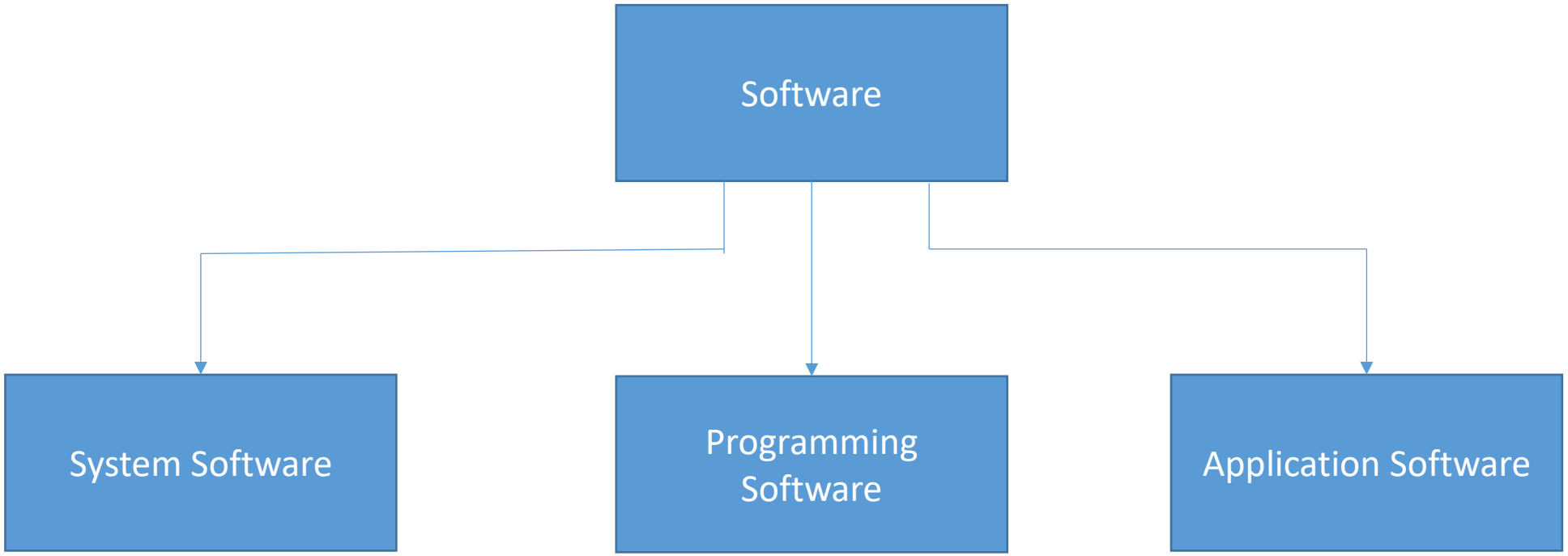
Introduction

Prepared By: Dr. D. P. Singh

Computer Software

- Computer software, or just software, is a collection of computer programs and related data that provides the instructions for telling a computer what to do and how to do it.
- Any set of instructions that guides the hardware and tells it how to accomplish each task.

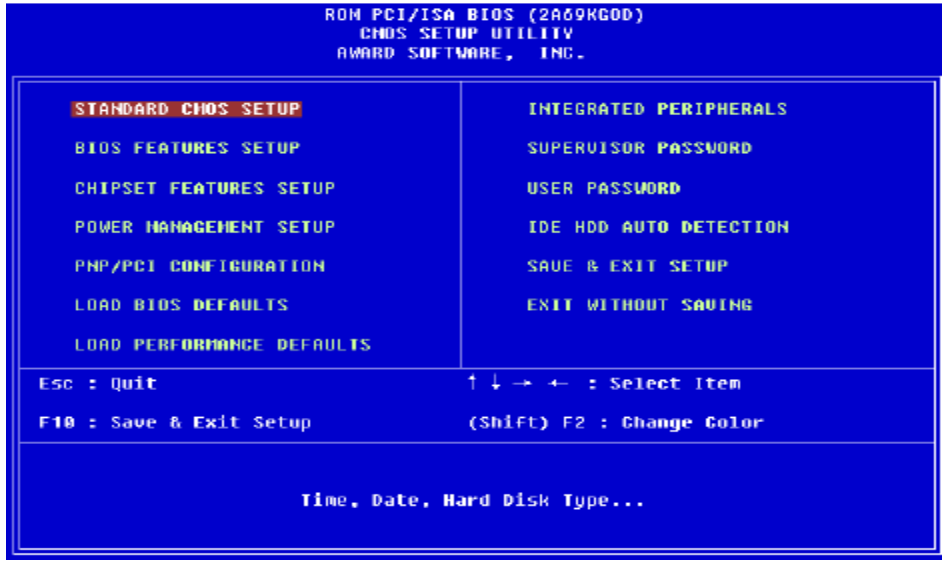
Types of Software



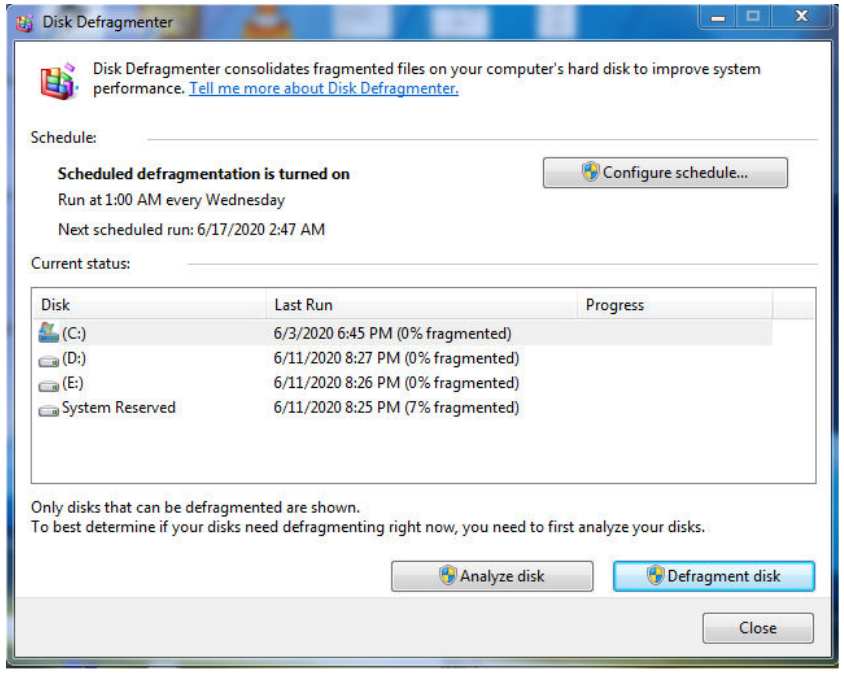
System Software

- System software is computer software designed to operate the computer hardware to provide basic functionality and to provide a platform for running application software.
- System software serves as the **interface between the hardware and the end users.**

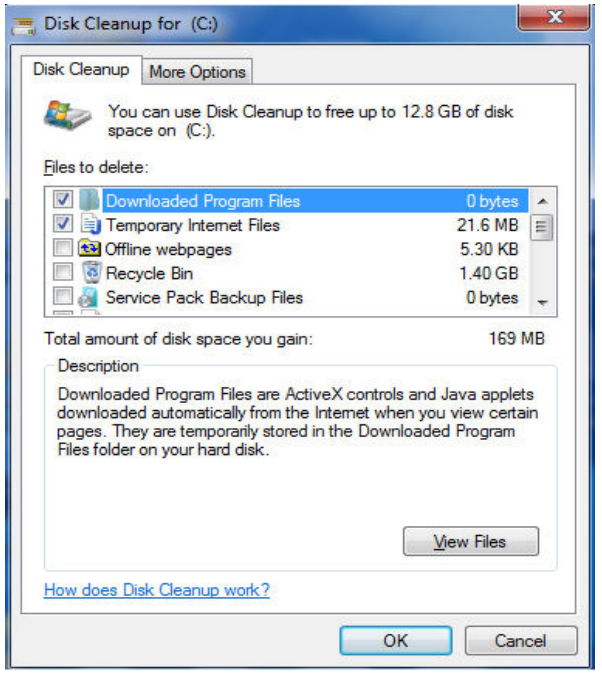
- The BIOS (basic input/output system) gets the computer system started after you turn it on and manages the data flow between the operating system and attached devices such as the hard disk, video adapter, keyboard, mouse, and printer.
- The boot program loads the operating system into the computer's main memory.



