CS 1203 SYSTEM SOFTWARE
QUESTION BANK - UNIT-I

CS1203 SYSTEM SOFTWARE 3 0 0 100

AIM

To have an understanding of foundations of design of assemblers, loaders, linkers, and macro processors.

OBJECTIVES

- To understand the relationship between system software and machine architecture.
- To know the design and implementation of assemblers
- To know the design and implementation of linkers and loaders.
- To have an understanding of macroprocessors.
- To have an understanding of system software tools.

UNIT I INTRODUCTION 8

System software and machine architecture – The Simplified Instructional Computer (SIC) - Machine architecture - Data and instruction formats - addressing modes - instruction sets - I/O and programming.

UNIT II ASSEMBLERS 10

Basic assembler functions - A simple SIC assembler – Assembler algorithm and data structures - Machine dependent assembler features - Instruction formats and addressing modes – Program relocation - Machine independent assembler features - Literals – Symbol-defining statements – Expressions - One pass assemblers and Multi pass assemblers - Implementation example - MASM assembler.

UNIT III LOADERS AND LINKERS 9

Basic loader functions - Design of an Absolute Loader – A Simple Bootstrap Loader - Machine dependent loader features - Relocation – Program Linking – Algorithm and Data Structures for Linking Loader - Machine-independent loader features - Automatic Library Search – Loader Options - Loader design options - Linkage Editors – Dynamic Linking – Bootstrap Loaders - Implementation example - MSDOS linker.

UNIT IV MACRO PROCESSORS 9

Basic macro processor functions - Macro Definition and Expansion – Macro Processor Algorithm and data structures - Machine-independent macro processor features - Concatenation of Macro Parameters – Generation of Unique Labels – Conditional Macro Expansion – Keyword Macro Parameters-Macro within Macro-Implementation example - MASM Macro Processor – ANSI C Macro language.

UNIT V SYSTEM SOFTWARE TOOLS 9

Text editors - Overview of the Editing Process - User Interface – Editor Structure. - Interactive debugging systems - Debugging functions and capabilities – Relationship with other parts of the system – User-Interface Criteria.

TOTAL : 45

TEXT BOOK

1. Leland L. Beck, “System Software – An Introduction to Systems Programming”, 3rd Edition, Pearson Education Asia, 2000.

REFERENCES

1. D. M. Dhamdhere, “Systems Programming and Operating Systems”, Second Revised Edition, Tata McGraw-Hill, 1999.
2. John J. Donovan “Systems Programming”, Tata McGraw-Hill Edition, 1972.

1) What is system software?

System software consists of variety of programs that supports the operations of a computer. This makes it 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.
Examples of system software are text-editors,compilers,loaders or linkers,debuggers,assemblers,and operating systems.

2) **How system software is different from Software?**
The most important characteristic in which most system software differ from application software is machine dependency.
An application program is primarily concerned with the solution to some problem,using computer as a tool. The focus is on the application,not on the application system.
System programs, on the other hand,are intended to support the operation and use of the computer itself,rather than any particular application. They are usually related to the architecture of the machine on which they are to run.

3) Explain the machine dependency of system software with examples?
An assembler is a system software. It translates mnemonic instructions into machine code; the instruction formats,addressing modes ,etc,are of direct concern in assembler design.
Similarly, compilers must generate machine language code,taking into account such hardware characteristics as the number and the types of registers and machine instructions available
Operating systems are directly concerned with the management of nearly all of the resources of a computer system.

- Example:
- When you took the first programming course
  - **Text editor** - create and modify the program
  - **Compiler**- translate programs into machine language
  - **Loader or linker** - load machine language program into memory
- and prepared for execution
  - **Debugger** - help detect errors in the program
- When you wrote programs in assembler language
  - **Assembler** - translate assembly program into machine language
  - **Macro processor** - translate macros instructions into its definition
- When you control all of these processes
  - By interacting with **the OS**

4) What are the important machine structures used in the design of system software?
- Memory structure
- Registers
- Data formats
- Instruction formats
- Addressing modes
- Instruction set

5) What is SIC machine?
SIC refers to Simplified Instruction Computer which is a hypothetical computer that has been designed to include the hardware features most often found on real machines,while avoiding unusual and irrelevant complexities. This allows to clearly separate the central concepts of a system software from the implementation details associated with a particular machine.

