CMR INSTITUTE OF TECHNOLOGY	USN											

Internal Assesment Test – I Answer Key

Subject : System Software				Code : 16MCA25
Date : 30.03.2017	Duration : 90 mins	Max Marks : 50	Sem : II	Branch : MCA

	Answer Any FIVE FULL Questions	Marks	OBE CO	OBE RBT
1(a)	**Define system software? Differentiate between application software and system software.**	[5]	CO1	L1,L2

System Software consists of a variety of programs that support the operation of a computer. It makes possible for the user to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.

They are usually related to the architecture of the machine on which they are to run.

Example: Assembler, Compiler, text editor, loader and linkers etc.

Comparison between System software and application software

System Software	Application Software
Intended to support the operation and use of the computer	An application program is primarily concerned with the solution of some problem, using the computer as tool
Focus is on the Computer system and not on the application	The focus is on the application not on the computing system.
It depends on the structure of the machine on which it is executed.	It does not depend on the structure of the machine it works
Ex. Operating system, Loader, Linkers, assembler, compiler, text editors etc.	Ex. Banking system, Inventory system.

		Marks	CO	RBT
(b)	**Write short note on VAX Machine Architecture**	[5]	CO1	L1

VAX family of computers was introduce by Digital equipment corporation (DEC) in 1978.

Memory : The VAX memory consists of 8-bit bytes. 2 consecutive bytes form word, 4 consecutive bytes form long word, 8 consecutive bytes form quad ward, and 16 consecutive bytes form an octaword. All VAX programs operate in a virtual address space of 232 bytes.

Registers: There are 16 general purpose registers on the VAX, denoted by Ro to R15, all are 32 bits in length. R15 is program counter, R14 is stack pointer, R13 is frame pointer, R12 is argument pointer, R11to R6 have no special functions and R0 to R5 are available for general use.

Data Formats: Integers are stored as binary numbers in byte, word, longword, quadword or octaword. 2's compliment representation is used for negative values. Characters are stored using their 8-bit ASCII codes. There are four different floating point data formats on the VAX, ranging in length from 4 to 16 bytes.

Instruction Format: VAX machine instruction uses a variable- length instruction format. Each instruction consist of an operation code (1 or 2 bytes) followed by up to six operand specifiers, depending on the type of instruction.

Addressing mode: VAX provide large number of addressing modes. register mode, register deferred mode, autoincrement and autodecrement modes, several base relative addressing modes program-counter relative modes ,indirect addressing mode (called deferred modes) ,immediate operands

Instruction Set : Goal of the VAX designers was to produce an instruction set that is symmetric with respect to data type. The instruction mnemonics are formed by a prefix that specifies the type of operation ,a suffix that specifies the data type of the operands, a modifier that gives the number of operands involved

Input and Output: Input and output on the VAX are accomplished by I/O device controllers Each controller has a set of control/status and data registers, which are assigned locations in the physical address space (called I/O space)

2(a) **Describe SIC Standard Model Instruction Format and Addressing Mode with suitable Examples** [5] CO1 L2

1) Instruction Formats

All machine instructions on the standard version of SIC have the following 24-bit format 8 1 15

opcode	x	address
8	1	15

The flag bit x is used to indicate indexed addressing mode.

2) Addressing Modes

There are two addressing modes, indicated by the setting of the x bit in the instruction.