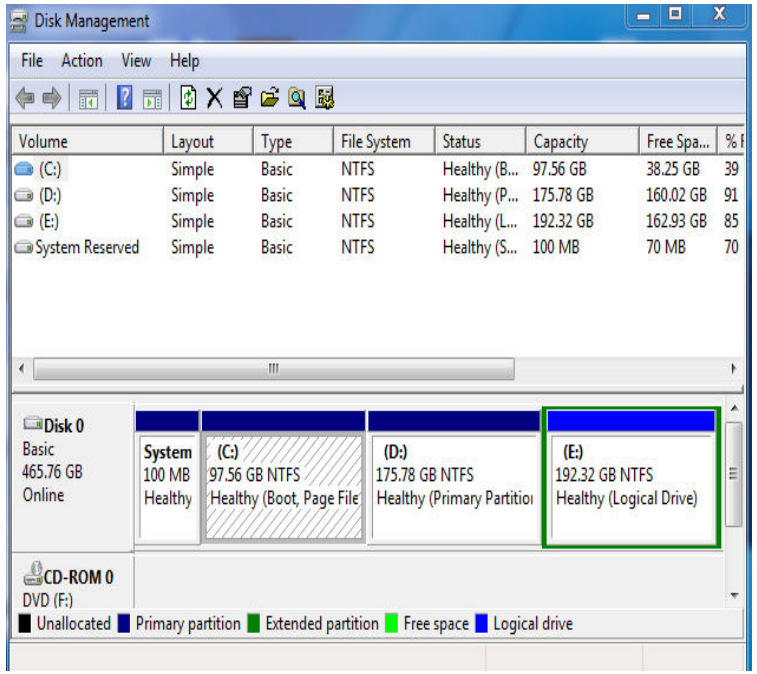
- System software also includes system utilities(Utility Software), such as the disk defragmenter and System Restore.



Disk Defragmenter



Disk Cleanup



Disk Partition Tool

Important Features of System Software

- Closeness to the system
- Fast speed
- Difficult to manipulate
- Written in low level language
- Difficult to design

Programming Software

- Programming software include tools in the form of programs or applications that software developers use to create, debug, maintain, or otherwise support other programs and applications.
- The term usually refers to relatively simple programs such as compilers, debuggers, interpreters, linkers etc.
- E.g. C,C++,JAVA, Python, PHP, Visual Basic etc.

Application Software

- A program or group of programs designed for end users
- Allows end users to accomplish one or more specific (non-computer related) tasks

Examples of Application Software

- Word processor (Microsoft word, WordPro, AppleWorks)
- Spreedsheet (Microsoft Excel, Lotus 1-2-3, Apple-Numbers, Apache OpenOffice-Calc)
- Presentation Software (Microsoft PowerPoint, Adobe Persuasion, Hypercard, OpenOffice Impress, Scala Multimedia)
- Database Management System (Microsoft Access, Oracle Database, MySQL, FoxPro)
- Dekstop Publisher (Adobe Indesign, QuarkXpress, MS Publisher)
  - Used to produce high-quality printed documents such as magazine, greeting card, posters, etc.
- Graphic Editor (Adobe Photoshop, PaintShop Pro, iPhoto, GIMP)
  - Graphics software or image editing software is a program or collection of programs that enable a person to manipulate visual images on a computer.
- Web Browser (Internet Explore (IE),Mozilla Firefox, Opera)

Software Licensing

- Is a legal instrument (by way of contract law) governing the usage or redistribution of software
- Allowing an individual or group to use a piece of software
- Nearly all applications are licensed
- Some are based on the number machines on which the licensed program can run whereas others are based on the number of users that can use the program

Types of Software Licensing

- Registerware
  - Refers to computer software which requires the user to give personal information through registration in order to download or use the program.
- Shareware/ Demoware
  - Refers to copyrighted commercial software that is distributed without payment on a trial basis and is limited by any combination of functionality, availability, or convenience.
- Freeware
  - Computer software that is available for use with no cost or for an optional fee.
  - Freeware is different from shareware, where the user is obliged to pay.

Types of Software Licensing cntd…

- Open source Software(OSS)
  - OSS is also a free software.
  - OSS can be defined as computer software for which the human-readable source code is made available under a copyright license (or arrangement such as the public domain) that meets the Open Source Definition.
  - This permits users to use, change, and improve the software, and to redistribute it in modified or unmodified form.
- Abandonware
  - It refers to sofware that is no longer available for purchase or that is at least a certain amount of years old.

License Key

- A software license key is a pattern of numbers and/or letters provided to licensed users of a software program.

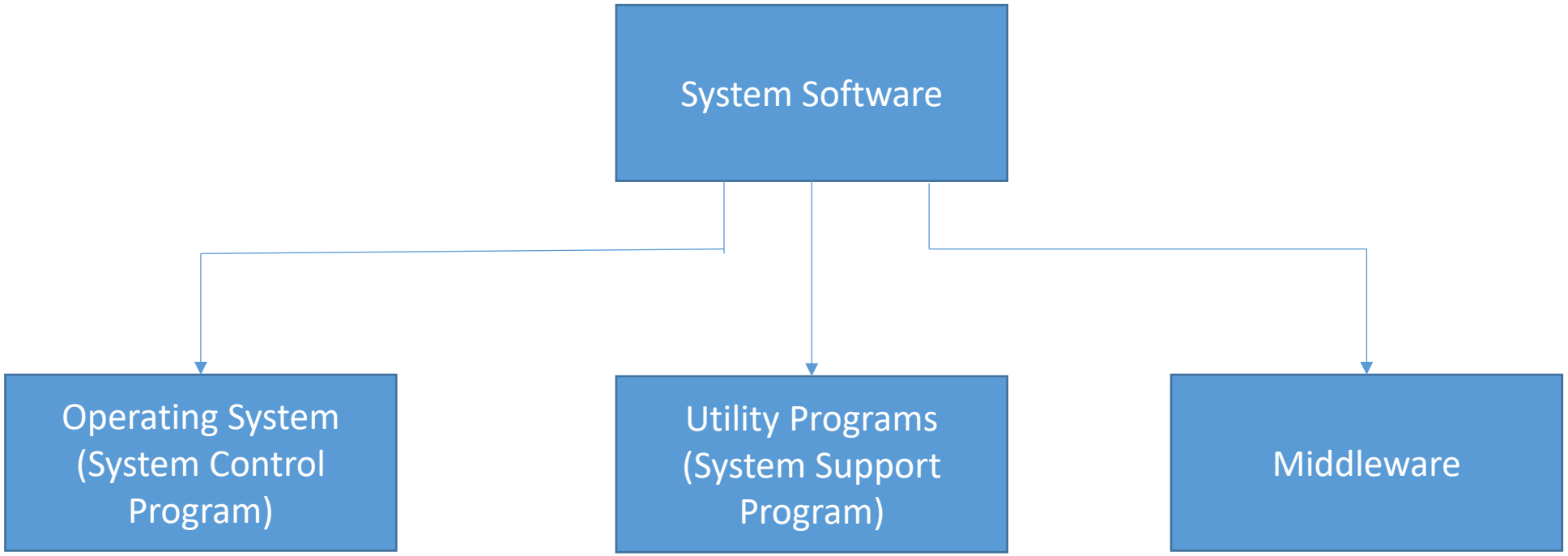
- License keys are typically created and delivered via a license generator once a software user has paid for the software and has agreed to the conditions of use and distribution as legally specified in the software license.

- Thank You

System Software and Machine Dependency

Prepared By: Dr. D. P. Singh

Types of System Software

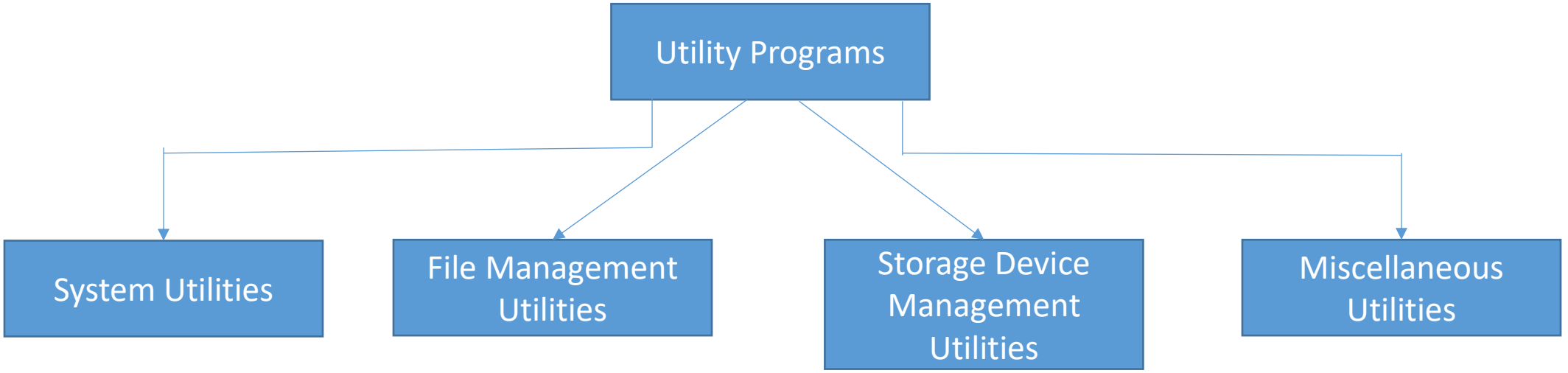


Operating System(OS)

- Is an interface between a computer user and computer hardware.
- Performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.
- E.g. Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Utility Programs

- Utility programs, as the name suggests not only help in executing various crucial tasks for the operating system but also help in overall maintenance of the system.
- Utility Programs can be broadly categorized into FOUR parts



Utility Programs Types

- **System Utilities:**
  - system utility programs are memory manager, antivirus and firewall, registry checker and cleaner, package installer and explorer etc.
- **File Management Utilities:**
  - File management utilities include tools such as data archivers, software backup tools, file compression tools and managers. With the help of these, users can manage their data in the form of files and folders.
- **Storage Device Management Utilities:**
  - These utility programs provide solutions for enhancing disk capacity, such as disk clean-up, partition management, formatting, disk space allocation, defragmentation, etc. With the help of this utility program, users can compartmentalize systems and external drives for efficient management of programs and files that are stored within.
- **Miscellaneous Utilities:**
  - There are various other programs that help in managing business operations. Some of these programs include data generators, HTML checkers and hex editors etc.

