NitteMeenakshi Institute of Technology

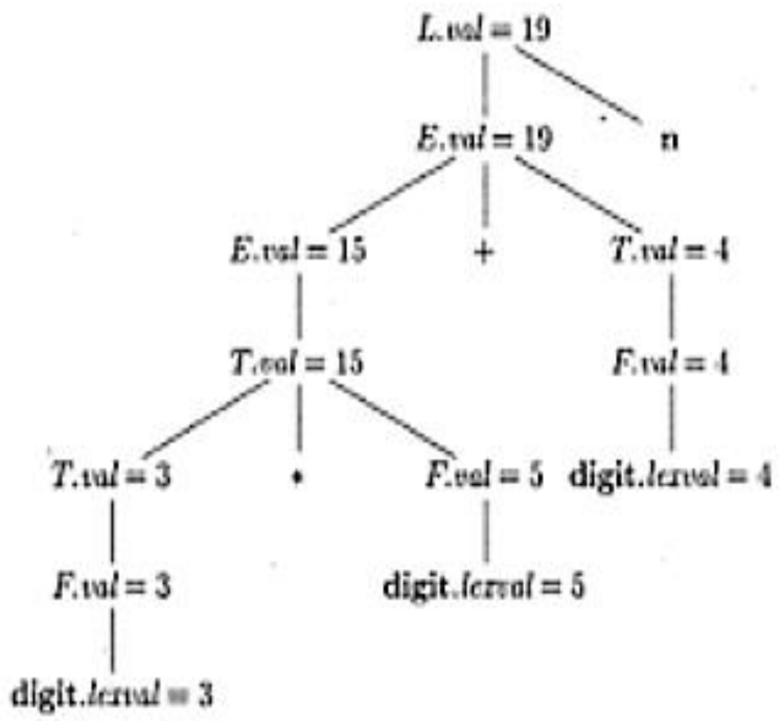
(AN AUTONOMOUS INSTITUTION AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM)
PB No. 6429, Yelahanka, Bangalore 560-064, Karnataka
Telephone: 080- 22167800, 22167860
Fax: 080 - 22167805

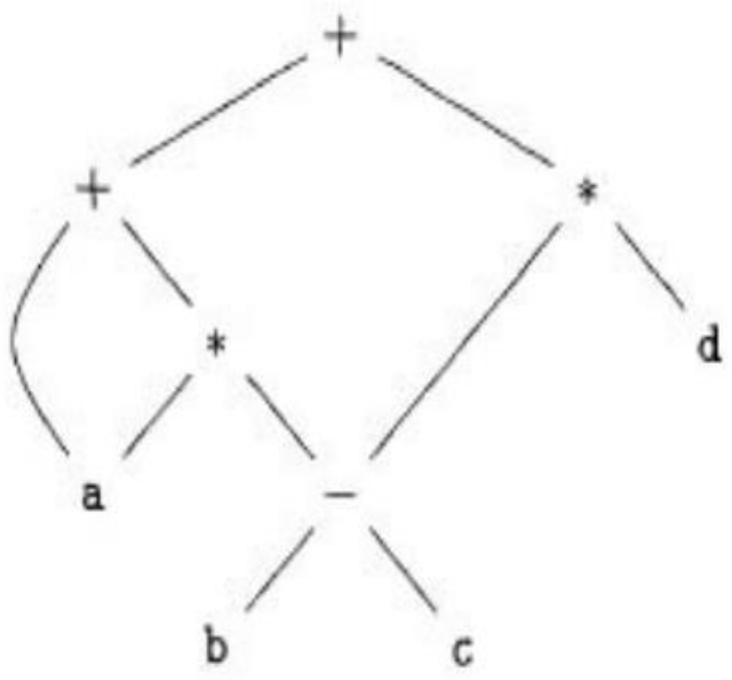
Department of Computer Science and Engineering

MID-SEM EXAMINATION-II ANSWER SCHEME

Semester/ Section:7/A/B/C — **Date: /10/2019**
Subject and Code:Compiler Design, 14CS73 — **Duration: 1.0 Hr**
Faculty Name: Dr. Saroja Devi H./Uma R./Kavya B.S. — **Max Marks: 30**

