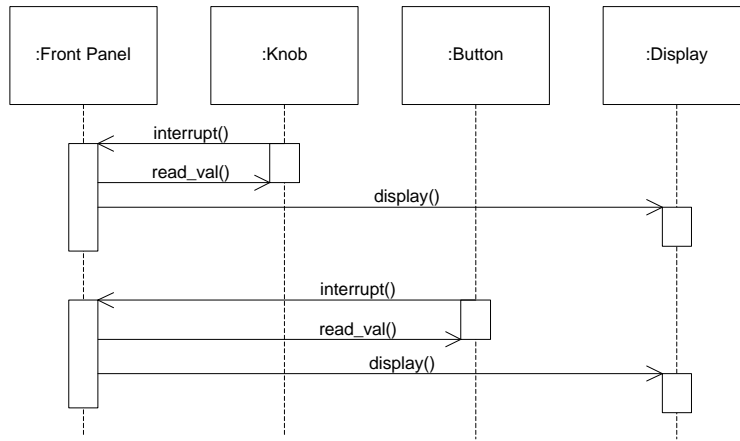


Q5-2.

a.



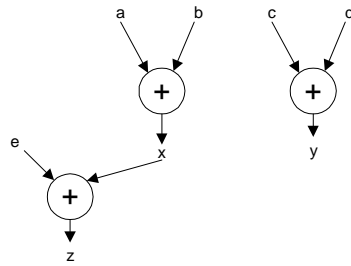
b.



Q5-3.

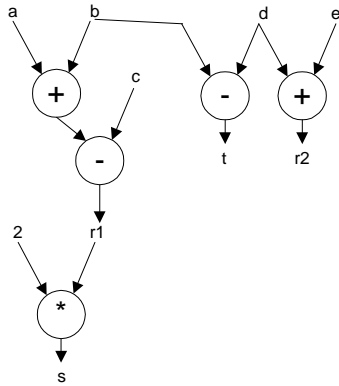
a.

$x = a + b;$
 $y = c + d;$
 $z = x + e;$



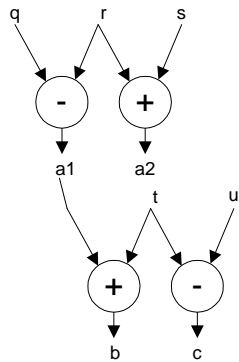
b.

$r1 = a + b - c;$
 $s = 2 * r1;$
 $t = b - d;$
 $r2 = d + e;$



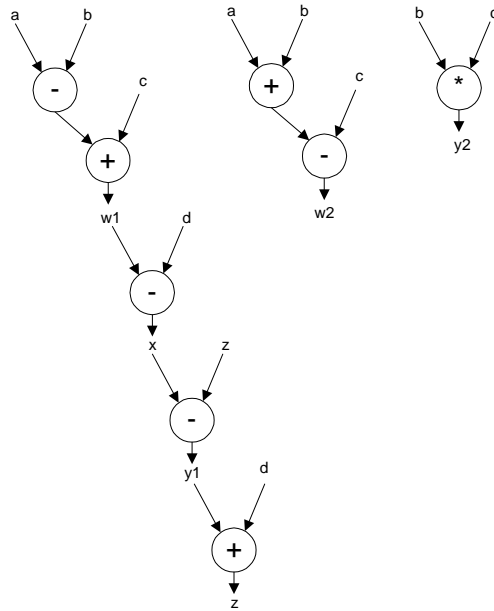
c.

$a1 = q - r;$
 $b = a1 + t;$
 $a2 = r + s;$
 $c = t - u;$



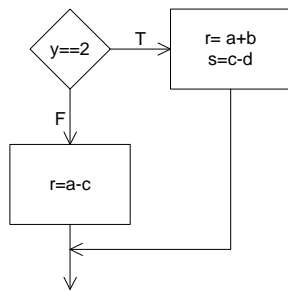
d.

w1=a-b+c;
x=w1-d;
y1=x-z;
w2=a+b-c;
z=y1+d;
y2=b*c;

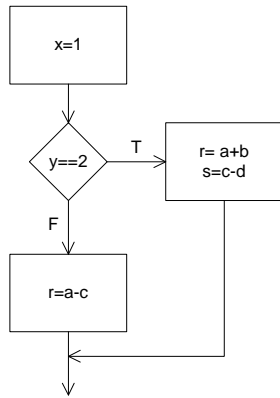


Q5-4.