Middleware

- Middleware is software that is used to bridge the gap between applications and other tools or databases.
- Middleware sits between an operating system and the applications that run on it.
- It is effectively software that provides a method of communication and data management between applications that would otherwise not have any way to exchange data
- A client can make a request with network-based interactions. That client is typically an application that resides on the front end, which is where the user interacts with software. Resources such as databases, message queues, NoSQL data stores and file servers are often referred to as being part of the back end. Middleware will sit between these ends.

System Software and Machine Architecture

- One characteristic in which most system software differs from applications software is machine dependency.
- System programs are intended to support the operation and use of the computer itself, rather than any particular application. For this reason, they are usually related to the architecture of the machine on which they are to run.
- Because most system software is machine-dependent, we must include real machines and real piece of software in our study. We will present the fundamental functions of each piece of software based on a Simplified Instructional Computer (SIC) – a hypothetical computer.

Simplified Instructional Computer(SIC)

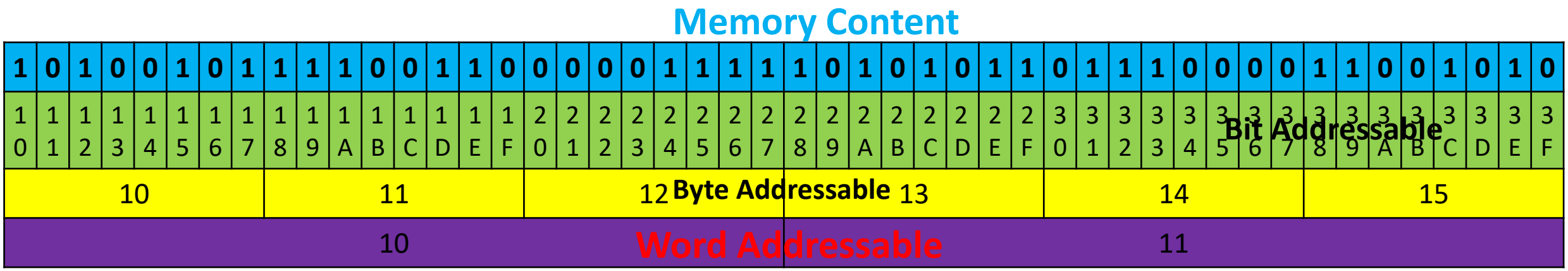
- **Two versions**
  - Standard Model
  - XE version(XE stands for “extra equipment” or “extra expensive”)

SIC Machine Architecture

- **Components**
  - Memory
  - Registers
  - Data Format
  - Instruction Format
  - Instruction Set
  - Input and Output

Memory

- Memory consists of 8-bit bytes.
- Any 3 consecutive bytes form a word (24 bits).
- All addresses on SIC are byte addresses; words are addressed by the location of their lowest numbered byte.
- There are a total of 32,768 (2^{15}) bytes in SIC memory.



Register

a	0
(100)	13

- There are five
  Address of 2nd elem
- Each register i
  Base address(a)+(size
  =100+4*1=104 (su

Bit Position	Field Name	Use
0	MODE	User mode=0, Supervisor Mode=1
1	IDLE	Running=0, idle=1
2-5	ID	Process ID
6-7	CC	Condition Code
8-11	MASK	Interrupt Mask
12-15		Unused
16-23	ICODE	Interruption Code

Mnemonic	Number	Special Use
A	0	Accumulator; use for arithmetic operations
X	1	Index Register; used for addressing
L	2	Linkage Register; jump to subroutine (JSUB) instruction stores the return address in this register
PC	8	Program Counter; contains the address of next instruction to be fetched for execution
SW	9	Status Word; contains a variety of information, including condition code (CC)

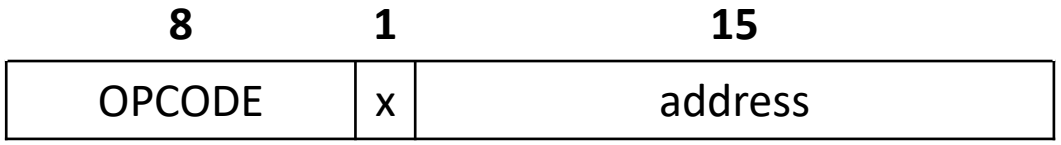
Data Format

- Integers are stored as 24-bit binary numbers.
- Characters are stored in 8-bit ASCII value.

- 2’s complement representation is used for negative values.

- No floating-point hardware on the standard version of SIC.

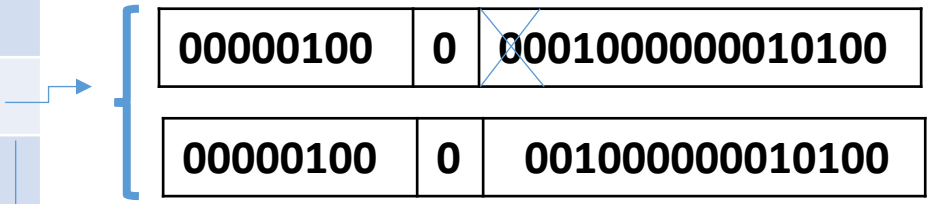
Instruction Formats



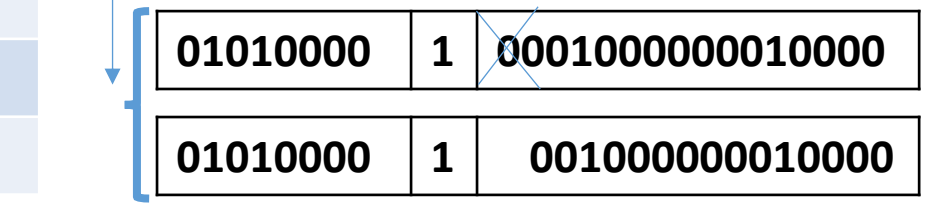
Addressing Modes:
Two addressing modes are available

Mode	Indication	Target Address (TA) Calculation
Direct	x=0	TA=address
Indexed	x=1	TA=address+(X)

Example			
	PROG	START	1000
1000		LDX (04)	ZERO
1003		LDCH (50)	STR1,X
		.	.
1010	STR1	BYTE	C'TEST'
1014	ZERO	WORD	0



041014 (in Hex)



509010 (in Hex)

Instruction Sets

- SIC provides a basic set of instructions that are sufficient for most simple tasks. (See Appendix A)
- Instructions that load and store registers (LDA, LDX, STA, STX, etc.).
- Instructions for integer arithmetic operations (ADD, SUB, MUL, DIV). All arithmetic operations involve register A and a word in memory.
- An instruction (COMP) that compares the value in register A with a word in memory; this instruction sets a condition code CC to indicate the result (<, =, or >).
- Conditional jump instructions (JLT, JEQ, JGT) can test the setting of CC, and jump accordingly.
- JSUB and RSUB are provided to subroutine linkage.

Input and Output

- On the standard version of SIC, input and output are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A.

- Each device is assigned a unique 8-bit code.

- Three I/O instructions: TD (Test Device), RD (Read Data), WD (Write Data).

- Thank You

SIC/XE Machine Architecture

Prepared By: Dr. D. P. Singh

Memory

- The memory structure for SIC/XE is the same as SIC.
- Maximum memory on a SIC/XE is 2^{20} bytes (1 Mbytes).

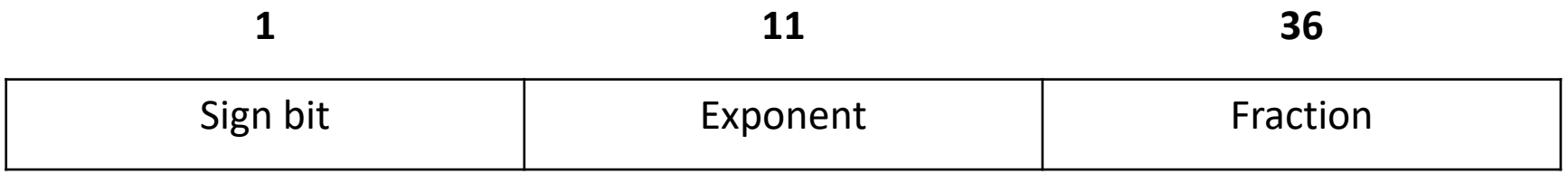
Registers

- Some Additional registers are provided by SIC/XE.
- SIC/XE has total registers (5(SIC)+ 4 more registers)

Mnemonic	Number	Special Use
B	3	Base Register; used for addressing
S	4	General Purpose register; no special use
T	5	General Purpose register; no special use
F	6	Floating point accumulator (48 bits)

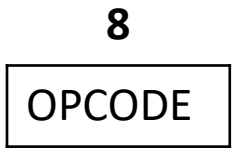
Data Formats

- SIC/XE provides the same data formats as SIC.
- In addition, a 48-bit floating-point data type is also provided.

