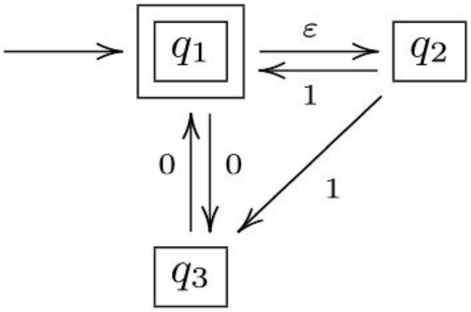
Section 01 (There are 4 questions, answer any 3 out of them) [12 *3 = 36]

1. a. Draw the block diagram of basic compilation phases. [3]
   b. Define the difference between Lexical Analyzer and Syntax Analyzer. [3]
   c. Convert following NFA to DFA using subset construction methodology. [3]

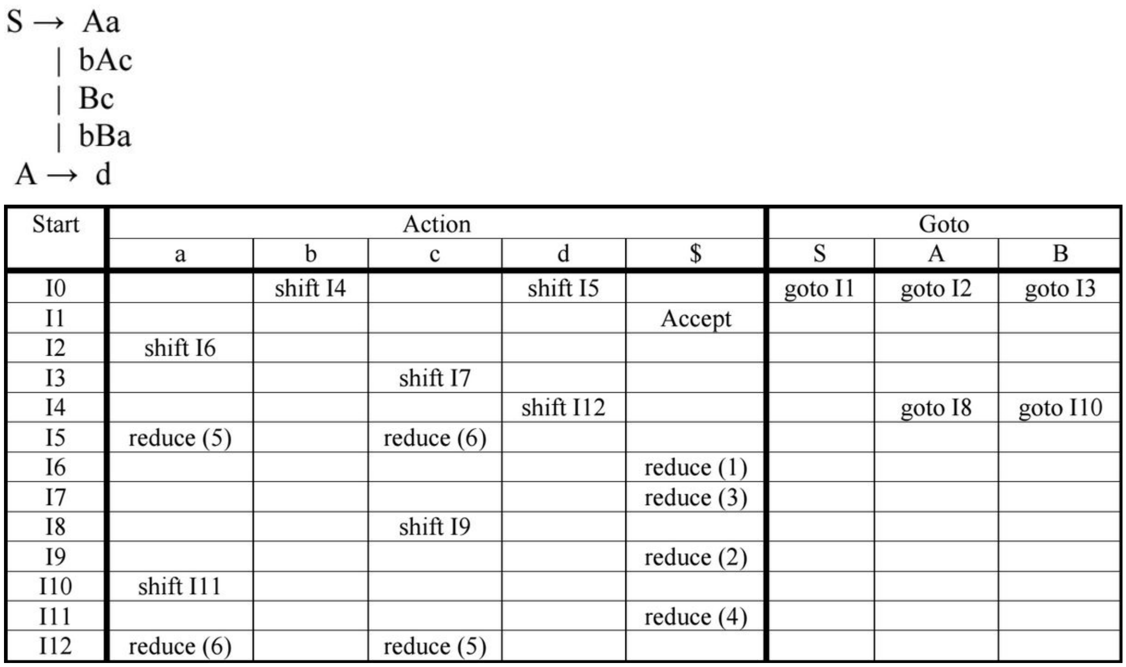


   d. Convert the regular expression mm(m|n)*n over the alphabet Σ = {m,n} directly to DFA. [3]

2. a. What is meant by left recursion in a grammar? [1]
   b. Consider the grammar with the set of terminals:
   S → (L) | a | b
   L → L,S | S
   Remove left-recursion from the grammar and find the First and Follow sets for each non-terminal of the modified grammar. [1+1+2]
   c. For the following grammar, construct the LR(1) parse table.[5]
   X → X + X | Y++
   Y → a
   d. What do you understand by shift-reduce and reduce-reduce conflicts? [2]

3. Construct the LALR parser table for the following grammar. Show all the necessary steps. [12]
   P → PaQ | Q
   Q → QR | R
   R → Rb | c | d

4. a. Construct the LR (0) Automation of following grammar. [6]
   E → E + T
   E → T
   T → T F
   T → F
   F → F *
   F → a
   F → b
   b. Parse input string “bda$” using following grammar and parsing table: [6]

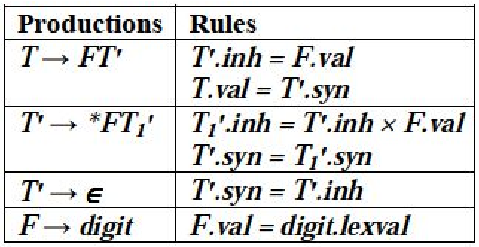


$S \rightarrow Aa$
$\quad | \; bAc$
$\quad | \; Bc$
$\quad | \; bBa$
$A \rightarrow d$

Start	Action					Goto		
	a	b	c	d	$	S	A	B
I0		shift I4		shift I5		goto I1	goto I2	goto I3
I1					Accept			
I2	shift I6							
I3			shift I7					
I4				shift I12			goto I8	goto I10
I5	reduce (5)		reduce (6)					
I6					reduce (1)			
I7					reduce (3)			
I8			shift I9					
I9					reduce (2)			
I10	shift I11							
I11					reduce (4)			
I12	reduce (6)		reduce (5)					

<u>Section 02: (There are 3 questions, answer any 2 out of them) [10*2 =20]</u>

1. a. Define synthesized and inherited attributes. [2]
   b. Translate the arithmetic expression a*-(b+c) into: [3]
      i. A syntax tree
      ii. Postfix notation
      iii. Three-address code
   c. Consider the following SDD: [3+2]



Productions	Rules
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow {*}FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
$F \rightarrow digit$	$F.val = digit.lexval$

      i. Draw the annotated parse tree for the expression 3*5*7 using the semantic rules given above.
      ii. Draw the dependency graph.