Note: Section A is Compulsory
Answer any two questions from Section B

		Section A		
Q. No		**Questions**	**Marks**	**CO/PO/Bloom levels mapping**
1	a	**Annotated parse tree for 3*5+4n** 	2	CO3/L6
	b	**3-address code for the expression (a+a*(b-c)+(b-c)*d)**	2	CO3/L6

		
		t1 = b – c
		t2 = a * t1
		t3 = a + t2
		t4 = t1 * d
		t5 = t3 + t4		

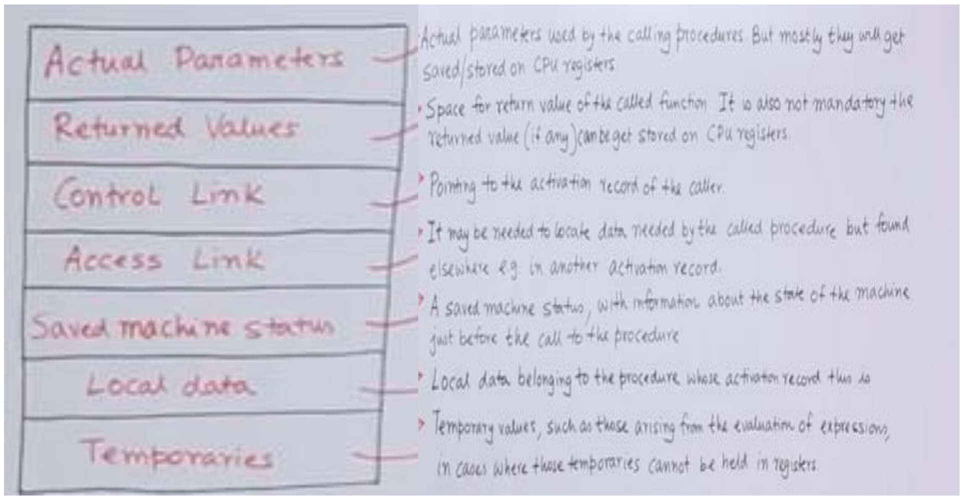
	c	**3-address machine code for**
		i. b=a[i]
		`LD  R1, i          // R1 = i`
		`MUL R1, R1, 8      // R1 = R1 * 8`
		`LD  R2, a(R1)      // R2 = contents(a + contents(R1))`
		`ST  b, R2          // b = R2`
		
		ii. if X < Y goto L
		LD R1, X
		LD R2, Y
		SUB R2, R2, R1
		BLTZ R2, M
			2	**CO4/L6**

Section B

2 a **Grammar productions and semantic rules for translating boolean expressions to intermediate code.** — **6 (1 mark each)** — **CO3/L2**

PRODUCTION	SEMANTIC RULES
$B \rightarrow B_1 \;\|\|\; B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \;\|\|\; label(B_1.false) \;\|\|\; B_2.code$
$B \rightarrow B_1 \;\&\&\; B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \;\|\|\; label(B_1.true) \;\|\|\; B_2.code$
$B \rightarrow \,!\, B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \;\|\|\; E_2.code$ $\|\|\; gen($'if' $E_1.addr$ **rel**$.op$ $E_2.addr$ 'goto' $B.true)$ $\|\|\; gen($'goto' $B.false)$
$B \rightarrow$ **true**	$B.code = gen($'goto' $B.true)$
$B \rightarrow$ **false**	$B.code = gen($'goto' $B.false)$

