USN | | | | | | | | | | |

CMR INSTITUTE OF TECHNOLOGY, BENGALURU.

Solution to Internal Assessment Test I – Mar. 2019

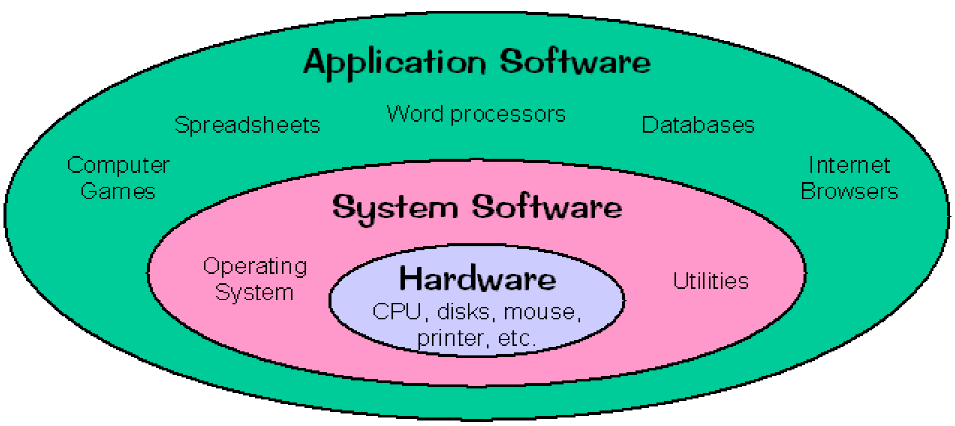
Sub:	System Software & Compiler Design					Sub Code:	15CS63	Branch:	CSE
Date:	6/03/2019	Duration:	90 min's	Max Marks:	50	Sem/Sec:	6/CSE(A,B,C)		OBE

1. a)What is System software? Write the difference between system software and application software

Solution:

- System software consists of a variety of programs that support the operation of a computer.
- The programs implemented in either software and (or) firmware (permanent software programmed into a read-only memory.) that makes the computer hardware usable.
- The software makes it possible for the users to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.
- Bios (basic input output system) It is a type of Firmware used during the booting process (power-on/start up).

Example: Text editor, Compiler, Assembler, Loader, Linker, Debugger and Operating system.



Difference between system software and application software

S.No.	System Software	Application Software
1.	System software is used for operating computer hardware.	Application software is used by user to perform specific task.
2.	System softwares are installed on the computer when operating system is installed.	Application softwares are installed according to user's requirements.
3.	In general, the user does not interact with system software because it works in the background.	In general, the user interacts with application sofwares.
4.	System software can run independently. It provides platform for running application softwares.	Application software can't run independently. They can't run without the presence of system software.
5.	Some examples of system softwares are compiler, assembler, debugger, driver, etc.	Some examples of application softwares are word processor, web browser, media player, etc.

1b)Explain the assembler directives with example.

Solution:

SIC assembler directives with examples

- Pseudo-Instructions
  - Not translated into machine instructions
  - Providing information to the assembler
- Basic assembler directives
  - START
  - END
  - BYTE
  - WORD
  - RESB
  - RESW

- Assembler directives are pseudo instructions
  - They provide instructions to the assembler itself
  - They are not translated into machine operation codes
- SIC assembler directive
  - START : specify name & starting address
  - END : end of source program, specify the first execution instruction
  - BYTE, WORD, RESB, RESW

  - End of record : a null char (00)
  - End of file : a zero-length record

2. Briefly explain SIC/XE machine Architecture.

Solution:-

- Memory-Maximum memory available on a SIC/XE system is 1 megabytes (1024 KB - 2^{20}) in memory
- Register-(3+6) additional registers, 24 bits in length

Mnemonic	Number	Special use
A	0	Accumulator (Used for arithmetic operation)
X	1	Index register(Used for addressing)
L	2	Linkage register (JSUB-jump to subroutine instruction stores the return address in this register)
PC	8	Program counter(Contains address of next instruction to be fetched for execution)
SW	9	Status word (Contains variety of information including condition code)
B	3	Base register; used for addressing
S	4	General working register
T	5	General working register