2. a. Consider the following grammar: [3+4=7]

      T → FT’
      T’ → +FT’
      T’ → ε
      F → 1 | 2 | 3 | . . . | 9

i)Construct an SDD for the grammar.

ii) Using SDD constructed in (i) give an annotated parse tree for the expression: 2+3+4.

b. Discuss the comparative advantages and disadvantages of the following three representations: [3]

i. Quadruples

ii. Triples

iii. Indirect triples

3. a. What is backpatching? What is the advantage of backpatching? Explain with an example. [1+1+2]

b. Consider the following code fragment:

```
do i=1+1;
f=i*5000; while (a[i] < v);
```

Write the three address code and its quadruple representation. [2+2]

c. Determine the equation to determine: [1+1]

i. i’th element of a 1-dimensional array.

ii. (i, j)’th element of a 2-dimensional array.

Section 03: (There are 2 questions, answer any 1 out of them) [1*14 = 14]

1. a. Write short notes on the following: [2+2]

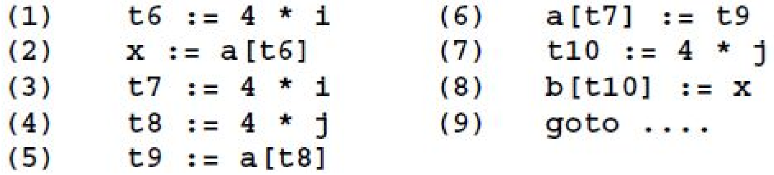
i. Basic blocks

ii. Peephole optimization

b. Draw the flow graph for the following program: [4]

```
begin
  prod := 0;
  i := 1;
  do begin
     prod := prod + a[i] * b[i];
     i := i + 1;
  end
  while (i <= 20);
end
```

c. For the following code fragment, determine the next-use information: [6]



```
(1)   t6 := 4 * i       (6)   a[t7] := t9
(2)   x := a[t6]        (7)   t10 := 4 * j
(3)   t7 := 4 * i       (8)   b[t10] := x
(4)   t8 := 4 * j       (9)   goto ....
(5)   t9 := a[t8]
```

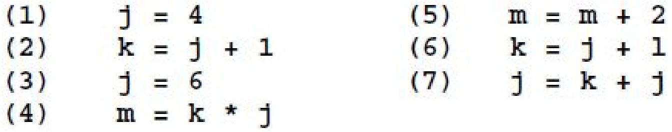
2. a. Consider the following fragment of intermediate code: [4]

```
w = 2
```

```
u = z
y = w + 1
v = y * y
r = v ** 2 //this is exponentiation
t = u * u
s = u * t
x = y * y
```

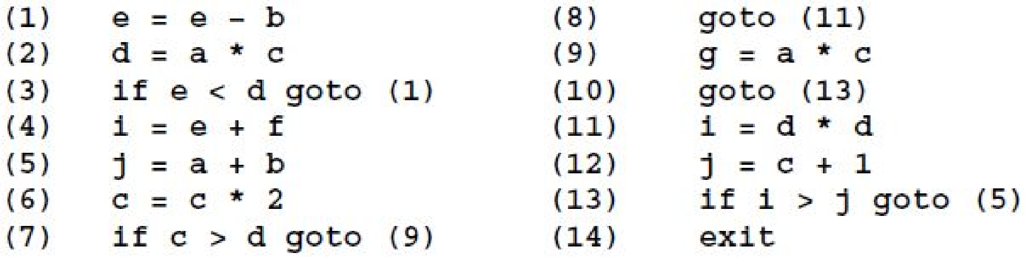
Assume the only variables live at the exit are s, x. Show the result of applying constant propagation, algebraic simplification, common sub-expression elimination, constant folding, copy propagation and dead code elimination as much as possible to this code. You should explain the changes in each step.

b. For the following code fragment, list all the dependencies between statements and draw the dependency graph. [3]



```
(1)   j = 4             (5)   m = m + 2
(2)   k = j + 1         (6)   k = j + 1
(3)   j = 6             (7)   j = k + j
(4)   m = k * j
```

c. Consider the following sequence of 3-address codes: [2 + 5]



```
(1)   e = e - b              (8)    goto (11)
(2)   d = a * c              (9)    g = a * c
(3)   if e < d goto (1)      (10)   goto (13)
(4)   i = e + f              (11)   i = d * d
(5)   j = a + b              (12)   j = c + 1
(6)   c = c * 2              (13)   if i > j goto (5)
(7)   if c > d goto (9)      (14)   exit
```

i. Draw the flow graph.

ii. Compute live variables at the end of each block using the iterative solution to dataflow equations for live variable analysis.