b **Activation record structure** — **3** — **CO4/L3**

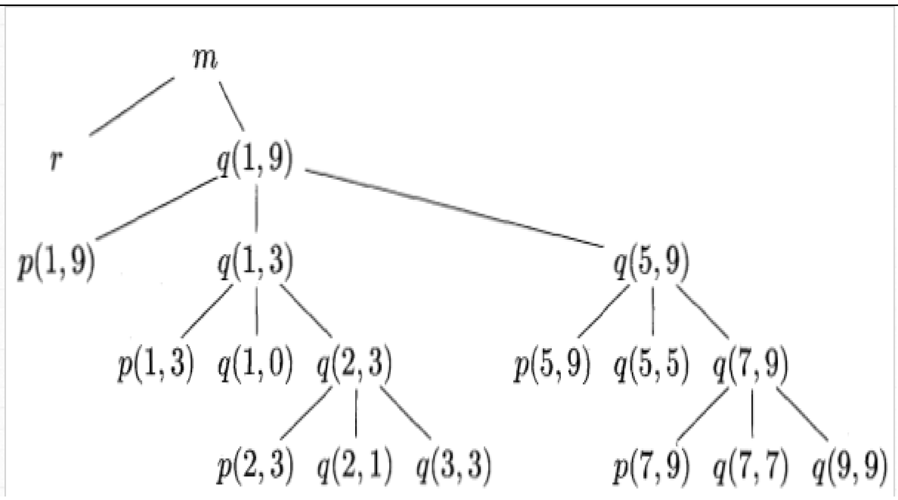
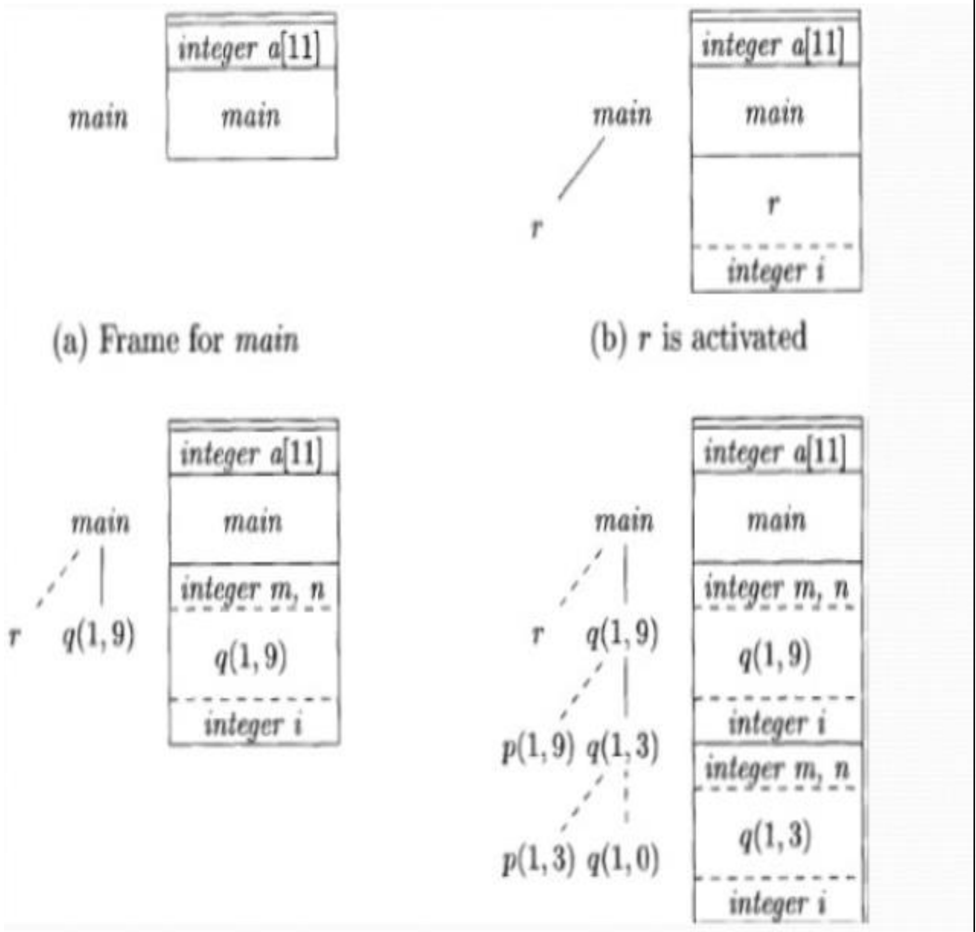