Mode	**Indication**	**Target address calculation**
Direct	x = 0	TA = address
Indexed	x = 1	TA = address + (x)

	Parentheses are used to indicate the contents of a register or a memory location. For example, (X) represents the contents of register X. **Direct addressing mode** Example LDA TEN | 0000 0000 | 1 | 001 0000 0000 0000 | 0 0 1 0 0 0 Opcode x TEN Effective address (EA) = 1000 Content of the address 1000 is loaded to Accumulator. **Indexed addressing mode** Example STCH BUFFER, X | 0101 0100 | 1 | 001 0000 0000 0000 | 5 4 1 0 0 0 Opcode x BUFFER Effective address (EA) = 1000+[X] = 1000+content of the index register X The Accumulator content, the character is loaded to the effective address.			
(b)	**Write a program for SIC/XE machine to copy a string "Master of Computer Applications" from LOC1 to LOC2** LDT #31 LDX #0 MOVECH LDCH LOC1,X STCH LOC2,X TIXR T JLT MOVECH . . LOC1 BYTE C' Master of Computer Applications'	[5]	CO1	L3

	LOC2 RESB 31			
3	**Describe SIC/XE Instruction Format and Addressing Mode with suitable Examples.**	[10]	CO1	L1

Instruction Formats

- SIC/XE has larger memory hence instruction format of standard SIC version is no longer suitable.
- SIC/XE provide two possible options; using relative addressing (Format 3) and extend the address field to 20 bit (Format 4).
- In addition SIC/XE provides some instructions that do not reference memory at all. (Format 1 and Format 2) .
- The new set of instruction format is as follows. Flag bit e is used to distinguish between format 3 and format 4. (e=0 means format 3, e=1 means format 4)

1. Format 1 (1 byte)

8
op

Example RSUB (return to subroutine)

opcode
0100 1100
4 C

2. Format 2 (2 bytes)

8	4	4
op	r1	r2

Example COMPR A, S (Compare the contents of register A & S)

Opcode	A	S
1010 0000	0000	0100
A 0	0	4

3. Format 3 (3 bytes)

6	1	1	1	1	1	1	12
op	n	i	x	b	p	e	disp

Example LDA #3(Load 3 to Accumlator A)

0000 00	0	1	0	0	0	0	0000 0000 0011
0	n	i	x	b	p	e	0 0 3

4. Format 4 (4 bytes)

6	1	1	1	1	1	1	20
op	n	i	x	b	p	e	address

Example JSUB RDREC(Jump to the address, 1036)

0100 10	1	1	0	0	0	1	0000 0001 0000 0011 0110
	n	i	x	b	p	e	

Addressing Modes

Two new relative addressing modes are available for use with instructions assembled using

	Format 3			

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

b represents for base relative addressing where as p represents program counter relative addressing. If both the bits b and p are 0 then target address is taken form the address field of the instruction (i.e displacement)

SIC/XE also support addressing modes that are assembled using Format 4.

Mode	Indication	Target address calculation
Direct	b=0, p=0, x=0	TA = disp
Indexed	x=1	TA = (x)+disp
Immediate	i=1, n=0	TA = operand value
Indirect	i=0, n=1	TA = address of operand value
simple	i=1, n=1 i=0, n=0	TA = location of the operand value

4(a) **List and explain any five assembler directives with examples.** [5] CO2 L1

In addition to the mnemonic machine instructions assembler uses following assembler directives. These statements are not translated into machine instructions. Instead they provide instructions to assembler itself.

<u>1) START</u>
START specify the name and starting address of the program.
Example: START 1000

<u>2) END</u>
Indicate the end of the source program and (optionally) specify the first executable instruction in the program.
Example: END FIRST

<u>3) BYTE</u>
Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.
Example: BYTE X’F1’

<u>4) WORD</u>
Generate one-word integer constant
Example: THREE WORD 3

<u>5) RESB</u>
Reserve the indicate number of bytes for a data area.
Example: BUFFER RESB 4096

<u>6) RESW</u>
Reserve the indicate number of words for a data area.
Example: LENGTH RESW 1

(b) **List and describe data structures used by two-pass assembler** [5] CO2 L1

The simple assembler uses following internal data structures:

1) Operation Code Table (OPTAB)
2) Symbol Table (SYMTAB).