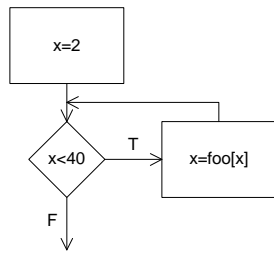
a.



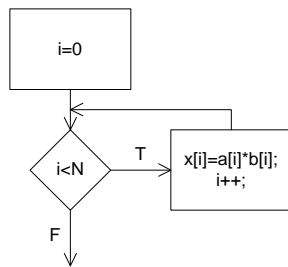
b.



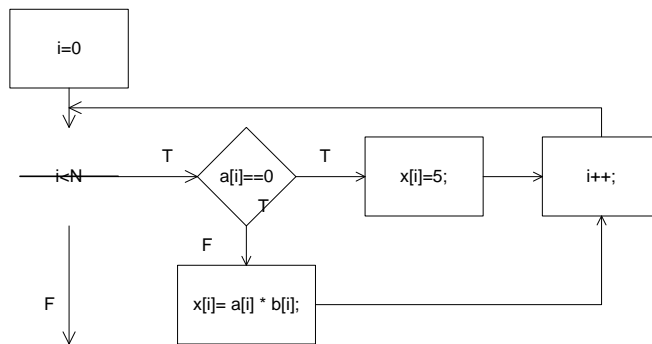
c.



d.



e.



Q5-5.

a.

p1=0x200
p2=0x21C

b.

p1=0x100
p2=0x108
p3=0x110

Q5-6. No, it will not because it will try to resolve reference to things that it has not seen yet in o1 and o2.

Q5-7.

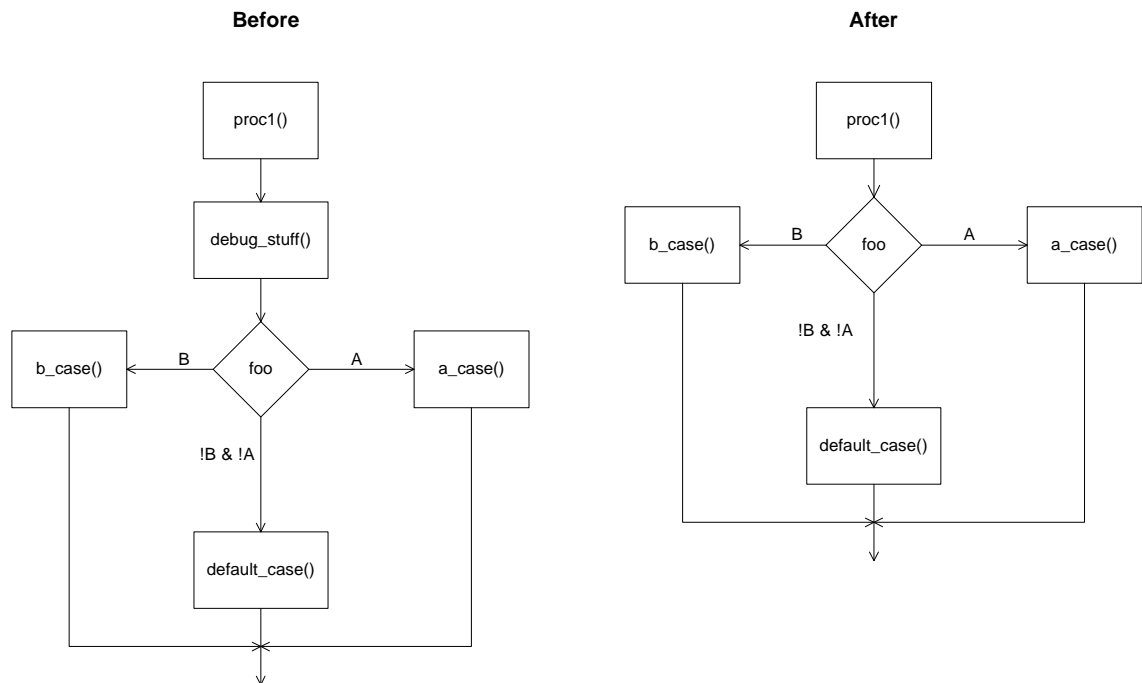
a.