Actual Parameters
Returned Values
Control Link
Access Link
Saved machine status
Local data
Temporaries
Actual parameters used by the calling procedures. But mostly they will get saved/stored on CPU registers
Space for return value of the called function. It is also not mandatory the returned value (if any) can be get stored on CPU registers.
Pointing to the activation record of the caller.
It may be needed to locate data needed by the called procedure but found elsewhere e.g. in another activation record.
A saved machine status, with information about the state of the machine just before the call to the procedure
Local data belonging to the procedure whose activation record this is
Temporary values, such as those arising from the evaluation of expressions, in cases where those temporaries cannot be held in registers.

c **Issues: Instruction Selection** — **3** — **CO4/L5**

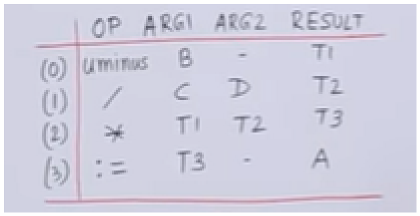
Complexity of mapping IR to target code is determined by:

- **the level of the IR: high level leads to poor code; low level leads to efficient code**
- **the nature of the instruction-set architecture, ex.floating point operations are done using separate registers.**
- **the desired quality of the generated code: speed & size. Redundant load/store, inefficient code (not using INC for increment) affect the efficiency.**

		Every three-address statement of the form x = y + z, where x, y, and z are statically allocated, can be translated into the code sequence **LD R0, y // R0 = y (load y into register R0)** **ADD R0, R0, z // R0 = R0+ z (add z t o R0)** **ST x, R0 // x = R0 (store R0 into x)** **But, a=b+c; d=a+e get translated into a sequence with redundant store in 4th statement** **1. LD R0, b // R0 = b** **2. ADD R0, R0, c // R0 = R0+c** **3. ST a, R0 // a = R0** **4. LD R0, a // R0 = a** **5. ADD R0, R0, e // R0 = R0+e** **6. ST d, R0 // d = R0** **Instruction a=a+1 can be implemented with 1 instruction INC a , rather than by a more obvious sequence that loads a into a register, adds one to the register, and then stores the result back into a.** **7. LD R0, a // R0 = a** **8. ADD R0, R0, #1 // R0 = R0 + 1** **9. ST a, R0 // a = R0**		
3	**a**	**Semantic actions for translation of Array references** $S \rightarrow \textbf{id} = E \; ; \quad \{ \; gen(\; top.get(\textbf{id}.lexeme) \; '=' \; E.addr); \; \}$ $\quad \mid \; L = E \; ; \quad \{ \; gen(L.addr.base \; '[' \; L.addr \; ']' \; '=' \; E.addr); \; \}$ $E \rightarrow E_1 + E_2 \quad \{ \; E.addr = \textbf{new} \; Temp\,();$ $\quad gen(E.addr \; '=' \; E_1.addr \; '+' \; E_2.addr); \; \}$ $\quad \mid \; \textbf{id} \quad \{ \; E.addr = top.get(\textbf{id}.lexeme); \; \}$ $\quad \mid \; L \quad \{ \; E.addr = \textbf{new} \; Temp\,();$ $\quad gen(E.addr \; '=' \; L.array.base \; '[' \; L.addr \; ']'); \; \}$ $L \rightarrow \textbf{id} \; [\; E \;] \quad \{ \; L.array = top.get(\textbf{id}.lexeme);$ $\quad L.type = L.array.type.elem;$ $\quad L.addr = \textbf{new} \; Temp\,();$ $\quad gen(L.addr \; '=' \; E.addr \; '*' \; L.type.width); \; \}$ $\quad \mid \; L_1 \; [\; E \;] \quad \{ \; L.array = L_1.array;$ $\quad L.type = L_1.type.elem;$ $\quad t = \textbf{new} \; Temp\,();$ $\quad L.addr = \textbf{new} \; Temp\,();$ $\quad gen(t \; '=' \; E.addr \; '*' \; L.type.width); \; \}$ $\quad gen(L.addr \; '=' \; L_1.addr \; '+' \; t); \; \}$	**6**	**CO3/L2**
	b	**Activation tree representation for implementing calls during the execution of a [11]**	**6**	**CO4/L3**