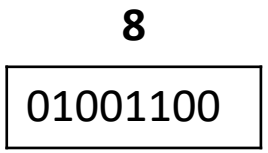


Instruction Formats

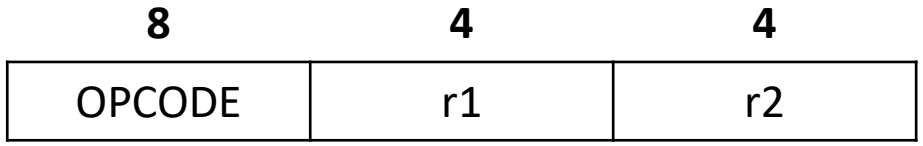
- 4- formats of instructions are available in SIC/XE
  - Format 1 (1 byte)



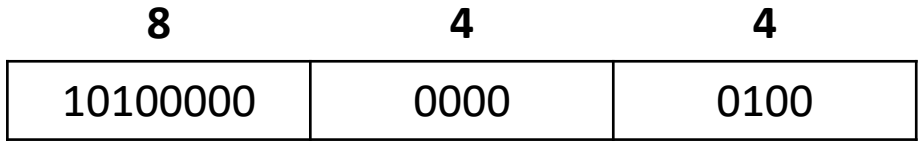
  - E.g. RSUB



  - Format 2 (2 bytes)

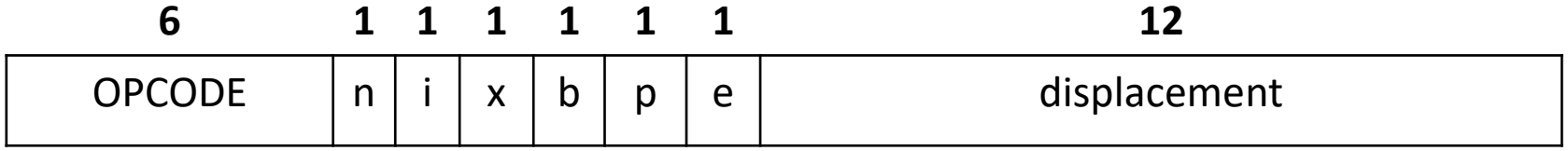


  - E.g. COMPR (A0) A,S

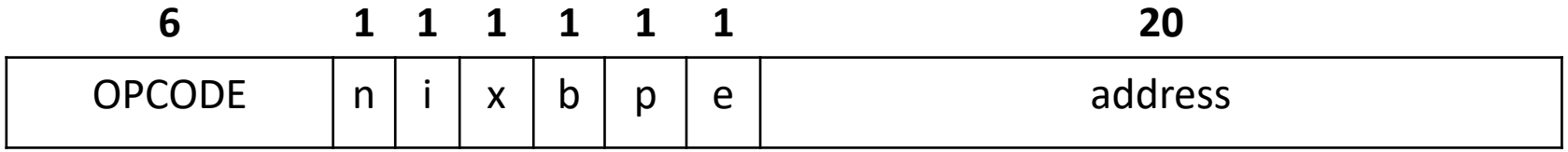


Instruction Formats

- Format 3 (3 bytes)



- Format 4 (4 bytes)



Instruction Formats cntd…

- The settings of the flag bits:
  - Bit *e*: to distinguish between Formats 3 and 4. (*e*=0 Format 3, *e*=1 Format 4)
- Addressing Modes:
  - Two new *relative addressing modes* (Base relative, Program-counter relative) 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 ≤disp≤4095)
Program Counter Relative	b=0, p=1	TA=(PC)+disp (-2048≤disp≤2047)

Instruction Formats cntd…

- *Indexing addressing mode*: if bit *x* is set to 1, the term (X) is added in the target address calculation (Formats 3 and 4).
- Setting of flag n and i

n	i	Addressing mode	explanation
0	0	Simple	the target address is taken as the location of the operand. (Practically not used)
0	1	Immediate	the target address itself is used as the operand value; no memory reference is performed.
1	0	Indirect	The word at the location given by the target address is fetched; the *value* contained in this word is taken as the *address* of the operand value.
1	1	Direct	the target address is taken as the location of the operand.

Examples of FORMAT 3 Instruction

- Base Relative Addressing Example

STX(10) LENGTH

(B)=0033

TA=0033

disp=TA-(B)=0033-0033=0000

Object Code (in HEX)=134000

0001 0000	**1**	**1**	**0**	**1**	**0**	**0**	**0000 0000 0000**

STCH(54) BUFFER,X

(B)=0033

BUFFER=0036

Disp=TA-(B)= 0036-0033=0003

Object code (In HEX)=57C003

0101 0100	**1**	**1**	**1**	**1**	**0**	**0**	**0000 0000 0011**

Examples of FORMAT 3 Instruction

- PC Relative Addressing Example

0000 STL (14) RETADR

0001 01~~00~~	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0010 1011**

TA=0030

Disp=TA-(PC)=0030-0003=~~0~~02D

Object Code (in HEX)=17202D

LDA(00) #9

0000 00~~00~~	**0**	**1**	**0**	**0**	**0**	**0**	**0000 0000 1001**

Object Code (in Hex)=010009

Examples of FORMAT 3 Instruction

002A J (2C) @RETADR

TA=0030

Disp=TA-(PC)=0030-002D=0003

Object Code(in Hex)=2E2003

001011~~00~~	**1**	**0**	**0**	**0**	**1**	**0**	**000000000011**

Example of FORMAT 4 Instruction

+JSUB(48) RDREC

TA=1036

0100 10~~00~~	**1**	**1**	**0**	**0**	**0**	**1**	**0000 0001 0000 0011 0110**

Object Code (in Hex)=4D101036

Instruction Set

- SIC/XE provides all of the instructions available on SIC.
- In addition, there are instructions to
- load and store the new registers (LDB, STB, etc)
- Floating-point arithmetic operations: ADDF, SUBF, MULF, DIVF
- Register-to-register arithmetic operations: ADDR, SUBR, MULR, DIVR.
- Supervisor call instruction: SVC.

Input and Output

- The I/O instructions for SIC are also available in SIC/XE.
- In addition, there are I/O channels that are used to perform the input and output while CPU is executing other instructions.
- This allows overlap of computing and I/O, resulting in more efficient system operations
- SIO, TIO and HIO are used to start, test and halt the operations of I/O channels.

Thank You

Addressing Modes in SIC/XE

Prepared By: Dr. D. P. Singh

Addressing Modes Summary

Addressing Type	n	i	x	b	p	e	Assembly Language Notation	Target Address Calculation
Direct	1	1	0	0	0	0	OP C	disp
	1	1	0	0	0	1	+OP m	Address
	1	1	0	0	1	0	OP m	(PC)+disp
	1	1	0	1	0	0	OP m	(B)+disp
	1	1	1	0	0	0	OP C,X	disp+(X)
	1	1	1	0	0	1	+OP m,X	Address+(X)
	1	1	1	0	1	0	OP m,X	(PC)+disp+(X)
	1	1	1	1	0	0	OP m,X	(B)+disp+(X)
Simple	0	0	0	-	-	-	OP m	b/p/e/disp
	0	0	1	-	-	-	OP m,X	b/p/e/disp+(X)

Addressing Modes Summary cntd…

Addressing Mode	n	i	x	b	p	e	Assembly Language Notation	Target Address Calculation
Indirect	1	0	0	0	0	0	OP @C	disp
	1	0	0	0	0	1	+OP @m	Address
	1	0	0	0	1	0	OP @m	(PC)+disp
	1	0	0	1	0	0	OP @m	(B)+disp
Immediate	0	1	0	0	0	0	OP #C	disp
	0	1	0	0	0	1	+OP #m	Address
	0	1	0	0	1	0	OP #m	(PC)+disp
	0	1	0	1	0	0	OP #m	(B)+disp

Example of addressing modes

Memory Content

	. .
3030	003600
	. .
3600	103000
	. .
6390	00C303
	. .
C303	003030

(B)	006000
(PC)	003000
(X)	000090

Machine Instruction									Target Address	Value Loaded in register A
Hex	Binary									
	OP	n	i	x	b	p	e	Disp/address		
032600	000000	1	1	0	0	1	0	0110 0000 0000	3600	103000
03C300	000000	1	1	1	1	0	0	0011 0000 0000	6390	00C303
022030	000000	1	0	0	0	1	0	0000 0011 0000	3030	103000
010030	000000	0	1	0	0	0	0	0000 0011 0000	30	000030
003600	000000	0	0	0	0	1	1	0110 0000 0000	3600	103000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

Thank You

SIC and SIC/XE Programming Examples

Prepared By: Dr. D. P. Singh

Example 1 (SIC)

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

	BYTE0	**BYTE1**	**BYTE2**
A	00	00	05

	BYTE0	**BYTE1**	**BYTE2**
ALPHA	00	00	05

	BYTE0	**BYTE1**	**BYTE2**
FIVE	00	00	05

	BYTE0
CHARZ	5A

	BYTE0
C1	5A

Example 1 (SIC/XE)

	LDA	**#5**
	STA	**ALPHA**
	LDCH	**#90**
	STCH	**C1**
	.	
	.	
ALPHA	**RESW**	**1**
C1	**RESB**	**1**

	BYTE0	**BYTE1**	**BYTE2**
A	00	00	05

	BYTE0	**BYTE1**	**BYTE2**
ALPHA	00	00	05

BYTE0
C1	5A

Example 2 (SIC)

