INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Date .....FN / AN *Time:* 2/3 Hrs. *Full Marks* ...90... *No. of Students* .50....
Autumn / ~~Spring~~ Semester, ~~19~~2000-2000 Deptt. ..CSE..... Sub No. 173003.
3RD. Yr. B. Tech.(Hons.) / ~~B. Arch.~~ / ~~M. Sc.~~ Sub. Name .Compiler Design...

Instructions : **Answer Any Three Questions** *Taking At Least One* **From Each Group.**

Group A

Question 1 **[8 + 12 + (4 + 6)]**

Consider the grammar $G_1 = (\{ ; \text{ id} = + - * /) (\text{int\_const}\}, \{P, S, E, T, F\}, R, P)$ and answer the following questions.

$$
\begin{aligned}
P &\rightarrow S \texttt{ ; } P \mid S \\
S &\rightarrow \texttt{id = } E \mid E \\
E &\rightarrow E \texttt{ + } T \mid E \texttt{ - } T \mid T \\
T &\rightarrow T \texttt{ * } F \mid T \texttt{ / } F \mid F \\
F &\rightarrow (S) \mid \texttt{id} \mid \texttt{int\_const}
\end{aligned}
$$

Assume that *integer* is the only data type.

1. If the grammar G_1 is not $LL(1)$, transform it (without changing the formal language) to $LL(1)$ form.
2. Construct the *Predictive Parsing Table* for the modified grammar.
3. Show the parse tree for the input 'y = 5 ; x = 2 * y', and explain how the semantic actions are to be associated with the parsing process (of this example). Note that the parsing here is **top down.**

Question 2 **[12 + 12 + 3 + 3]**

1. Show that the grammar $G_2 = (\{a, b, c\}, \{S, T\}, R, S)$ is $LR(1)$ but not $LALR(1)$.

$$
\begin{aligned}
S &\rightarrow aSa \mid bSb \mid aTb \mid bTa \mid c \\
T &\rightarrow c
\end{aligned}
$$

2. Construct the *operator precedence table* for the grammar $G_3 = (\{\texttt{; while do id := + ==}\}, \{S, E\}, R, S)$. The symbols have their usual meaning (associativity, precedence etc).

$$
\begin{aligned}
S &\rightarrow S \texttt{ ; } S \mid \texttt{while } E \texttt{ do } S \mid \texttt{id := } E \\
E &\rightarrow E \texttt{ + } E \mid E \texttt{ == } E \mid \texttt{id}
\end{aligned}
$$

3. Both $LL(1)$ and $LR(1)$ parsers use stack to store some information. In what way are the contents of the stacks in these two cases different? Explain your answer with examples.
4. For a language construct you may either have a *left recursive* or a *right recursive* grammar. Which one do you prefer for writing an $LALR(1)$ parser. Explain your answer with an example.

[illegible] **[6 + 6 + 6 + 6 + 6]**

[illegible] mantic actions associated to the following constructs of our language. You may [illegible] simplifying assumptions and also modify the production rules without changing [illegible]

1. [illegible] *tatement* → LOOP *exp* TIMES *statement*.
2. [illegible] *statement* → WHILE *exp* DO *statement*. (Assume that it is there).
3. [illegible] *call_fun* (*act_para_list_opt*)
   [illegible] → ID
   [illegible] *list_opt* → NULL | *act_para*
   [illegible] → *act_para* , *act_para* | *exp*
4. [illegible] *array_name array_elem*
   [illegible] *name* → ID
   [illegible] *elem* → *array_elem* [*exp*] | [*exp*]
5. [illegible] modification do you suggest in the semantic actions and (1) and (2) if the '*break*' [illegible] ment is present?