- 1 additional register, 48 bits in length

i F 6 Floating-point accumulator (48 bits)

Data format

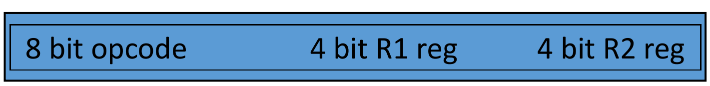
- 24-bit binary number for integer
- 2’s complement for negative values
- 48-bit floating-point data type
- The exponent is between 0 and 2047
- $f*2^{(e-1024)}$
- 0: set all bits to 0

Instruction formats

- Relative addressing - format 3 (e=0)
- Extend the address to 20 bits - format 4 (e=1)
- Don’t refer memory at all - formats 1 and 2
-

1.Format 1 - 8 bit (1 byte)

2.Format 2 – 16 bit (2 bytes)



Example:

- ADDR T,A R2 <- (R2) + (R1)
- 9050

3.Format 3 - 24 bit (3 byte)

6 bit opcoden i x b p e 12 bit displacement

Example:

- SUB N A <- (A) – (N)
- 1F2051
- 0001 1111 0010 0000 0101 0001
- Opcodenixbpe pc rel address

4.Format 4 – 32 bit (4 bytes

6 bit opcoden i x b p e 20 bit address

Example:

- +ADD SEC A <- (A) + (M..M+2)
- 1B100159
- 0001 1011 0001 0000 0000 0001 0101 1001
- Opcodenixbpe address

Addressing modes

- n i x b p e
- Simple n=0, i=0 (SIC) or n=1, i=1
- Immediate n=0, i=1 TA=Values
- Indirect n=1, i=0 TA=(Operand)
- Base relative b=1, p=0 TA=(B)+disp

  0 <= disp<= 4095
- PC relative b=0, p=1 TA=(PC)+disp

  -2048 <= disp<= 2047

Mode	Indication	Target address calculation	
Base relative	b = 1, p = 0	TA = (B) + disp	(0 ≤ disp ≤ 4095)
Program-counter relative	b = 0, p = 1	TA = (PC) + disp	(−2048 ≤ disp ≤ 2047)

Addressing mode

- Direct b=0, p=0 TA=disp
- Index x=1 $TA_{new}=TA_{old}+(X)$
- Index+Base relative x=1, b=1, p=0 TA=(B)+disp+(X)
- Index+PC relative x=1, b=0, p=1 TA=(PC)+disp+(X)
- Index+Direct x=1, b=0, p=0

Format 4 e=1

1.Base Relative Addressing Mode

	n	i	x	b	p	e	
opcode				1	0		disp

b=1, p=0, TA=(B)+disp (0≤disp≤4095)

2.Program-Counter Relative Addressing Mode

	n	i	x	b	p	e	
opcode				0	1		disp

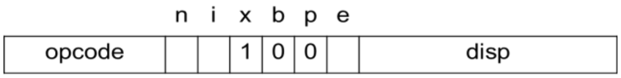
b=0, p=1, TA=(PC)+disp (-2048≤disp≤2047)

3.Direct Addressing Mode

	n	i	x	b	p	e	
opcode				0	0		disp

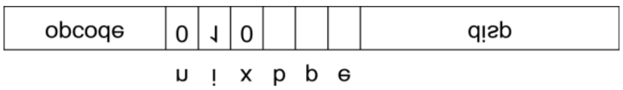
b=0, p=0, TA=disp (0≤disp≤4095)

4.Index Addressing Mode



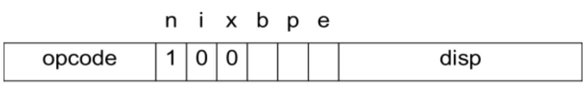
b=0, p=0, TA=(X)+disp

5.Immediate Addressing Mode



n=0, i=1, x=0, operand=disp

6.Indirect Addressing Mode



n=1, i=0, x=0, TA=(disp)

Instruction set

- Format 1, 2, 3, or 4
- Load and store registers (LDB, STB, etc.)
- Floating-point arithmetic operations (ADDF, SUBF, MULF, DIVF)
- Register-to-register arithmetic operations (ADDR, SUBR, MULR, DIVR)
- A special supervisor call instruction (SVC) is provided

I/O

- 1 byte at a time, TD, RD, and WD
- SIO, TIO, and HIO are used to start, test, and halt the operation of I/O channels.