	LDX	ZERO
MOVECH	LDCH	STR1,X
	STCH	STR2,X
	TIX	ELEVEN
	JLT	MOVECH
	.	
	.	
STR1	BYTE	C’TEST STRING’
STR2	RESB	11
ZERO	WORD	0
ELEVEN	WORD	11

	T	**E**	**S**	**T**		**S**	**T**	**R**	**I**	**N**	**G**
STR1	54	45	53	54	20	53	54	52	49	4E	47

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
STR2	54	45	53	54	20	53	54	52	49	4E	47

	0	**1**	**2**
ZERO	00	00	00

	0	**1**	**2**
X	00	00	[redacted]

	0	**1**	**2**
ELEVEN	00	00	0B

	0	**1**	**2**
A	xx	xx	[redacted]

Example 2 (SIC/XE)

	LDT	#11
	LDX	#0
MOVECH	LDCH	STR1,X
	STCH	STR2,X
	TIXR	T
	JLT	MOVECH
	.	
	.	
STR1	BYTE	C’TEST STRING’
STR2	RESB	11

	T	**E**	**S**	**T**		**S**	**T**	**R**	**I**	**N**	**G**
STR1	54	45	53	54	20	53	54	52	49	4E	47

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
STR2	54	45	53	54	20	53	54	52	49	4E	47

	0	**1**	**2**
ZERO	00	00	00

	0	**1**	**2**
X	00	00	[redacted]

	0	**1**	**2**
T	00	00	0B

	0	**1**	**2**
A	xx	xx	[redacted]

Example 3 (SIC)

	JSUB	READ
	.	
	.	
READ	LDX	ZERO
RLOOP	TD	INDEV
	JEQ	RLOOP
	RD	INDEV
	STCH	RECORD,X
	TIX	K100
	JLT	RLOOP
	RSUB	
	.	
	.	
INDEV	BYTE	X'F1'
RECORD	RESB	100
ZERO	WORD	0
K100	WORD	100

	Byte0	**Byte1**	**Byte2**
X	00	00	00 01 02

	Byte0
INDEV	F1

	Byte0	**Byte1**	**Byte2**	**.**	**.**	**.**	**.**	**.**	**Byte99**
RECORD	54	45	53						

	Byte0	**Byte1**	**Byte2**
ZERO	00	00	00

	Byte0	**Byte1**	**Byte2**
K100	00	00	64

Suppose the characters read by INDEV are TEST…….

Example 3 (SIC/XE)

	JSUB	READ
	.	
	.	
READ	LDX	#0
	LDT	#100
RLOOP	TD	INDEV
	JEQ	RLOOP
	RD	INDEV
	STCH	RECORD,X
	TIXR	T
	JLT	RLOOP
	RSUB	
	.	
	.	
INDEV	BYTE	X'F1'
RECORD	RESB	100

	Byte0	**Byte1**	**Byte2**
X	00	00	00

	Byte0
INDEV	F1

	Byte0	**Byte1**	**Byte2**	**.**	**.**	**.**	**.**	**.**	**Byte99**
RECORD	54	45	53						

	Byte0	**Byte1**	**Byte2**
T	00	00	64

Suppose the characters read by INDEV are TEST…….

Thank You

Assemblers

Prepared By: Dr. D. P. Singh

Basic Assembler Features

- Translating mnemonic language code to its equivalent object code.
- Assigning machine addresses to symbolic labels.

SIC Assembler Directives

- Assembler directives are pseudo instructions
  - They provide instructions to the assembler itself
  - They are not translated into machine operation codes

Assembler Directive	Use
START	Specify name and starting address of the program
END	Indicate the end of the source program and (optionally) specify the first executable instruction in the program
BYTE	Generate Character or hexadecimal constant, occupying as many bytes as needed to represent the constant
WORD	Generate one word integer constant
RESB	Reserve the indicated number of bytes for a data area
RESW	Reserve the indicated number of words for a data area

End of Record: A null char (00)
End of File: a zero length record

Simple SIC Assembler

The translation of source program to object code requires to accomplish the following functions

1. Convert mnemonic operation codes to their machine language equivalents
2. Convert symbolic operands to their equivalent machine addresses
3. Decide the proper instruction format
4. Convert the data constants to internal machine representations
5. Write the object program and the assembly listing

All of these functions except 2 can easily be accomplished by sequential processing of the source program one line at a time

Forward Referencing

	PROG	START	1000
1000		LDA	FIVE
1003		STA	ALPHA
1006		LDCH	CHARZ
1009		STCH	C1
100C	ALPHA	RESW	1
100F	FIVE	WORD	5
1012	CHARZ	BYTE	C'Z'
1013	C1	RESB	1
		END	

Because of Forwarding Referencing most assemblers make two passes over the source program

First pass: scan the source program for label definitions and assign addresses

Second pass: perform actual translation

In addition to translating the instructions of the source program, assembler must process the assembler directives.

Finding Object Codes

- Example 1

	PROG1	START	1000	Object code(in HEX)
1000		LDA (00)	FIVE	00100F
1003		STA (0C)	ALPHA	OC100C
1006		LDCH (50)	CHARZ	501012
1009		STCH (54)	C1	541013
100C	ALPHA	RESW	1	
100F	FIVE	WORD	5	000005
1012	CHARZ	BYTE	C'Z'	5A
1013	C1	RESB	1	
		END		

00000000	0	001000000001111
00001100	0	001000000001100
01010000	0	001000000010010
01010100	0	001000000010011

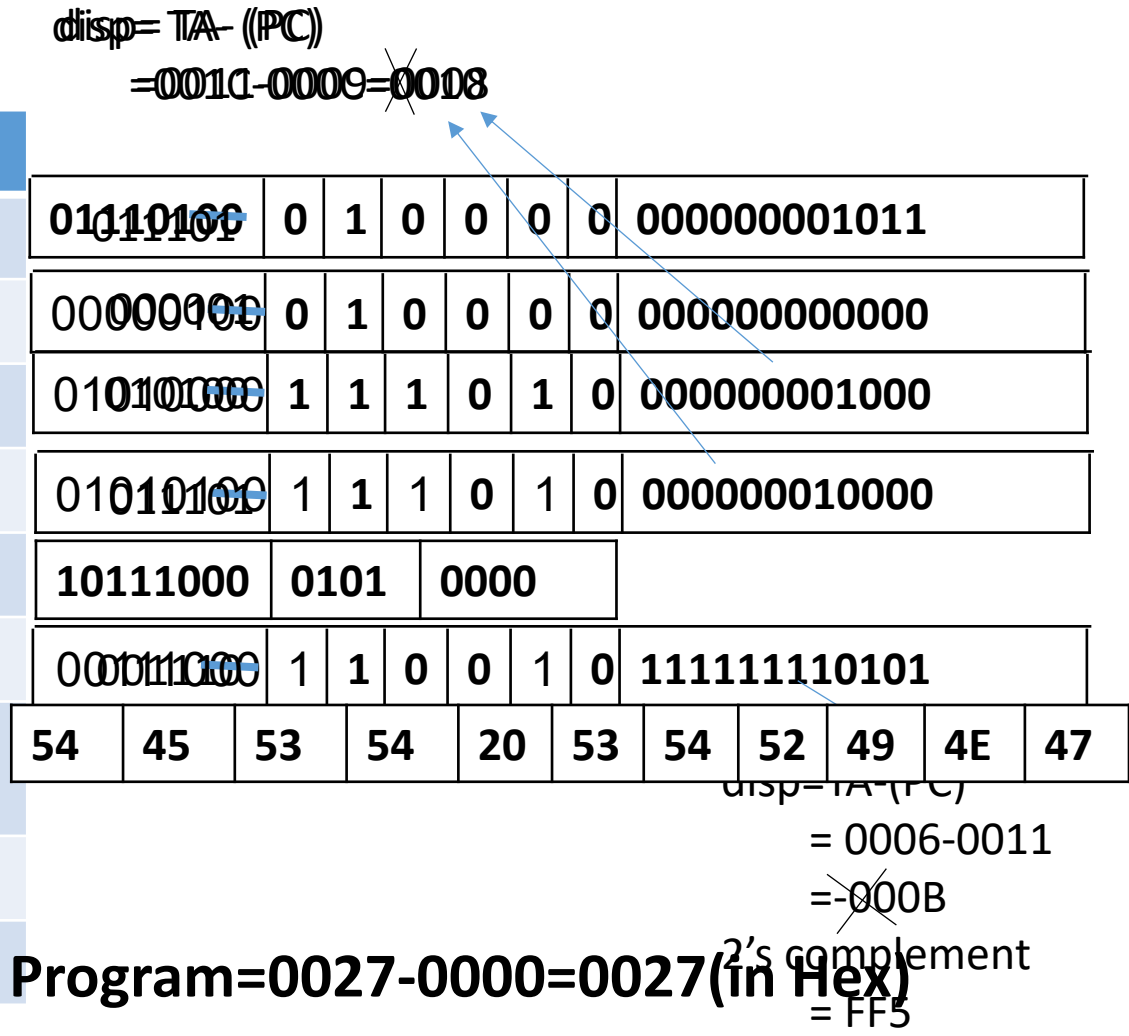
00000000	00000000	00000101
01011010		

Size of the Program=1014-1000=0014(in Hex)

Finding Object Codes cntd…

Example 2

	PROG2	START	0	
0000		LDT(74)	#11	75000B
0003		LDX(04)	#0	050000
0006	MOVECH	LDCH(50)	STR1,X	53A008
0009		STCH(54)	STR2,X	77A010
000C		TIXR(B8)	T	B850
000E		JLT(38)	MOVECH	3B2FF5
0011	STR1	BYTE	C'TEST STRING'	5445535420535452494E47
001C	STR2	RESB	11	
		END		



Size of the Program=0027-0000=0027(in Hex)

Thank You

ELF (Executable and Linking Format)

Prepared By: Dr. D. P. Singh

Types of Object Files

- **Relocatable file: (.o file)**
  - Holds code and data suitable for linking with other object files to create an executable or a shared object file.
- **Executable file: (a.out file)**
  - Holds a program suitable for execution.
- **Shared object file: (.so file)**
  - Holds code and data suitable for linking in two contexts. First, the linkage editor may process it with other relocatable and shared object files to create another object file. Second, the dynamic linker combines it with an executable file and other shared objects to create a process image.