{a+b,c+d}
(a+b) - (c+d)
[(a+b) - (c+d)] + e

b.

{a+b, d+e}
{(a+b)-c, (d+e) + f}

Q5-8.



Q5-9.

```
for(i=0;i<32;i++)
{
    x[i]=a[i]*c[i];
}
```

a. **unrolled 2 times**

```
for(i=0;i<16;i++)
{
    x[(2*i)]=a[(2*i)]*c[(2*i)];
    x[(2*i)+1]=a[(2*i)+1]*c[(2*i)+1];
}
```

b. **unrolled 3 times**

```
for(i=0;i<10;i++)
{
    x[(3*i)]=a[(3*i)]*c[(3*i)];
    x[(3*i)+1]=a[(3*i)+1]*c[(3*i)+1];
    x[(3*i)+2]=a[(3*i)+2]*c[(3*i)+2];
}
x[30]=a[30]*c[30];
x[31]=a[31]*c[31];
```

Q5-10.

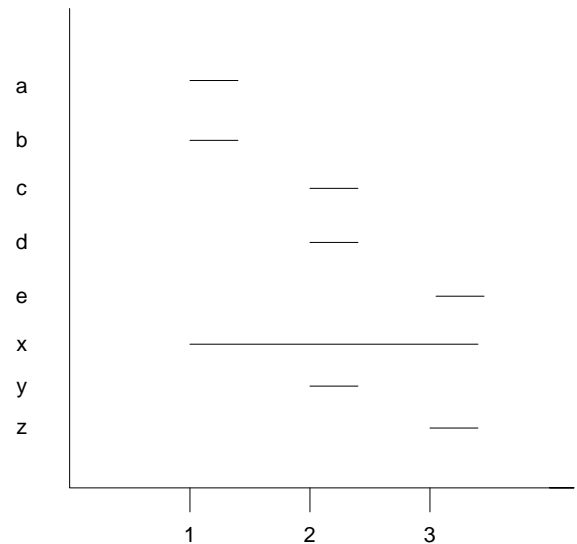
Yes, it is possible. Since the $a[i]$ calculation inside the second loop is effectively a constant, it is possible to move it outside the second loop.

Q5-11.

a.

```
x=a+b;
y=c+d;
z=x+e;
```

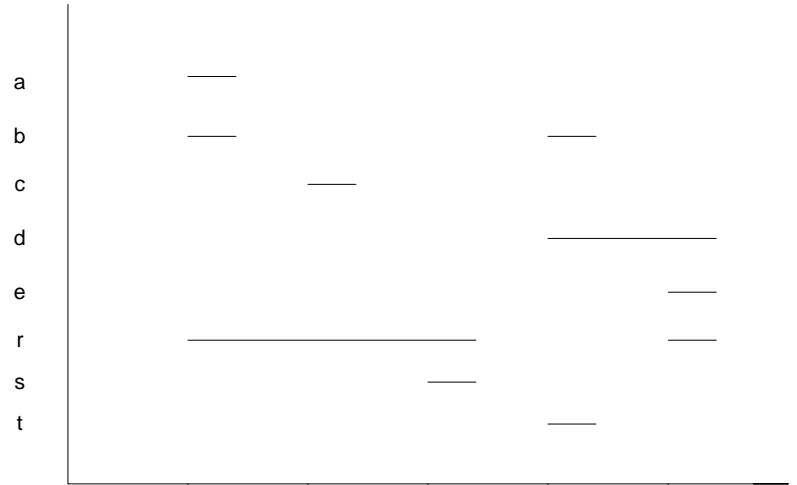
number of registers =4



b.