3. Algorithm of pass1 of a two pass assembler.

Pass 1

Pass 1:

```
begin
   read first input line
   if OPCODE = 'START' then
      begin
         save #[OPERAND] as starting address
         initialize LOCCTR to starting address
         write line to intermediate file
         read next input line
      end {if START}
   else
      initialize LOCCTR to 0

   write last line to intermediate file
   save (LOCCTR - starting address) as program length
end {Pass 1}
```

25

```
while OPCODE ≠ 'END' do
   begin
      if this is not a comment line then
         begin
            if there is a symbol in the LABEL field then
               begin
                  search SYMTAB for LABEL
                  if found then
                     set error flag (duplicate symbol)
                  else
                     insert (LABEL,LOCCTR) into SYMTAB
               end {if symbol}
            search OPTAB for OPCODE
            if found then
               add 3 {instruction length} to LOCCTR
            else if OPCODE = 'WORD' then
               add 3 to LOCCTR
            else if OPCODE = 'RESW' then
               add 3 * #[OPERAND] to LOCCTR
            else if OPCODE = 'RESB' then
               add #[OPERAND] to LOCCTR
            else if OPCODE = 'BYTE' then
               begin
                  find length of constant in bytes
                  add length to LOCCTR
               end {if BYTE}
            else
               set error flag (invalid operation code)
         end {if not a comment}
      write line to intermediate file
      read next input line
   end {while not END}
```

4. **Generate object program for given program.**

Given LDX= 04 LDA=00 ADD=18 TIX=2C JLT=38 STA=0C RSUB=4C

Loc	**Length**	**Label**	**Opcode**	**Operand**	**Object code**
4000		SUM	START	4000	
4000	3	FIRST	LDX	ZERO	045788
4003	3		LDA	ZERO	005788
4006	3	LOOP	ADD	TABLE,X	18C015

4009	3		TIX	COUNT	2C5785
400C	3		JLT	LOOP	384006
400F	3		STA	TOTAL	0C578B
4012	3		RSUB		4C0000
4015	1770	TABLE	RESW	2000 (1770)	
5785	3	COUNT	RESW	1	
5788	3	ZERO	WORD	0	
578B	3	TOTAL	RESW	1	
			END	FIRST	