6) Explain SIC Machine architecture.

SIC Machine Architecture

- **Memory**
  - 215 bytes in the computer memory
  - 3 consecutive bytes form a word
  - 8-bit bytes
- **Registers**

Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; JSUB
PC	8	Program counter
SW	9	Status word, including CC

- **Data Formats**
  - Integers are stored as 24-bit binary numbers; 2's complement representation is used for negative values
  - No floating-point hardware
- **Instruction Formats**

opcode (8)	x	address (15)

- **Addressing Modes**

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

- **Instruction Set**
  - **integer arithmetic operations**: ADD, SUB, MUL, DIV, etc.
    - All arithmetic operations involve register A and a word in memory, with the result being left in the register
  - **comparison**: COMP
    - COMP compares the value in register A with a word in memory, this instruction sets a condition code CC to indicate the result
  - **conditional jump instructions**: JLT, JEQ, JGT
    - these instructions test the setting of CC and jump accordingly
  - **subroutine linkage**: JSUB, RSUB
    - JSUB jumps to the subroutine, placing the return address in register L
    - RSUB returns by jumping to the address contained in register L

- **Input and Output**
  - Input and output are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A
  - The Test Device (TD) instruction tests whether the addressed device is ready to send or receive a byte of data
  - Read Data (RD)

- Write Data (WD)

7) What are the contents of status word register?

Bit position	Field name	Use
0	MODE	0=user mode, 1=supervisor mode
1	IDLE	0=running, 1=idle
2~5	ID	Process identifier
6~7	CC	Condition code
8~11	MASK	Interrupt mask
12~15		Unused
16~23	ICODE	Interruption code

8)
9)
10)
11) Explain SIC/XE architecture
SIC/XE Machine Architecture

➢ **Memory**

- Almost the same as that previously described for SIC
- However, 1 MB (2^{20} bytes) maximum memory available

➢ **More Registers**

Mnemonic	Number	Special use
B	3	Base register; used for addressing
S	4	General working register
T	5	General working register
F	6	Floating-point acumulator (48bits)

➢ **Data Formats**
- Floating-point data type: frac*2(exp-1024)
  - frac: 0~1
  - exp: 0~2047

➢ **Instruction Formats**

Format 1		
op(8)		
Format 2		
op(8)	r1(4)	r2(4)

Format 3						e=0	
op(6)	n	I	x	b	p	e	disp(12)

Format 4						e=1	
op(6)	n	I	x	b	p	e	address (20)

- **Instruction Set**
  - new registers: LDB, STB, etc.
  - floating-point arithmetic: ADDF, SUBF, MULF, DIVF
  - register move: RMO
  - register-register arithmetic: ADDR, SUBR, MULR, DIVR
  - supervisor call: SVC
    - generates an interrupt for OS
- **Input/Output**

SIO, TIO, HIO: start, test, halt the operation of I/O device

12) Explain SIC/XE Instruction formats.

- Instruction Formats
- Larger memory means an address cannot fit into a 15-bit field
- Extend addressing capacity
- Use some form of *relative addressing* -> instruction
- format 3
- Extend the address field to *20 bits* -> instruction format 4
- Additional instructions do not reference memory
- Instruction format 1 & 2

13) Explain SIC/XE addressing modes.