File Format

- Object files participate in program linking (building a program) and program execution (running a program). For convenience and efficiency, the object file format provides parallel views of a file's contents, reflecting the differing needs of these activities.

Linking View
ELF Header
Program Header Table (Optional)
Section 1
.
.
Section n
Section Header Table

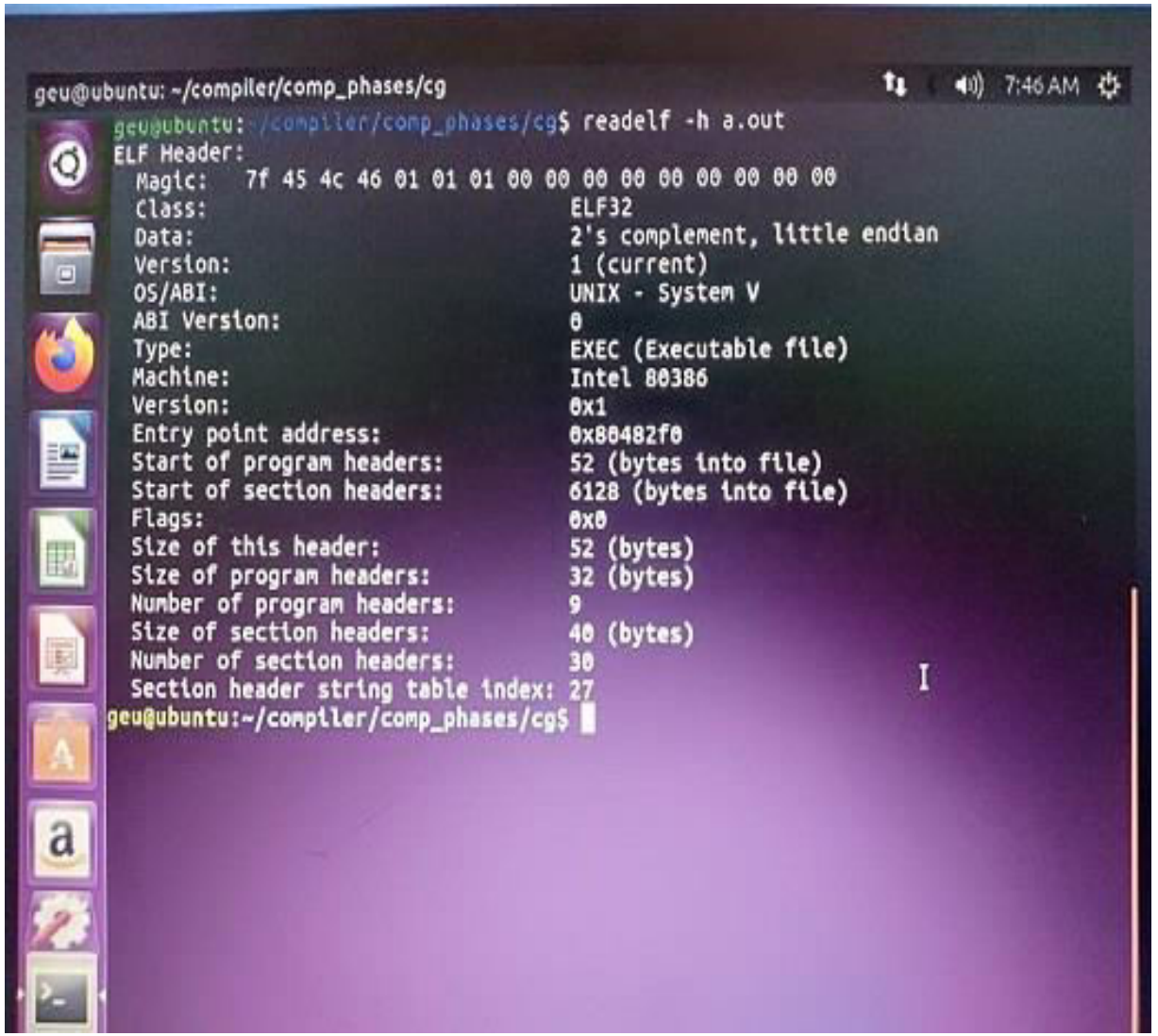
Execution View
ELF Header
Program Header Table
Segment 1
.
.
Segment n
Section Header Table (Optional)

ELF object file format

ELF Header
Program Header Table
.text
.data
.rodata
.bss
.sym
.rel.text
.rel.data
.rel.rodata
.line
.debug
.strtab
Section Header Table

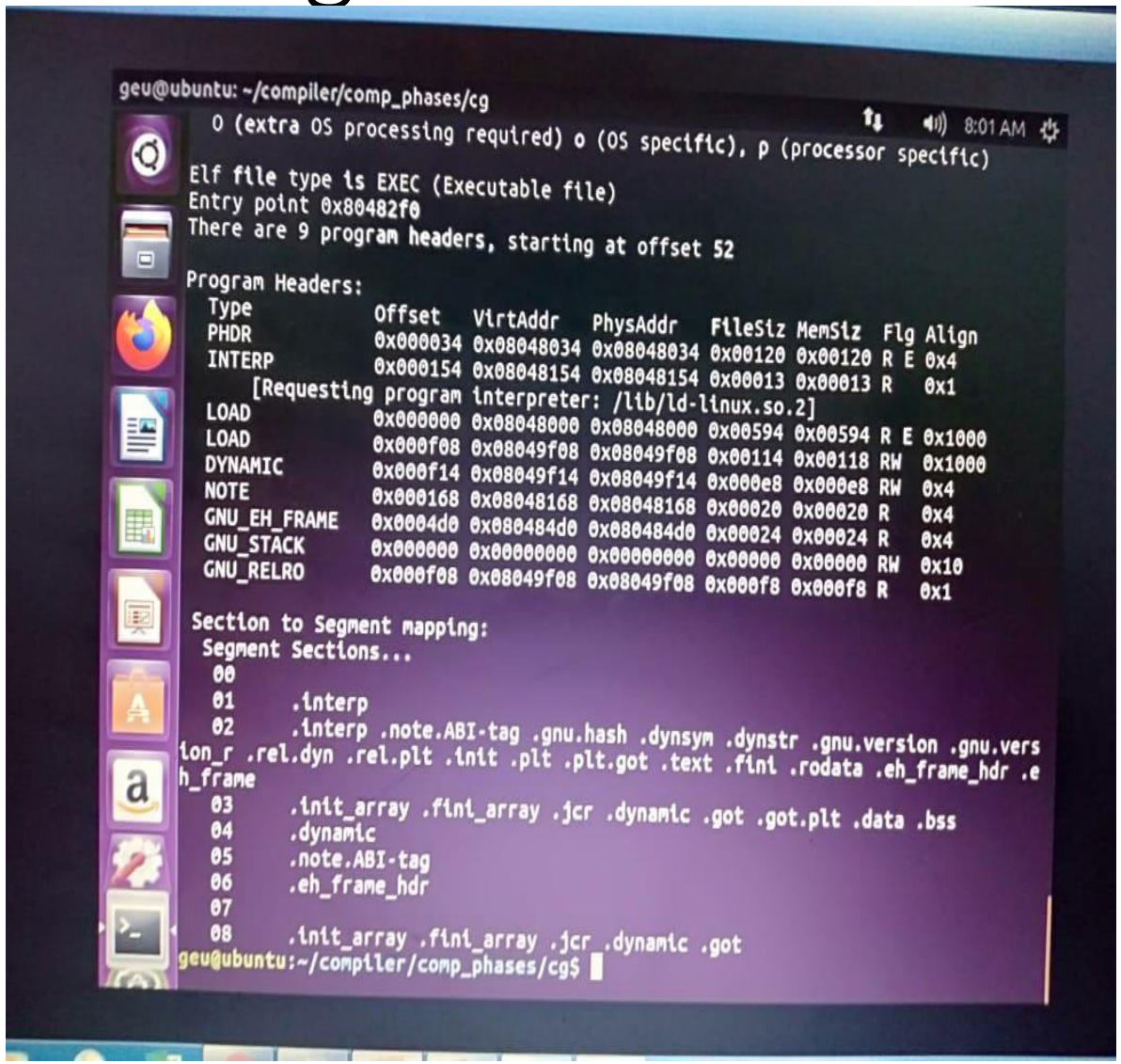
- **Program Header Table**
  - Describing zero or more segments
- **Section Header Table**
  - Describing zero or more sections
- Segments contain the information that is necessary for run time execution of the file
- Sections contain the information for linking and relocation
- Note: Although the figure shows the program header table immediately after the ELF header, and the section header table following the sections, actual files may differ. Moreover, sections and segments have no specified order. Only the ELF header has a fixed position in the file.