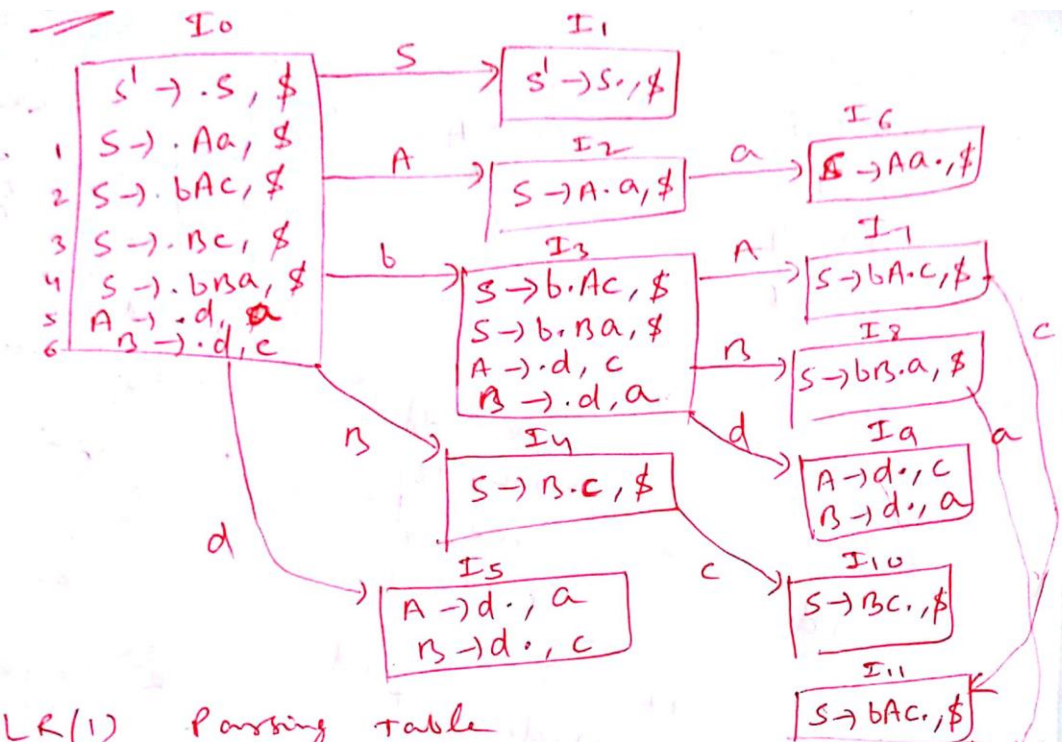
5. **Show that following grammar**

S->Aa |bAc |Bc |bBa

A→d B→d

Is LR(1) but not LALR(1).

Solution:



LR(1) Parsing Table

States	a	b	c	d	$	S	A	B
	ACTION					GOTO		
0		s_3		s_5		1	2	4
1					ACC			
2	s_6							
3				s_9			7	8
4			s_{10}					
5	r_5		r_6					
6					r_1			
7			s_{11}					
8	s_{12}		[illegible]					
9	r_6		r_5					
10					r_3			
11					r_2			
12					r_4			