$r = a + b - c;$
 $s = 2 * r;$
 $t = b - d;$
 $r = d + c;$

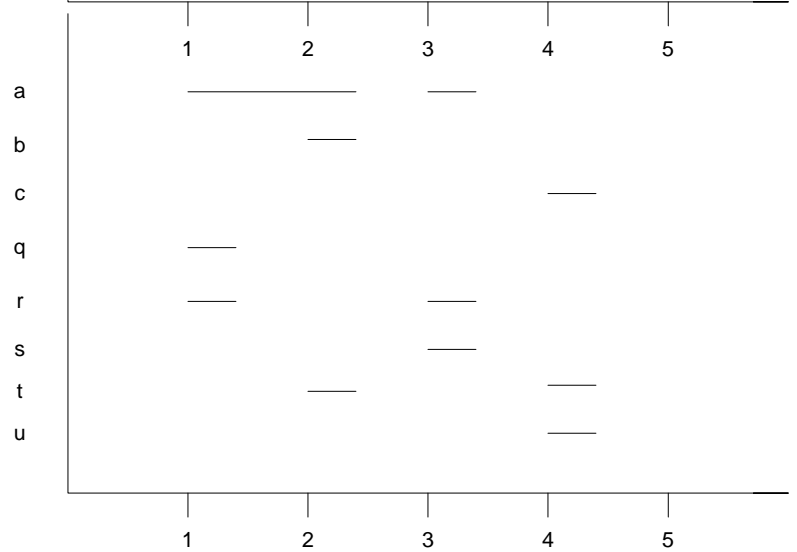
number of registers = 3



c.

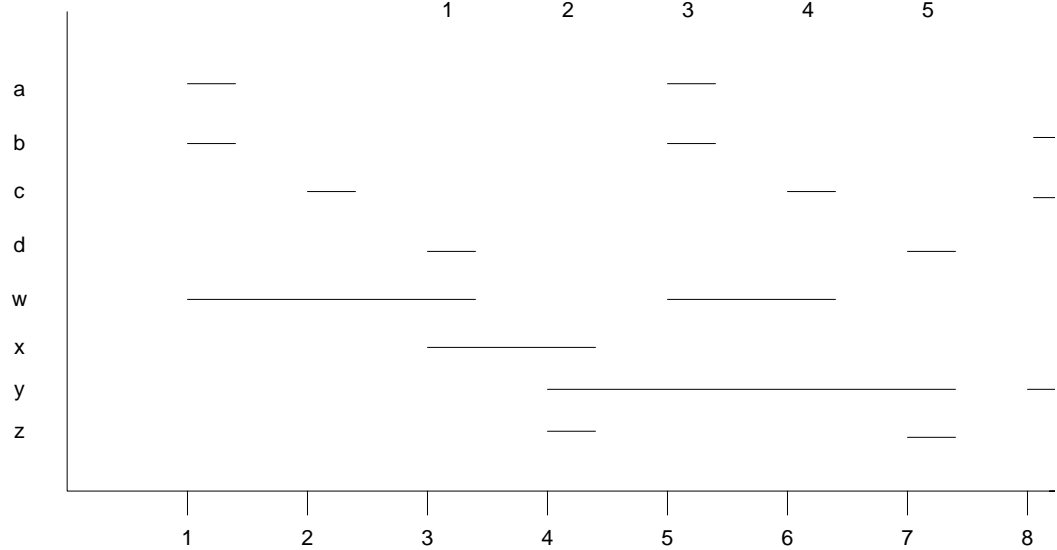
$a = q - r;$
 $b = a + t;$
 $a = r + s;$
 $c = t - u;$

number of registers = 3



d.

