ECE3073 Computer Systems

Practice Questions

Program Design and Analysis: Compilation

i) Design patterns list four programming design patterns and suggest one typical application for each pattern:

1) State Machine – any application where the program has to sequence through a number of distinct states in a way that is dependent on one or more input. Software to control a coin-in-the-slot vending machine could be based on a state machine.

2) Circular buffer – applications where a program must access a fixed number of previous values in a continuous data stream. Digital FIR filters would use this kind of data storage.

3) Binary tree – binary trees can be used to organise data, for instance they are used in almost every 3D video game to determine what objects need to be rendered.

4) Bubble sort – or any other any other kind of more efficient sort. It can be used to sort a set of data values into the order of their magnitude.

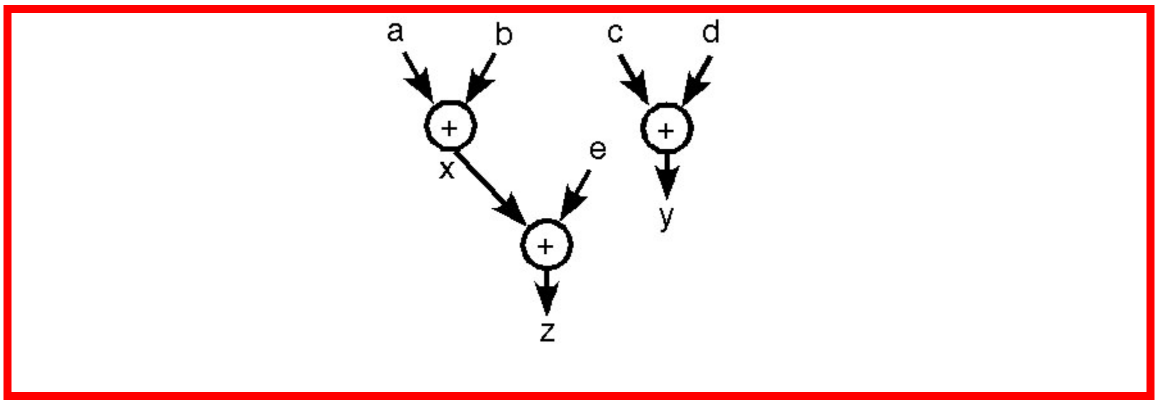
ii) Data Flow Graph

For each block of code given below rewrite in single assignment form and then draw the data flow graph for that form. From the graph determine the partial order.

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

single assignment form:

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



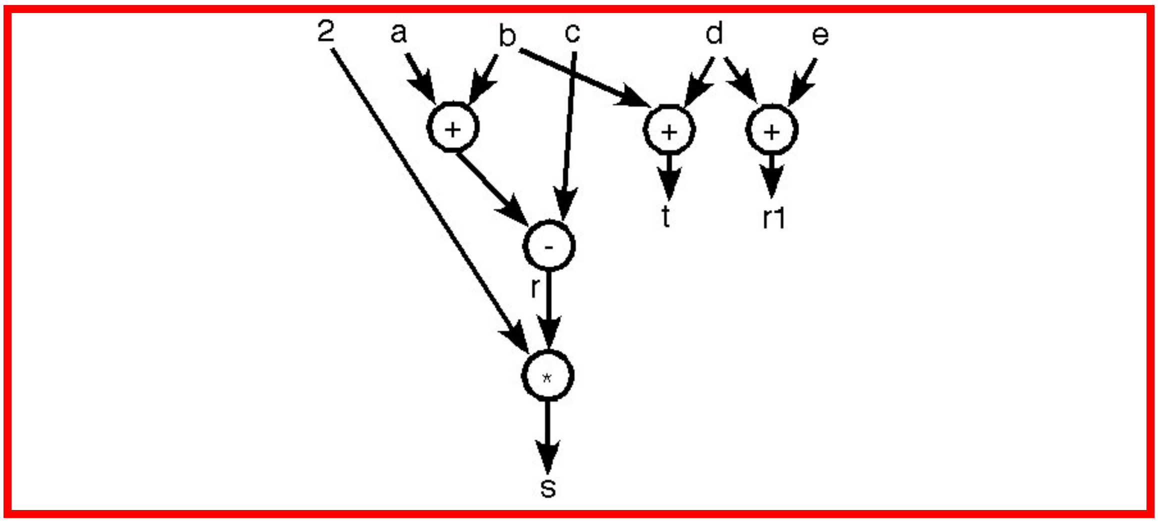
partial order

1) a + b, c + d
2) x + e

b) r = a + b - c
s = 2 * r
t = b + d
r = d + e

single assignment form:

r = a + b - c
s = 2 * r
t = b + d
r1= d + e



Partial order:

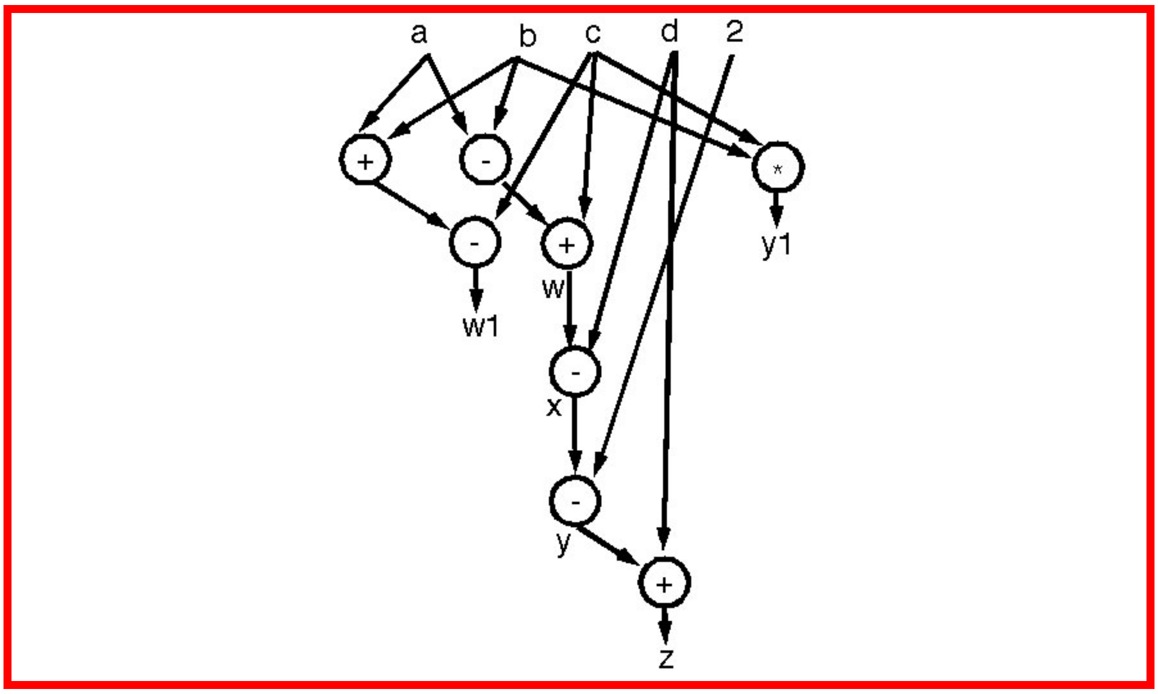
1) a + b – c, b + d, d + e
2) 2 * r

c) w = a - b + c

x = w - d
y = x - 2
w = a + b – c
z = y + d
y = b * c

single assignment form:

w = a - b + c
x = w - d
y = x - 2
w1 = a + b – c
z = y + d
y1 = b * c

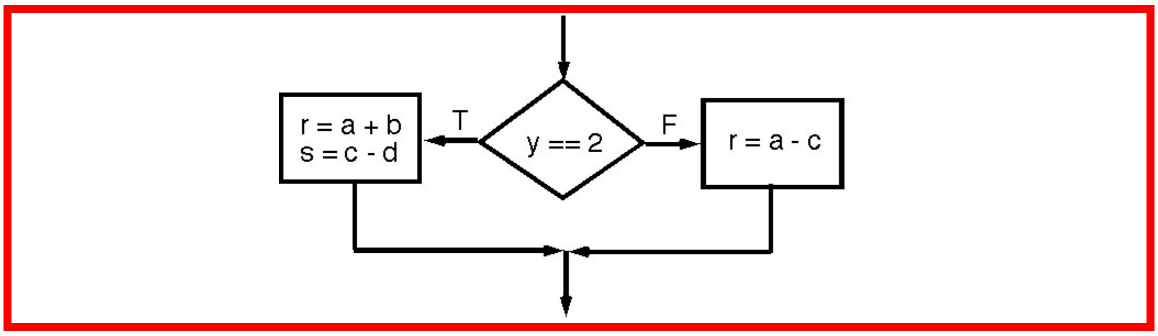


Partial order:

1) a – b + c, a + b – c, b * c
2) w – d
3) x – 2
4) y + d

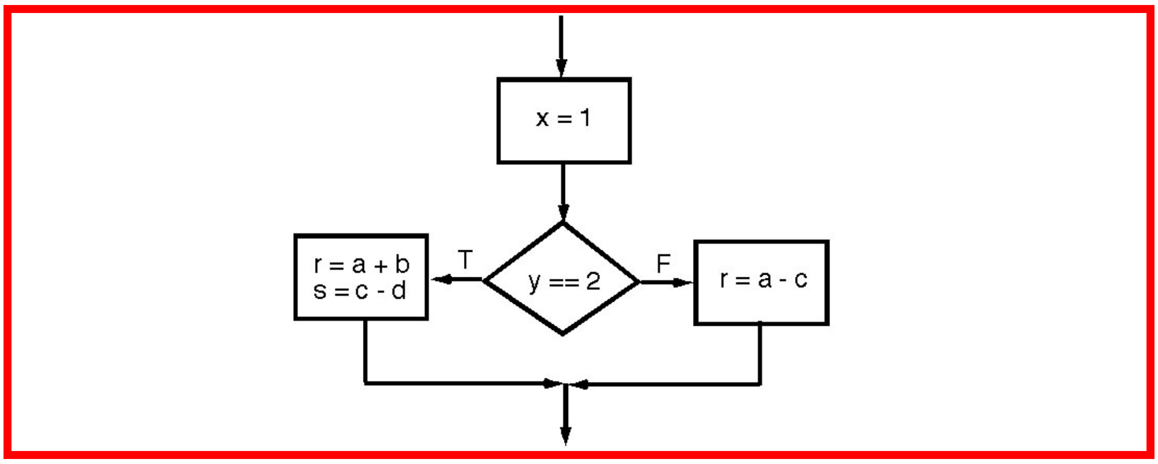
ii) Draw the Control Data Flow Graph for the following code fragments:

a) if(y == 2) {r = a + b; s = c – d;}
else r = a – c;



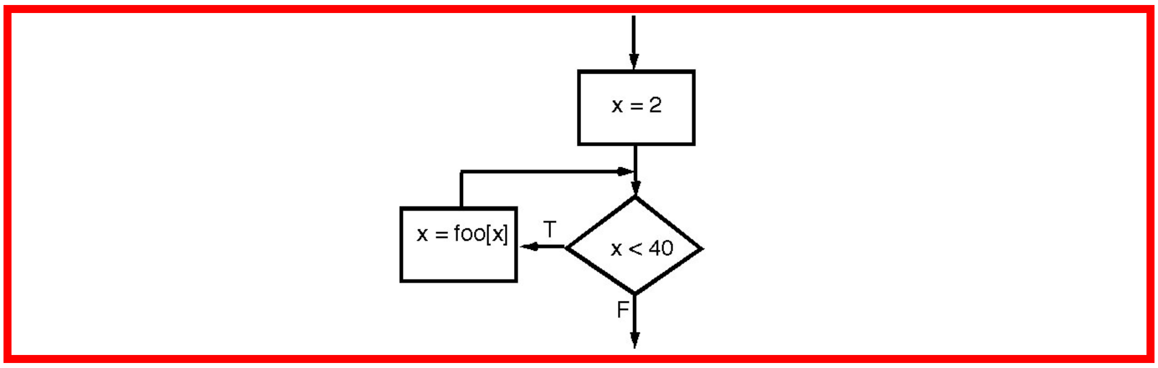
b)

```
x = 1;
if ( y == 2 ) {r = a + b; s = c – d;}
else ( r = a – c}
```