Group B

[illegible] **[15 + 10 + 5]**

[illegible] are the basic blocks (Block-1 to Block-7) of a procedure.

```
[illegible]
  [illegible] $sp  40
  [illegible]
  [illegible]
  [illegible]
  [illegible]
  [illegible]
  [illegible]

[illegible] 2
  [illegible]
  [illegible]
  [illegible] 4 < v5
  [illegible] goto Block-3
  [illegible] Block-7

[illegible] 3
  [illegible]
  j [illegible]

[illegible] 4
  [illegible]
  [illegible]
  [illegible] 8 < v9
  [illegible] goto Block-5
  [illegible] Block-6

Block-5
  v11 = a
  v12 = i
  v13 = 20
  v14 = v12 * v13
  v15 = j
  v16 = v14 + v15
  v17 = v11 + v16
  v18 = *v17
  t = v18
  v19 = a
  v20 = i
  v21 = 20
  v22 = v20 * v21
  v23 = j
  v24 = v22 + v23
  v25 = v19 + v24
  v26 = a
  v27 = j
  v28 = 20
  v29 = v27 * v28
  v30 = i
  v31 = v29 + v30
  v32 = v26 + v31
  v33 = *v32

Block-5(contd.)
  *v25 = v33
  v34 = a
  v35 = j
  v36 = 20
  v37 = v35 * v36
  v38 = i
  v39 = v37 + v38
  v40 = v34 + v39
  v41 = t
  *v40 = v41
  v42 = j
  v43 = 1
  v44 = v42 + v43
  j = v44
  goto Block-4

Block-6
  v45 = a
  v46 = i
  v47 = 20
  v48 = v46 * v47
  v49 = i
  v50 = v48 + v49
  v51 = v45 + v50

Block-6(contd.)
  v52 = 2
  v53 = n
  v54 = v52 * v53
  *v51 = v54
  v55 = i
  v56 = 1
  v57 = v55 + v56
  i = v57
  goto Block-2

Block-7
  $sp = $sp + 40
  goto $ra
```

The *control-flow graph* of the procedure is shown in Figure 1. Improve the code of the

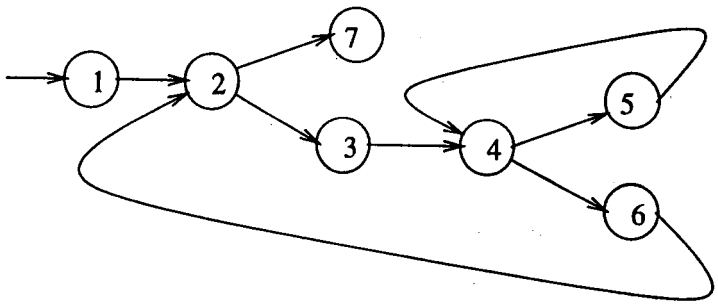

Figure 1: Control-Flow Graph

rocedure in the following way (Don't reuse the virtual registers).

1. Improve the code in every basic block by local value numbering, constant folding, constant propagation, copy propagation, common subexpression elimination, strength reduction and useless instruction elimination. Show the improved code of **Block-5** with proper explanation.

2. Transform the blocks in the *static single assignment* (SSA) form and perform global value numbering. Show the relevant portions of each block and explain your answer.

3. Without doing any formal analysis try to do global improvement of the code e.g. elimination of common subexpression, moving *loop-invarients* out of the loop etc. Show only the relevant portions.

Question 5 **[10 + 10 + 10]**

Consider the following program in our language.

```
insert(int m[], int i, int data) -> null {
  int j ;
  for(j = i - 1; j >= 0; j = j - 1)
    if m[j] < data then m[j + 1] = m[j]
    else break ;
  m[j + 1] = data
}
```

1. Translate it to the *3-address code* intermediate form. Do not reuse the compiler defined variables (the virtual registers).

2. Consider the largest basic block (having the maximum number of instructions) B. Use *graph colouring* to allocate mimimum number of physical registers (`$1, ···, $8, $22, ···, $25`) to the virtual registers.

3. After the register allocation translate the basic block B to DEC Alpha assembly code. [Some of the DEC Alpha instructions are { `lda, ldl, ldq, ldil, stl, stq, addl, subl, mull, not, or, and, cmpeq, cmplt, cmple, sll, srl, sra, beq, bne, blt, ble, bgt, bge, ret` }.]