- **How to compute TA?**

Addressing modes:

Mode	**Indication**	**Target address calculation**	operand
Base relative	b=1, p=0	TA=(B)+disp (0<=disp<=4095)	(TA)
PC-relative	b=0, p=1	TA=(PC)+disp (-2048<=disp<=2047)	(TA)
Direct	b=0, p=0	TA=disp (format 3) or address (format 4)	(TA)
Indexed	x=1	TA=TA+(X)	(TA)

- **How the target address is used?**

Mode	**Indication**	**operand value**
immediate addressing	i=1, n=0	TA
indirect addressing	i=0, n=1	((TA))
simple addressing	i=0, n=0	SIC instruction (all end with 00)
	i=1, n=1	SIC/XE instruction

- **Addressing modes**
  - **Base relative (n=1, i=1, b=1, p=0)**
  - **Program-counter relative (n=1, i=1, b=0, p=1)**
  - **Direct (n=1, i=1, b=0, p=0)**
  - **Immediate (n=0, i=1, x=0)**
  - **Indirect (n=1, i=0, x=0)**

- **Indexing (both n & i = 0 or 1, x=1)**
- **Extended (e=1 for format 4, e=0 for format 3)**

14) What is direct addressing?

- ***Direct Addressing***
  - The target address is taken directly from the *disp* or *address* field

	n	i	x	b	p	e	
opcode	1	1		0	0		disp/address

Format 3 (e=0): n=1, i=1, ***b=0, p=0***, TA=disp (0≤disp ≤4095)

Format 4 (e=1): n=1, i=1, ***b=0, p=0***, TA=address

15) What is indexed addressing?

- ***Indexed Addressing***
  - The term (X) is added into the target address calculation

	n	i	x	b	p	e	
opcode	1	1	0				disp/address

n=1, i=1, ***x=1***

Ex. *Direct Indexed Addressing*

Format 3, TA=(X)+disp

Format 4, TA=(X)+address

16) What is immediate addressing?

- ***Immediate Addressing – no memory access***

	n	i	x	b	p	e	
opcode	0	1	0				disp/address

n=0, i=1, x=0, **operand**=disp //format 3

n=0, i=1, x=0, **operand**=address //format 4

17) What is indirect addressing?

- ***Indirect Addressing***

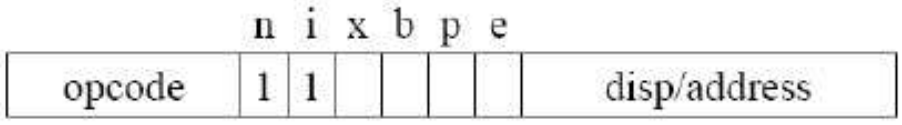
	n	i	x	b	p	e	
opcode	1	0	0				disp/address

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

n=1, i=0, x=0, TA=(address), **operand** = (TA) = ((address))

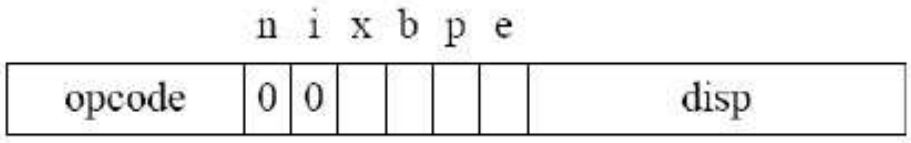
18) What is simple addressing mode?

Simple Addressing Mode



Format 3: *i=1, n=1*, TA=disp, **operand** = (disp)

Format 4: *i=1, n=1*, TA=address, **operand** = (address)



i=0, n=0, TA=b/p/e/disp (SIC standard)

19) What are the instruction set for SIC/XE?

Instruction Set

a. new registers: LDB, STB, etc.
b. floating-point arithmetic: ADDF, SUBF, MULF, DIVF
c. register move: RMO
d. register-register arithmetic: ADDR, SUBR, MULR, DIVR
e. supervisor call: SVC
   - generates an interrupt for OS (Chap 6)

Input/Output

f. SIO, TIO, HIO: start, test, halt the operation of I/O device

20) Give programming examples of SIC.