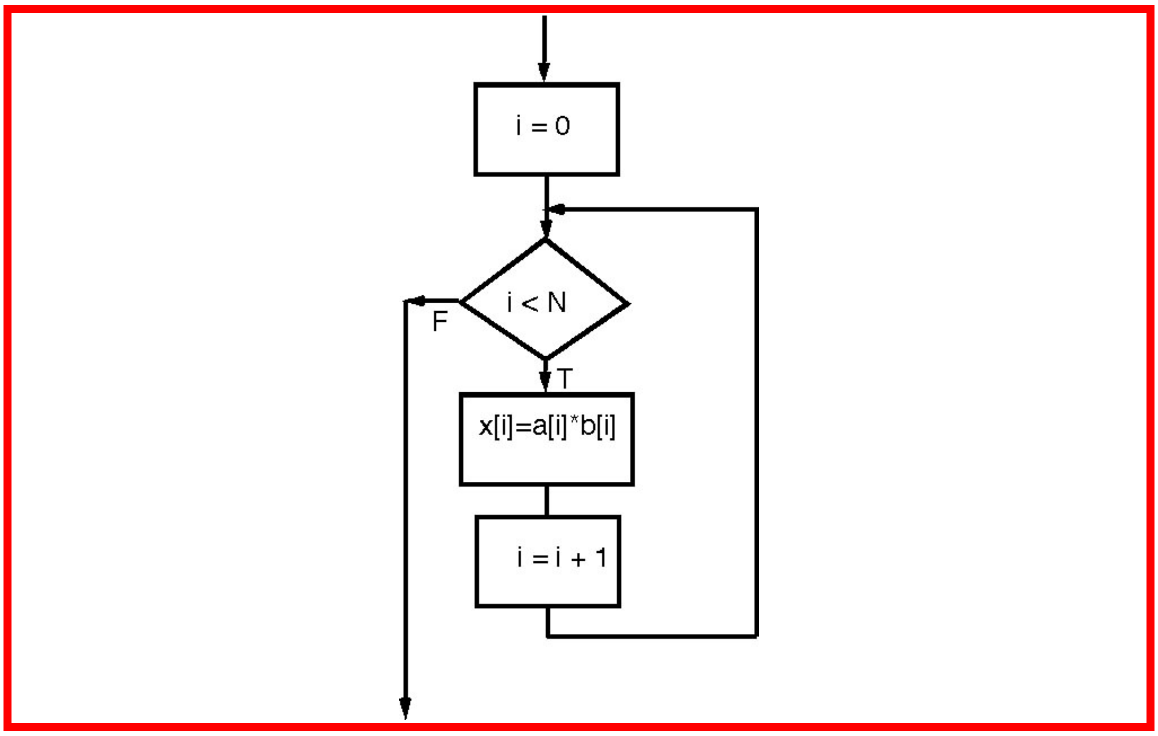
c)

```
x = 2;
while ( x < 40 ) {
        x = foo[x];
}
```