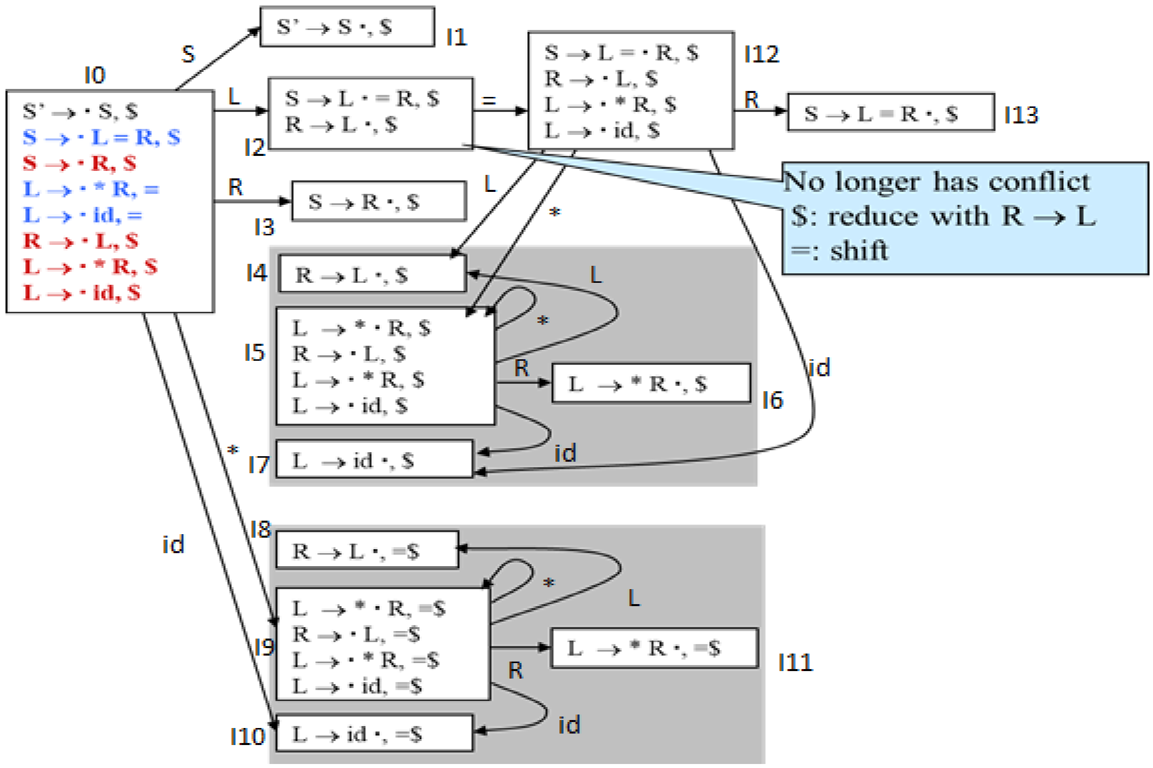
LALR(1) Parsing table

states	a	b	c	d	$	S	A	B
0		s3		s59		1	2	4
1				A	Acc			
2	s6							
3				s59			7	8
4			s10					
59	r5/r6		r5/r6					
6					r1			
7			s11					
8	s12							
10					r3			
11					r2			
12					r4			

In the LALR(1) table there is reduce/reduce conflict. So the grammar is not LALR(1) but LR(1).

.6.Obtain LR(1) items and Construct Canonical LR Parsing Table for the following grammar.

S→L=R | R , L→ *R | id , R→L



1)S→L=R

2)S-> R

3) L→ *R

4)L-> id

5) R→L

LR parsing table

state	*	id	=	$	S	L	R
0	S9	S10			1	2	3
1				accept			
2			S12	r5			
3				r2			
4				r5			
5	S5	S7				4	6

6				r3				
7				r4				
8			r5	r5				
9	S9	S10					8	11
10			r4	r4				
11			r3	r3				
12	S5	S7					4	13
13				r1				

7.Construct SLR parsing table for the following grammar and show the parsing of input string add$

S→CC **C→aC | d**

Solution:

LR(0) items:

I_0: S'→**.S**

S→.CC

C→.aC

C→. d

I_1: Goto(I_0,S)

S'→**S.**

I_2: Goto(I_0,C)

S→C.C

C→.aC

C→. d

I_3: Goto(I_0,a)

C→a.C

C→.aC

C→. d

I_4: Goto(I_0,d)

C→d.

I_5: Goto(I_2,C)

S→CC.

Goto(I_2,a)=I_3 **Goto(**I_2,d)=I_4

I_6: Goto(I_3,a) **Goto**(I_3,a)=I_3 **Goto**(I_3,d)=I_4

C→aC.

SLR Table:

States	a	d	$	S	C
0	S3	S4		1	2
1			Accept		
2	S3	S4			5
3	S3	S4			6
4	R3	R3	R3		
5			R1		
6	R2	R2	R2		

Parsing of the input string add$

Stack	Input	Action
$0	add$	Shift
$0a3	dd$	Shift
$0a3d4	d$	Reduce C→d
$0a3C6	d$	Reduce C→aC
$0C2	d$	shift
$0C2d4	$	Reduce C→d
$0C2C5	$	Reduce S→CC
$0S1	$	Accept

8.Explain handle pruning? Consider the following grammar.
S→ T L;
T→ int | float

L→ L,id | id
Parse the input string int a, b; using shift-reduce parser and draw the bottom-up parse tree.

Solution:

A “handle” of a string is a substring that matches the RHS of a production and whose reduction to the non-terminal (on the LHS of the production) represents one step along the reverse of a rightmost derivation toward reducing to the start symbol.

Handle Pruning

left to right bottom-up parsing constructs a rightmost derivation in reverse

handle = substring that matches the body of a production

handle reduction = a step in the reverse of rightmost derivation

S→ T L;
T→ int | float
L→ L,id | id
Parse the input string int a, b; using shift-reduce parser and draw the bottom-up parse tree.

Right sentential form	handle	Reducing production
Int id1,id2	T	T->int
T id1,id2	L	L->id
T L,id2	L,id2	L->L,id2
D	TL	D->TL

<u>Parse the input string inta,b</u>

Stack	**input**	**action**
$	int id1,id2 ;$	shift
$int	id1,id2; $	Reduce T->int
$T	id1,id2 ;$	shift
$T id1	,id2;$	Reduce L->id
$T L	,id2; $	shift
$T L,	id2; $	shift
$T L,id2	;$	Reduce L->L,id
$T L	;$	shift
$T L;	$	Reduce S->TL;
$S	$	Accept

Draw bottom up parser:-