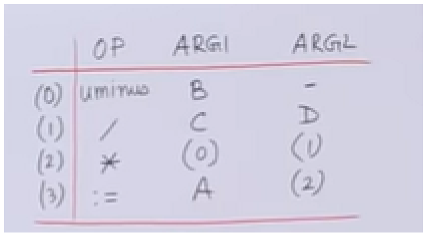
		 m
		r
		q(1,9)
		p(1,9)
		q(1,3)
		q(5,9)
		p(1,3) q(1,0) q(2,3)
		p(5,9) q(5,5) q(7,9)
		p(2,3) q(2,1) q(3,3)
		p(7,9) q(7,7) q(9,9)
		
		The stack growth.
		
		 integer a[11]
		main
		(a) Frame for main
		r
		integer i
		(b) r is activated
		q(1,9)
		integer m, n
		p(1,9) q(1,3)
		p(1,3) q(1,0)
			(3+3)	

4	**a**	**3-address code for the expression A= (-B*(C/D).** **T1 = -B** **T2 = C/D** **T3 = T1 * T2** **A = T3** **Quadruple**	**3**	**CO3/L6**



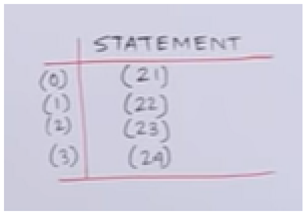
	OP	ARG1	ARG2	RESULT
(0)	uminus	B	-	T1
(1)	/	C	D	T2
(2)	*	T1	T2	T3
(3)	:=	T3	-	A

Triple



	OP	ARG1	ARG2
(0)	uminus	B	-
(1)	/	C	D
(2)	*	(0)	(1)
(3)	:=	A	(2)

Indirect triple



	STATEMENT
(0)	(21)
(1)	(22)
(2)	(23)
(3)	(24)

b **SDT for the control construct if , if-else and while loop with backpatching** — 3 — CO3/L2

1) $S \rightarrow \textbf{if}\ (\ B\ )\ M\ S_1$ $\{\ backpatch(B.truelist,\ M.instr);$
 $S.nextlist\ =\ merge(B.falselist,\ S_1.nextlist);\ \}$

2) $S \rightarrow \textbf{if}\ (\ B\ )\ M_1\ S_1\ N\ \textbf{else}\ M_2\ S_2$
 $\{\ backpatch(B.truelist,\ M_1.instr);$
 $backpatch(B.falselist,\ M_2.instr);$
 $temp\ =\ merge(S_1.nextlist,\ N.nextlist);$
 $S.nextlist\ =\ merge(temp,\ S_2.nextlist);\ \}$

3) $S \rightarrow \textbf{while}\ M_1\ (\ B\ )\ M_2\ S_1$
 $\{\ backpatch(S_1.nextlist,\ M_1.instr);$
 $backpatch(B.truelist,\ M_2.instr);$
 $S.nextlist\ =\ B.falselist;$
 $emit('\texttt{goto}'\ M_1.instr);\ \}$