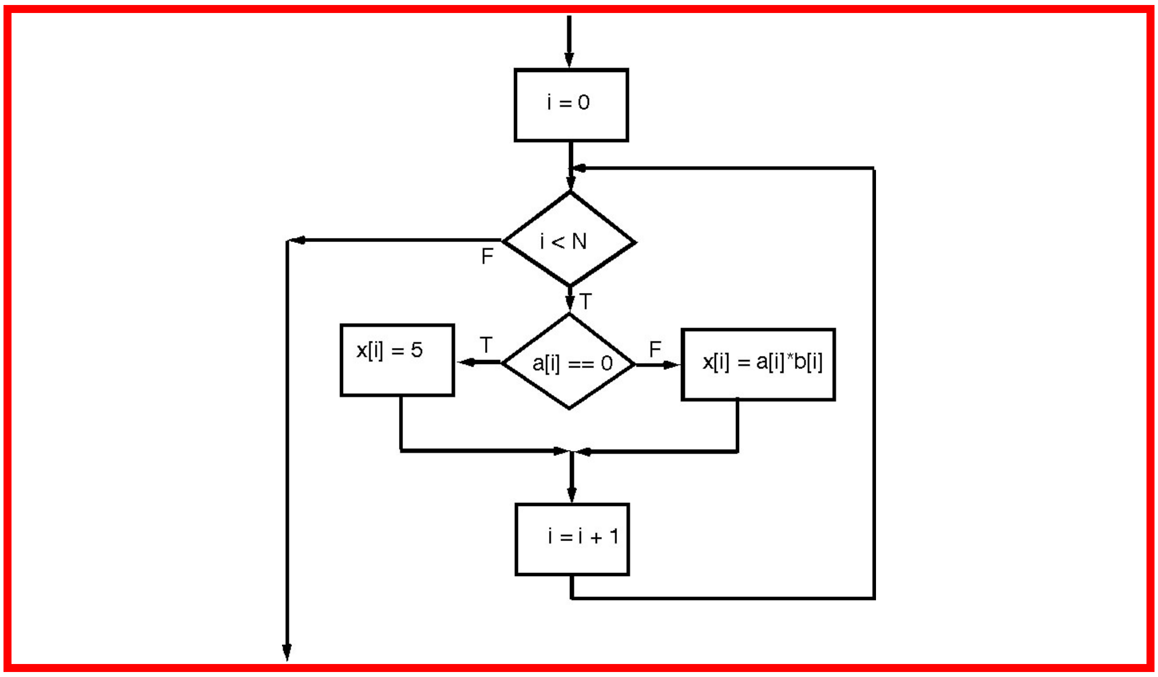


d)

```
for ( i = 0; i  <  N; i++)
        x[i] = a[1] * b[i];
```



e) for (i = 0; i < N; i++) {
   if (a[i] == 0) x[i] = 5;
   else x[i] = a[i] * b[i];



Show the contents of the assembler's symbol table at the end of code generation for the following programs (Note that each instruction occupies 4 bytes)

a)

```
        ORG 200
p1      ADR r4,a
```

```
        LDR r0.[r4]
        ADR r4,e
        LDR r1,[r4]
        ADD r0,r0,r1
        CMP r0,r1
        BNE q1
p2      ADR r4,e
```

p1	200
p2	228

b)

```
        ORG 100
p1      CMP r0,r1
        BEQ x1
p2      CMP r0,r2
        BEQ x2
p3      CMP r0,r3
        BEQ x3
```

p1	100
p2	108
p3	116

iii) Explain the difference between row major format and column major format for storing arrays.

For a 2-dimensional array the first index references the rows of the array and the second the columns. When storing a 2-dimensional array in computer memory elements are stored in successive memory locations. For row major organisation complete rows of the array are stored one after the other. Therefore, in successive memory locations the column address is incremented for each location, the row address is only incremented when all of the columns is a row have been covered. In column major format the roles of the rows and columns in the above description are swapped.

iv) In code optimisation

a) In the context of expression simplification briefly explain:

1) Constant folding

Processing parts of an expression involving constants that can be pre-computed, for example:

4.0 / 3.0 * 3.14159 * r * r * r = 4.18878 * r * r * r

2) Algebraic expression simplification

Reorganising algebraic expressions to reduce the number of mathematical operations required and to use faster operations, for example:

a * b + a * c = a * (b + c) // two operations rather than three

3) Strength reduction

In some special situations an expression can be reformulated so that it can be performed by a faster computer operation. For example:

8 * x = x <<3 (shift is quicker than multiplication)

b) Explain what is meant by dead code elimination and point out a situation where this could cause a problem.

c) Describe procedure inlining and explain its advantages and disadvantages

When a procedure is called the processor must arrange to pass parameters to the procedure, the procedure must recover the parameters, perform its task and then arrange to pass the results back to the calling program. If the code for the task is inserted into the calling program the overhead for passing parameters is not required and the program will execute faster. However, the resulting code will be longer, particularly if the same procedure is called multiple times.

d) Describe loop unrolling and explain its advantages and disadvantages

Instead of using a loop structure the code for the loop is written out repeatedly the same number of times that the loop would execute. This is quicker because the overhead of managing the loop variable (updating each time through the loop and testing for completion) is not required. However, the code is longer.