	3) Location Counter (LOCCTR). 1) OPTAB: It is used to lookup mnemonic operation codes and translates them to their machine language equivalents. In more complex assemblers the table also contains information about instruction format and length In pass 1 the OPTAB is used to look up and validate the operation code in the source program. In pass 2, it is used to translate the operation codes to machine language. OPTAB is usually organized as a hash table, with mnemonic operation code as the key.The hash table organization is particularly appropriate, since it provides fast retrieval with a minimum of searching. Most of the cases the OPTAB is a static table- that is, entries are not normally added to or deleted from it. In such cases it is possible to design a special hashing function or other data structure to give optimum performance for the particular set of keys being stored. 2) SYMTAB: This table includes the name and value for each label in the source program, together with flags to indicate the error conditions (e.g., if a symbol is defined in two different places). During Pass 1: labels are entered into the symbol table along with their assigned address value as they are encountered. All the symbols address value should get resolved at the pass 1. During Pass 2: Symbols used as operands are looked up the symbol table to obtain the address value to be inserted in the assembled instructions. SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval. Since entries are rarely deleted, efficiency of deletion is the important criteria for optimization. Both pass 1 and pass 2 require reading the source program. 3) LOCCTR: Apart from the SYMTAB and OPTAB, this is another important variable which helps in the assignment of the addresses. LOCCTR is initialized to the beginning address mentioned in the START statement of the program. After each statement is processed, the length of the assembled instruction is added to the LOCCTR to make it point to the next instruction.Whenever a label is encountered in an instruction the LOCCTR value gives the address to be associated with that label.			
5	**Write a short note on following machine Independent assembler features.** **i) Literals.** **ii) Symbol-Defining Statements.** **iii) Expressions** **iv) Program Blocks** **v) Control Section** 1) Literals A literal is defined with a prefix = followed by a specification of the literal value. Example: 45 001A ENDFIL LDA =C"EOF" All the literal operands used in a program are gathered together into one or more literal pools. This is usually placed at the end of the program. The assembly listing of a program containing literals usually includes a listing of this literal pool, which shows the assigned addresses and the generated data values. In some cases it is placed at some other location in the object program. An assembler directive LTORG is used. Whenever the LTORG is encountered, it creates a literal pool that contains all the literal operands used	[10]	CO2	L2

since the beginning of the program.

2) Symbol-Defining Statements

1) EQU

Most assemblers provide an assembler directive that allows the programmer to define symbols and specify their values. The directive used for this is EQU (Equate). The general form of the statement is

Symbol EQU value

This statement defines the given symbol (i.e., enter it into SYMTAB) and assigning to it the value specified.

3) Expressions

- Assemblers also allow use of expressions in place of operands in the instruction. Each such expression must be evaluated to generate a single operand value or address.
- Assemblers generally allow arithmetic expressions formed according to the normal rules using arithmetic operators +, - *, /.
- Individual terms in the expression may be constants, user-defined symbols, or special terms.
- The common special term used is * (the current value of location counter) which indicates the value of the next unassigned memory location. Thus the statement BUFFEND EQU *

4) Program blocks

Program block refers to segment of code that are rearranged within a single object program unit and control section to refer to segments that are translated into independent object program units.

Assembler Directive USE indicate which portion of the source program belong to various blocks

USE [blockname]

At the beginning, statements are assumed to be part of the unnamed (default) block.

If no USE statements are included, the entire program belongs to this single block.

Each program block may actually contain several separate segments of the source program. Assemblers rearrange these segments to gather together the pieces of each block and assign address.