4) $S \rightarrow \{\ L\ \}$ $\{\ S.nextlist\ =\ L.nextlist;\ \}$

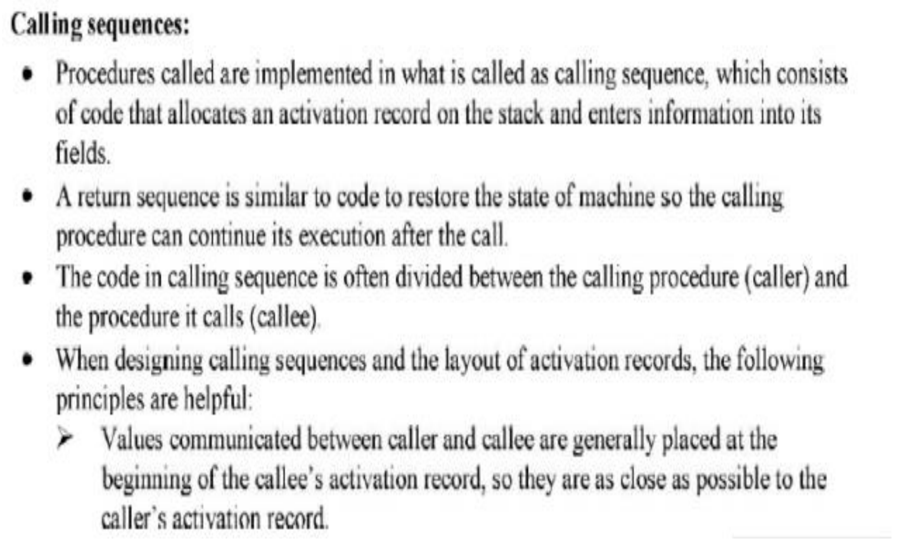
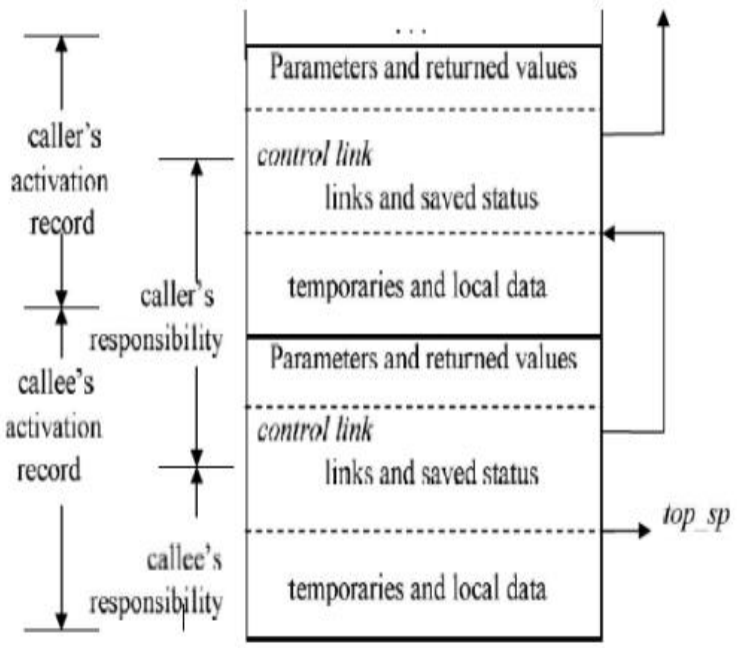
5) $S \rightarrow A\ ;$ $\{\ S.nextlist\ =\ \textbf{null};\ \}$

6) $M \rightarrow \epsilon$ $\{\ M.instr\ =\ nextinstr;\ \}$

7) $N \rightarrow \epsilon$ $\{\ N.nextlist\ =\ makelist(nextinstr);$
 $emit('\texttt{goto \_}');\ \}$

8) $L \rightarrow L_1\ M\ S$ $\{\ backpatch(L_1.nextlist,\ M.instr);$
 $L.nextlist\ =\ S.nextlist;\ \}$

9) $L \rightarrow S$ $\{\ L.nextlist\ =\ S.nextlist;\ \}$

	c	**Division of tasks between a caller and callee**  **Calling sequences:** • Procedures called are implemented in what is called as calling sequence, which consists of code that allocates an activation record on the stack and enters information into its fields. • A return sequence is similar to code to restore the state of machine so the calling procedure can continue its execution after the call. • The code in calling sequence is often divided between the calling procedure (caller) and the procedure it calls (callee). • When designing calling sequences and the layout of activation records, the following principles are helpful: ➢ Values communicated between caller and callee are generally placed at the beginning of the callee's activation record, so they are as close as possible to the caller's activation record.  . . . Parameters and returned values caller's activation record control link links and saved status temporaries and local data caller's responsibility Parameters and returned values callee's activation record control link links and saved status top_sp callee's responsibility temporaries and local data **Division of tasks between caller and callee**	**6**	**CO4/L3**