ELF Header



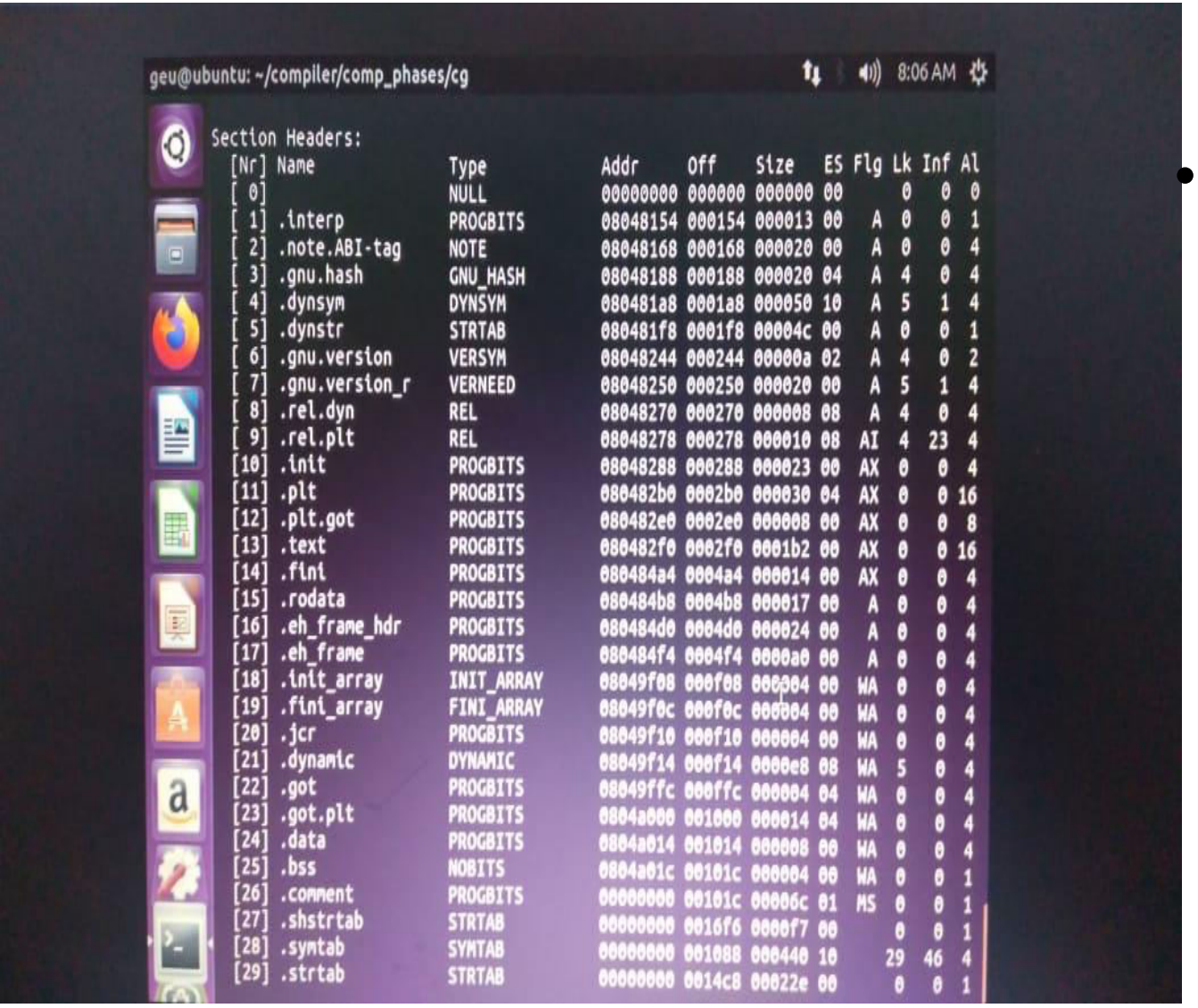
- Magic: This ELF header magic provides information about the file. The first 4 hexadecimal parts define that this is an ELF file (45=**E**,4c=**L**,46=**F**), prefixed with the **7f** value.
- Class: It can a **32-bit** (=01) or **64-bit** (=02) architecture.
- Data: 01 for **LSB** (Least Significant Bit), also known as little-endian. Then there is the value 02, for **MSB** (Most Significant Bit, big-endian).
- Version: next value shows the version

Program Header



- An ELF file consists of zero or more segments, and describe how to create a process/memory image for runtime execution. When the kernel sees these segments, it uses them to map them into virtual address space, using the mmap(2) system call. In other words, it converts predefined instructions into a memory image.
- **GNU_EH_FRAME**
- This is a sorted queue used by the GNU C compiler (gcc). It stores exception handlers. So when something goes wrong, it can use this area to deal correctly with it.
- **GNU_STACK**
- This header is used to store stack information. The stack is a buffer, or scratch place, where items are stored, like local variables.

Section Header



- For executable files there are four main sections: **.text**, **.data**, **.rodata**, and **.bss**. Each of these sections is loaded with different access rights

- **.text:**
  - Contains executable code. It will be packed into a segment with read and execute access rights.
- **.data**
- Initialized data, with read/write access rights
- **.rodata**
- Initialized data, with read access rights only.
- **.bss**
- Uninitialized data, with read/write access rights.

- **.rel.text, .rel.data, .rel.rodata:**
  - These contain the relocation information for the corresponding text or data
- **.symtab:**
  - This section holds the symbol table
- **.strtab:**
  - This section holds the strings.

- **.init:**
  - This section contains the code that is used for process initialization
- **.fini**
  - This section contains the code that is used for process termination

Thank You

Writing Simple Object Program

Prepared By: Dr. D. P. Singh

SIC Program

	COPY	START	1000	OBJECT CODE
1000	FIRST	STL	RETADR	141033
1003	CLOOP	JSUB	RDREC	482039
1006		LDA	LENGTH	001036
1009		COMP	ZERO	281030
100C		JEQ	ENDFIL	301015
100F		JSUB	WRREC	482061
1012		J	CLOOP	3C1003
1015	ENDFIL	LDA	EOF	00102A
1018		STA	BUFFER	0C1039
101B		LDA	THREE	00102D
101E		STA	LENGTH	0C1036
1021		JSUB	WRREC	482061
1024		LDL	RETADR	081033

1027		RSUB		4C0000
102A	EOF	BYTE	C'EOF'	454F46
102D	THREE	WORD	3	000003
1030	ZERO	WORD	0	000000
1033	RETADR	RESW	1	
1036	LENGTH	RESW	1	
1039	BUFFER	RESB	4096	
	.	SUBROUTINE TO READ		
	.	RECORD INTO BUFFER		
2039	RDREC	LDX	ZERO	041030
203C		LDA	ZERO	001030
203F	RLOOP	TD	INPUT	E0205D
2042		JEQ	RLOOP	30203F

2045		RD	INPUT	D8205D
2048		COMP	ZERO	281030
204B		JEQ	EXIT	302057
204E		STCH	BUFFER,X	549039
2051		TIX	MAXLEN	2C205E
2054		JLT	RLOOP	38203F
2057	EXIT	STX	LENGTH	101036
205A		RSUB		4C0000
205D	INPUT	BYTE	X’F1’	F1
205E	MAXLEN	WORD	4096	001000
	.	SUBROUTINE TO WRITE		
	.	RECORD FROM BUFFER		
2061	WRREC	LDX	ZERO	041030

2064	WLOOP	TD	OUTPUT	E02079
2067		JEQ	WLOOP	302064
206A		LDCH	BUFFER,X	509039
206D		WD	OUPUT	DC2079
2070		TIX	LENGTH	2C1036
2073		JLT	WLOOP	382064
2076		RSUB		4C0000
2079	OUTPUT	BYTE	X’05’	05
		END	FIRST	

- Assembler writes the generated object code onto some output device. This object program will later be loaded into memory for execution.
- The simple object program format we use contains three types of records

  - Header Record
  - Text Record
  - End Record

Format of Header Record

Col 1	2 3 4 5 6 7	8 9 10 11 12 13	14 15 16 17 18 19
H	Program Name	Starting address of object program(in Hex)	Length of object program in bytes (in Hex)

Format of Text Record

Col 1	**2 3 4 5 6 7**	**8 9**	**10 11 12 13 14 15 . . . 69**
T	Starting address for object code in this record	Length of object code in this record in bytes (Hex)	Object codes, represented in Hex (2 columns per byte of object code)

Format of End Record

Col 1	2	3	4	5	6	7
E	Address of first executable instruction in object program (in Hex)					

Object Program

2045		RD	INPUT	D8205D
2064	WLOOP	TD	OUTPUT	E02079
2067		JEQ	WLOOP	302064
206A		LDCH	BUFFER,X	509039
206D		WD	OUPUT	DC2079
2070		TIX	LENGTH	2C1036
2073		JLT	WLOOP	382064
2076		RSUB		4C0000
2079	OUTPUT	BYTE	X'05'	05
		END	FIRST	
2061	WRREC	LDX	ZERO	041030