Pass1

A separate location counter for each program block is maintained. Save and restore LOCCTR when switching between blocks. At the beginning of a block, LOCCTR is set to 0. Assign each label an address relative to the start of the block. Store the block name or number in the SYMTAB along with the assigned relative address of the label Indicate the block length as the latest value of LOCCTR for each block at the end of Pass1 Assign to each block a starting address in the object program by concatenating the program blocks in

	a particular order Pass2 : Calculate the address for each symbol relative to the start of the object program by adding: The location of the symbol relative to the start of its block.The starting address of this block 5) Control Sections and program linking • A control section is a part of the program that maintains its identity after assembly; each control section can be loaded and relocated independently of the others. • Different control sections are most often used for subroutines or other logical subdivisions. The programmer can assemble, load, and manipulate each of these control sections separately. • Because of this, there should be some means for linking control sections together. For example, instructions in one control section may refer to the data or instructions of other control sections. Since control sections are independently loaded and relocated, the assembler is unable to process these references in the usual way. Such references between different control sections are called external references. • The assembler generates the information about each of the external references that will allow the loader to perform the required linking. When a program is written using multiple control sections, the beginning of each of the control section is indicated by an assembler directive – assembler directive: CSECT The syntax secname CSECT • separate location counter is maintained for each control section Control sections differ from program blocks in that they are handled separately by the assembler.			
6	**Explain in details about Pass1 algorithm for two-pass assembler.**	[10]	CO2	L2

	Assembler 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
	    while OCODE != '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}
	  write last line to intermediate file
	  save (LOCCTR – starting address) as program length
	end {Pass 1}
	``			

7(a)	**Explain about Multi-Pass assembler.** Consider the following example ALPHA EQU BETA BETA EQU DELTA DELTA RESW 1 The symbol BETA cannot be assigned a value when it is encountered during the first pass because DELTA has not yet been defined. As a result, ALPHA cannot be evaluated during second pass. This means that any assembler that makes only two sequential passes over the source program cannot resolve such a sequence of definition.	[5]	CO2	L2

	Prohibiting forward references in symbol definition is not a serious inconvenience. Forward references tend to create difficulty for a person reading the program as well as for the assembler. The general solution is multi pass assembler that can make has many passes are needed to process the definition of symbols. It is not necessary for such an assembler to make more than two passes over the entire program. Instead, the portions of the program that involve forward references in symbol definition are saved during pass. Additional passes through these stored definitions are made as the assembly progresses. There are several ways to accomplish the task outlined above. • Store those symbol definitions that involve forward references in the symbol table. • This table also indicates which symbols are dependent on the values of others, to facilitate symbol evaluation.			
(b)	**Write short note on SPARC Assembler.** Sun OS SPARC assembler Assembler language program is dived into units called sections. Predefine section names .TEXT – Executable instruction .DATA- Initialized read/write data .RODATA- Read only data .BSS – uninitialized data areas Programmer can switch between sections at any times using assembler directives.Separate location counter for each section. When assembler switches to new section it also switches to location counter associated with that	[5]	CO2	L1
8(a)	**Give the target address generated for following machine instruction.** **03C300 h** **if (B)=006000** **010030 h** **(PC)=003000** **0310C303 h** **(x)=000090** <u>03C300 h</u> TA = (x)+(b)+address TA= 000090+006000+300 TA=6390	[6]	CO1	L3

	<u>010030 h</u> TA = address TA= 30 <u>0310C303 h</u> TA = address TA= C303			
(b)	**What is Program Relocation? How relocation is achieved using Modification Record?** It is often desirable to have more than one program at a time sharing the memory and other resources of the machine. In such a situation the actual starting address of the program is not known until the load time. Program in which the address is mentioned during assembling itself. This is called Absolute Assembly or Absolute Program. Since assembler will not know actual location where the program will get loaded, it cannot make the necessary changes in the addresses used by the program. However, the assembler identifies for the loader those parts of the program which need modification. An object program that has the information necessary to perform this kind of modification is called the relocatable program. This can be accomplished with a Modification record having following format: Modification record Col. 1 M Col. 2-7 Starting location of the address field to be modified, relative to the beginning of the program (Hex) Col. 8-9 Length of the address field to be modified, in half-bytes (Hex) One modification record is created for each address to be modified The length is stored in half-bytes. The starting location is the location of the byte containing the leftmost bits of the address field to be modified. If the field contains an odd number of half-bytes, the starting location begins in the middle of the first byte.	[4]	CO2	L1