$w = a - b + c;$
 $;$
 $x = w - d;$
 $y = x - z;$
 $w = w + a + b - c;$
 $z = y + d;$
 $y = b * c;$



number of registers = 4

Q5-12. One way to solve this problem is to draw lifetime graphs for the various orderings.

a.

$$x=a+b;$$

$$z=x+e;$$

$$y=c+d;$$

number of registers = 3

b.

$$r=a+b-c;$$

$$s=2*r;$$

$$t=b-d;$$

$$r=d+e;$$

number of registers = 3

c.

$$a=q-r;$$

$$b=a+t;$$

$$a=r+s;$$

$$c=t-u;$$

number of registers = 3

d.

$$w=a-b+c;$$

$$x=w-d;$$

$$y=x-z;$$

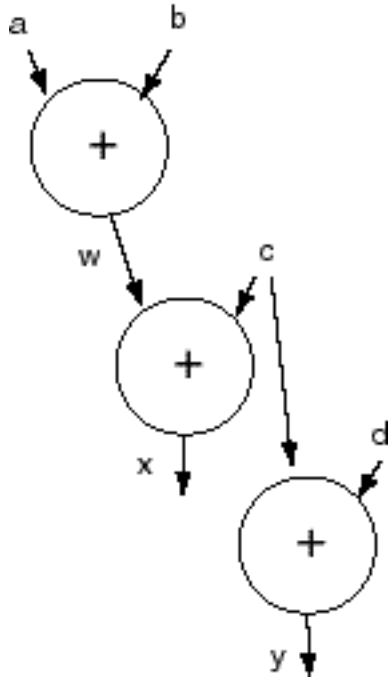
$$z=y+d;$$

$$w=a+b-c;$$

$$y=b*c;$$

number of registers = 3

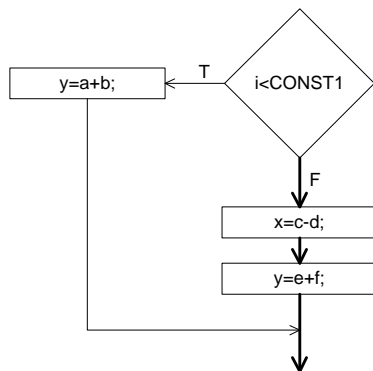
Q5-13. $w=a+b;$
 $x=c+w;$
 $y=c+d;$



The data flow graph shows the flow of data into and out of registers. The number of registers required for any operation is thus equal to the number of edges touching the operation. In a load-store machine, this should in general be 3. By ordering the operations vertically to represent time, we can see how many registers are needed at any one time.

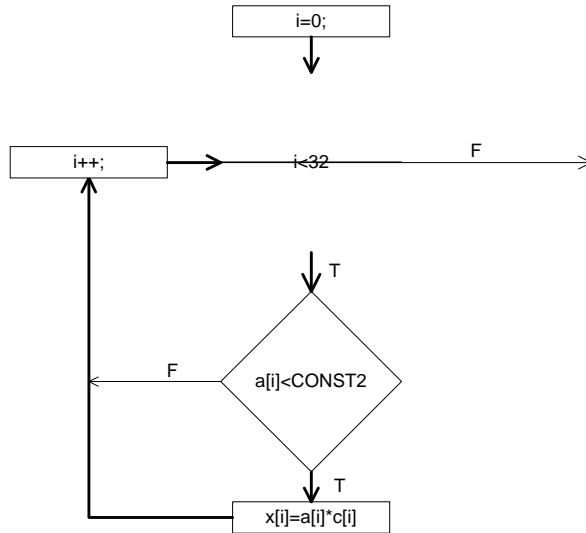
Q5-14.

- a. Longest path is through the false path.