SIC Programming Examples

-- Data movement

```
ALPHA    RESW    1
FIVE     WORD    5
CHARZ    BYTE C'Z'
C1       RESB 1
         .
         .
         LDA     FIVE
         STA     ALPHA
         LDCH    CHARZ
         STCH    C1
```

(a)

- No memory-memory move instruction
- 3-byte word:
  - LDA, STA, LDL, STL, LDX, STX
- 1-byte:
  - LDCH, STCH
- Storage definition
  - WORD, RESW

- BYTE, RESB

➢ All arithmetic operations are performed using register A, with the result being left in register A.

Arithmetic operation

```
            LDA     ALPHA       LOAD ALPHA INTO REGISTER A
            ADD     INCR        ADD THE VALUE OF INCR
            SUB     ONE         SUBTRACT 1
            STA     BETA        STORE IN BETA
            LDA     GAMMA       LOAD GAMMA INTO REGISTER A
            ADD     INCR        ADD THE VALUE OF INCR
            SUB     ONE         SUBTRACT 1
            STA     DELTA       STORE IN DELTA
             .
             .
             .
ONE         WORD    1           ONE-WORD CONSTANT
.                               ONE-WORD VARIABLES
ALPHA       RESW    1
BETA        RESW    1
GAMMA       RESW    1
DELTA       RESW    1
INCR        RESW    1
```

(a)

Looping and indexing

```
            LDX     ZERO        INITIALIZE INDEX REGISTER TO 0
MOVECH      LDCH    STR1,X      LOAD CHARACTER FROM STR1 INTO REG A
            STCH    STR2,X      STORE CHARACTER INTO STR2
            TIX     ELEVEN      ADD 1 TO INDEX, COMPARE RESULT TO 11
            JLT     MOVECH      LOOP IF INDEX IS LESS THAN 11
             .
             .
             .
STR1        BYTE    C'TEST STRING'   11-BYTE STRING CONSTANT
STR2        RESB    11               11-BYTE VARIABLE
.                                    ONE-WORD CONSTANTS
ZERO        WORD    0
ELEVEN      WORD    11
```

(a)

21) Give an example for SIC IO programming.

```
INLOOP    TD      INDEV       TEST INPUT DEVICE
          JEQ     INLOOP      LOOP UNTIL DEVICE IS READY
          RD      INDEV       READ ONE BYTE INTO REGISTER A
          STCH    DATA        STORE BYTE THAT WAS READ
          .
          .
          .
OUTLP     TD      OUTDEV      TEST OUTPUT DEVICE
          JEQ     OUTLP       LOOP UNTIL DEVICE IS READY
          LDCH    DATA        LOAD DATA BYTE INTO REGISTER A
          WD      OUTDEV      WRITE ONE BYTE TO OUTPUT DEVICE
          .
          .
          .
INDEV     BYTE    X'F1'       INPUT DEVICE NUMBER
OUTDEV    BYTE    X'05'       OUTPUT DEVICE NUMBER
DATA      RESB    1           ONE-BYTE VARIABLE
```

Figure 1.6 Sample input and output operations for SIC.

22) Explain Pentium Pro Architecture.

Pentium Pro Architecture

- **Memory**
- ***physical level***: address are byte addresses
  - word (2 bytes), doubleword (or dword) (4 bytes)
  - Some operations are more efficient when operands are aligned
- ***logical level***: programmers usually view the x86 memory as ***a collection of segments***.
  - An address consists of ***segments*** and ***offsets***
  - In some cases, a segment can also be divided into ***pages***
  - The segment/offset address specified by the programmer is translated into a physical address by the x86 ***MMU (Memory Management Unit)***
- **Registers**
- Eight 32-bit general-purpose registers:
  - EAX, EBX, ECX, EDX: data manipulation
  - ESI, EDI, EBP, ESP: address
- Special-purpose registers:
  - Two 32-bit registers:
- EIP: pointer to the next instruction
- FLAGS: status word
  - Six 16-bit segment registers:
- CS: code segment register
- SS: stack segment register
- DS, ES, FS, and GS: data segments
- Floating-point unit (FPU)
- All above registers are available to application programmers
  - Other registers used only by system programs such as OS.