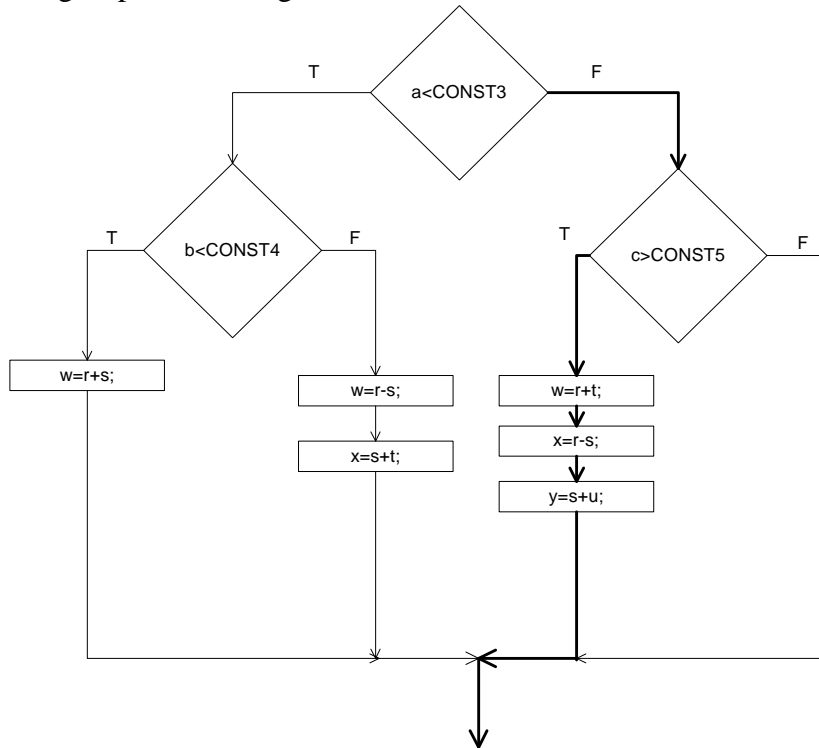
- b.

Longest path is through the for and if statement



c.

Longest path is through the first false and the second true statements



Q5-15. See problem 5-14 for the Flow graphs

- a. Through the true path
- b. Through the $i < 32$ false path
- c. $A \geq \text{CONST3} \ \&\& \ c \leq \text{CONST5}$

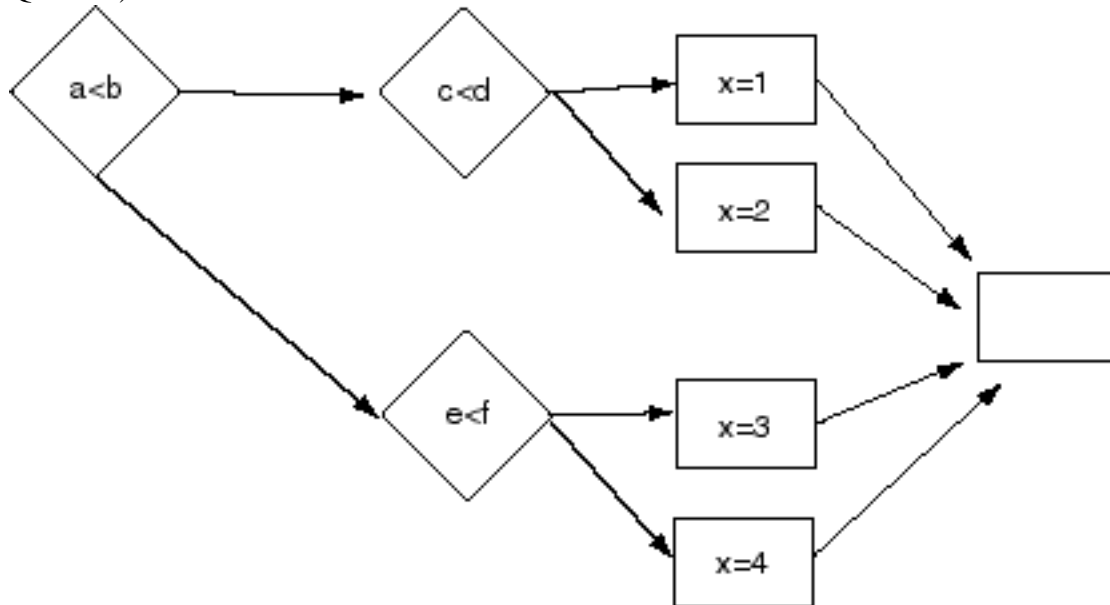
Q5-16.

- a. They must be placed such that they are $1024i$ apart (where i is an integer).
- b. They must be placed $1024i + 3$ apart (where i is an integer).
- c. They should be placed ≥ 4 and ≤ 1020 apart in order to produce no conflicts.

Q5-17.

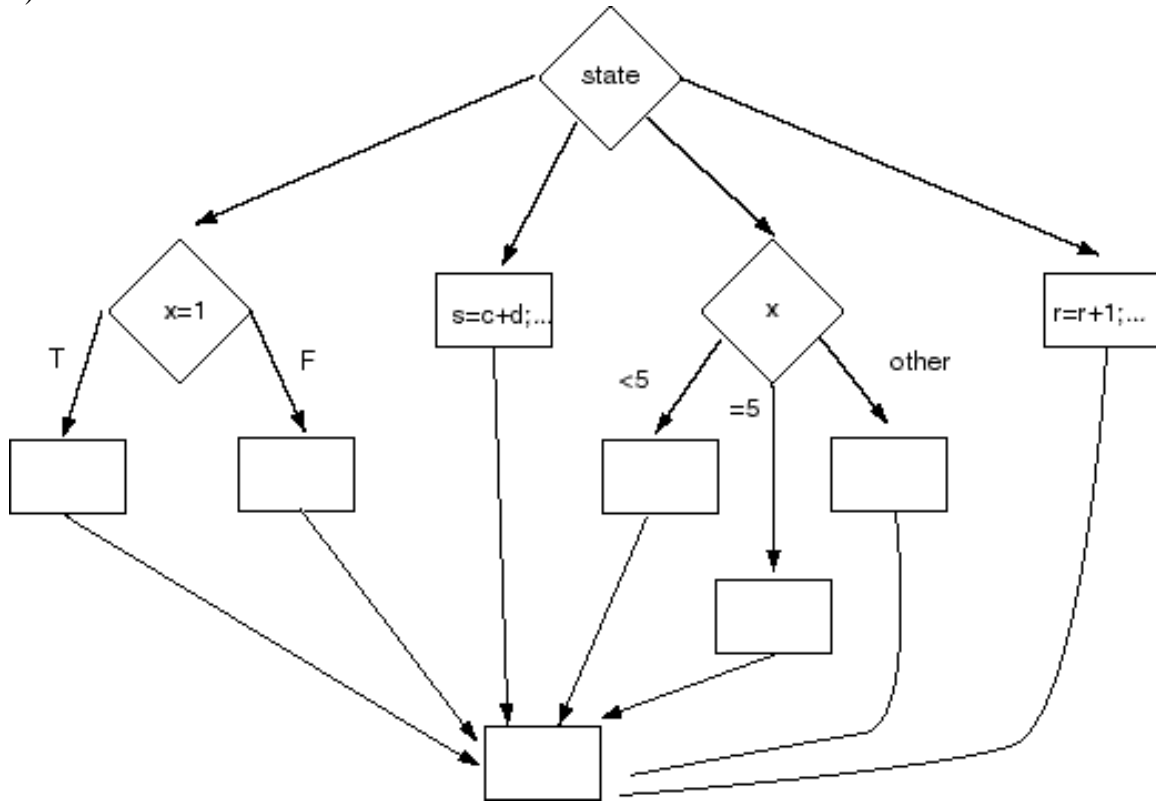
The person who actually wrote the code will be more likely to only test the things that he or she is not confident in. Hence, the person with no knowledge of the system will be more likely to do a more thorough, non-biased job.

Q5-18. a)



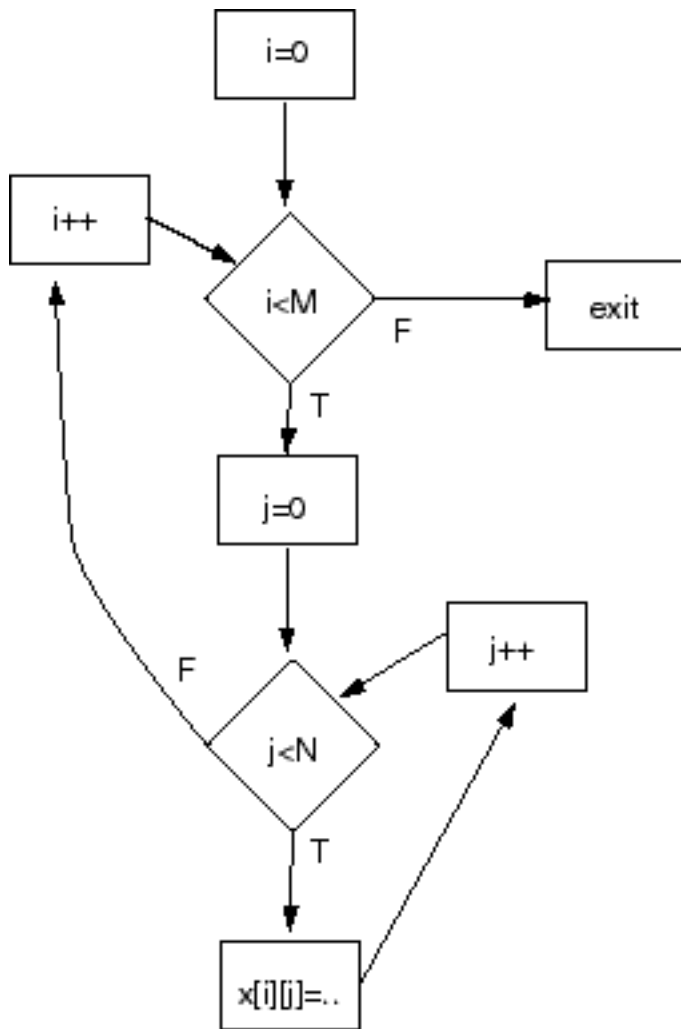
8 nodes, 10 edges, 1 component: $M = 10 - 8 + 2*1 = 4$.

b)



11 nodes, 16 edges, 1 component: $M = 16 - 11 + 2*1 = 7$.

c)



7 nodes, 9 edges, 1 component: $M = 9 - 7 + 2*1 = 4$.

Q5-19.

a.
 if(a<b || ptr1 == NULL) proc1();
 else proc2();

tests to run:

a<=b, ptr1 != NULL
 a>b, ptr1 != NULL
 a<=b, ptr1 == NULL
 a>b, ptr1 == NULL

b.

```
switch(x)
{
    case 0: proc1(); break;
    case 1: proc2(); break;
    case 2: proc3(); break;
    case 3: proc4(); break;
    default: dproc(); break;
}
```

tests to run:

```
x==0
x==1
x==2
x==3
x<0
x>3
```

c.

```
if(a<5 && b >7) proc1();
else if (a<5) proc2();
else if(b>7) proc3();
else proc4();
```

test to run:

```
a<5, b>7
a<5, b<=7
a>=5, b>7
a>=5, b<=7
```

Q5-20.

a.

```
x=a+b;
if(x<20) proc1();
else{
    y=c+d;
    while(y<10)
        y=y+e;
}
```

b.

```
r=10;
s=a-b;
for(i=0; i<10; i++)
    x[i]=a[i]*b[s];
```

c.

```
x=a-b;
y=c-d;
z=e-f;
if(x<10){
    q=y+e;
    z=e+f;
}
if(x<y) proc1();
```

Q5-21.

- x: any value will exercise def-use. Y: any value will exercise first def-use; must use $y < 10$ to exercise second def.
- r is not used. S is always exercised.
- X is not used. To exercise y and z def-use pairs, set $x < 10$.

Q5-22.

By feeding in known values of the $x[i]$'s and the coefficients, it will be possible to verify the results. Additionally, it will be possible to verify the results by running the random tests on a known working system and then comparing the results

Q5-23.

Sorry, this really should be a lab exercise, since the answer depends on the code used.