H	C	O	P	Y			0	0	1	0	0	0	0	0	1	0	7	A														
T	0	0	1	0	0	0	1	E	1	4	1	0	3	3	4	8	2	0	3	9	0	0	1	0	3	6	2	8	1	0	3	0
	3	0	1	0	1	5	4	8	2	0	6	1	3	C	1	0	0	3	0	0	1	0	2	A	0	C	1	0	3	9	0	0
	1	0	2	D																												
T	0	0	1	0	1	E	1	5	0	C	1	0	3	6	4	8	2	0	6	1	0	8	1	0	3	3	4	C	0	0	0	0
	4	5	4	F	4	6	0	0	0	0	0	3	0	0	0	0	0	0														
T	0	0	2	0	3	9	1	E	0	4	1	0	3	0	0	0	1	0	3	0	E	0	2	0	5	D	3	0	2	0	3	F
	D	8	2	0	5	D	2	8	1	0	3	0	3	0	2	0	5	7	5	4	9	0	3	9	2	C	2	0	5	E	3	8
	2	0	3	F																												
T	0	0	2	0	5	7	1	C	1	0	1	0	3	6	4	C	0	0	0	0	F	1	0	0	1	0	0	0	0	4	1	0
	3	0	E	0	2	0	7	9	3	0	2	0	6	4	5	0	9	0	3	9	D	C	2	0	7	9	2	C	1	0	3	6
T	0	0	2	0	7	3	0	7	3	8	2	0	6	4	4	C	0	0	0	0	0	5										
E	0	0	1	0	0	0																										

Thank You

Assembler Algorithm and Data Structures

Prepared By: Dr. D. P. Singh

Two Pass SIC Assembler

- Pass 1 (Define symbols)
  - Assign addresses to all statements in the program
  - Save the addresses assigned to all labels for use in Pass 2
  - Perform assembler directives, including those for address assignment, such as BYTE and RESW etc.
- Pass 2 (assemble instructions and generate object program)
  - Assemble instructions (generate opcode and look up addresses)
  - Generate data values defined by BYTE, WORD, etc.
  - Perform processing of assembler directives not done during Pass 1
  - Write the object program and the assembly listing

Assembler Data Structures

- Operation Code Table (OPTAB)
- Symbol Table (SYMTAB)
- Location Counter (LOCCTR)

Location Counter (LOCCTR)

- A variable that is used to help in the assignment of addresses, i.e., LOCCTR gives the address of the associated label.
- LOCCTR is initialized to be the beginning address specified in the START statement.
- After each source statement is processed during pass 1, the length of assembled instruction or data area to be generated is added to LOCCTR.

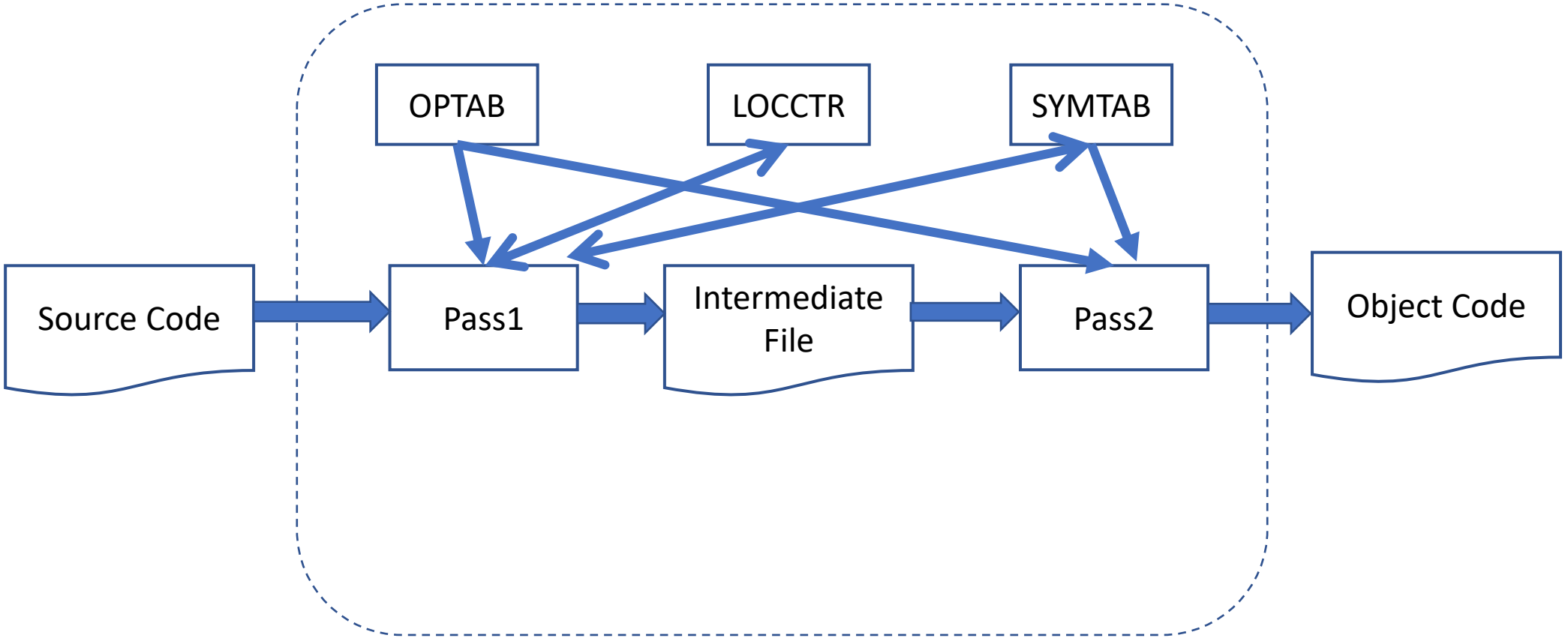
Operation Code Table (OPTAB)

- Contents:
  - Mnemonic operation codes (as the keys)
  - Machine language equivalents
  - Instruction format and length

  Note: SIC/XE has instructions of different lengths
- During pass 1:
  - Validate operation codes
  - Find the instruction length to increase LOCCTR
- During pass 2:
  - Determine the instruction format
  - Translate the operation codes to their machine language equivalents
- Implementation: array or static hash table (entries are not normally added to or deleted from it)
  - Hash table organization is particularly appropriate

Symbol Table (SYMTAB)

- Contents:
  - Label name
  - Label address
  - Flags (to indicate error conditions)
  - Data type or length
- During pass 1:
  - Store label name and assigned address (from LOCCTR) in SYMTAB
- During pass 2:
  - Symbols used as operands are looked up in SYMTAB
- Implementation:
  - a dynamic hash table for efficient insertion and retrieval
  - Should perform well with non-random keys (LOOP1, LOOP2).


OPTAB
LOCCTR
SYMTAB
Source Code
Pass1
Intermediate File
Pass2
Object Code

Pseudo Code: 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
   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
```

	PROG	**START**	**1000**
		LDA	FIVE
		STA	ALPHA
		LDCH	CHARZ
		STCH	C1
	ALPHA	RESW	1
	FIVE	WORD	5
	CHARZ	BYTE	C'Z'
	C1	RESB	1

LOCTR	**1000**

Pseudo Code: Pass 1 cntd…

OPTAB	
LDA	00
STA	0C
LDCH	50
STCH	54
.	.

LOCTR	1000

SYMTAB	
ALPHA	100C
FIVE	100F
CHARZ	1012
C1	1013

```
            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}
```

	PROG	START	1000
		LDA	FIVE
		STA	ALPHA
		LDCH	CHARZ
		STCH	C1
	ALPHA	RESW	1
	FIVE	WORD	5
	CHARZ	BYTE	C’Z’
	C1	RESB	1

Pseudo Code: Pass 2

Pass 2:

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
      begin
         write listing line
         read next input line
      end {if START}
   write Header record to object program
   initialize first Text record
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               search OPTAB for OPCODE
               if found then
                  begin
                     if there is a symbol in OPERAND field then
                        begin
                           search SYMTAB for OPERAND
                           if found then
                              store symbol value as operand address
                           else
                              begin
                                 store 0 as operand address
                                 set error flag (undefined symbol)
                              end
                        end {if symbol}
```

SYMTAB	
ALPHA	100C
FIVE	100F
CHARZ	1012
C1	1013

OPTAB	
LDA	00
STA	0C
LDCH	50
STCH	54
.	.

HPROG 001000000014

T001000

	PROG	START	1000
		LDA	FIVE
		STA	ALPHA
		LDCH	CHARZ
		STCH	C1
	ALPHA	RESW	1
	FIVE	WORD	5
	CHARZ	BYTE	C'Z'
	C1	RESB	1
		END	

Pseudo Code: Pass 2 cntd…

T001000 00100F

```
                else
                    store 0 as operand address
                assemble the object code instruction
            end {if opcode found}
        else if OPCODE = 'BYTE' or 'WORD' then
            convert constant to object code
        if object code will not fit into the current Text record then
            begin
                write Text record to object program
                initialize new Text record
            end
        add object code to Text record
    end {if not comment}
    write listing line
    read next input line
end {while not END}
write last Text record to object program
write End record to object program
write last listing line
end {Pass 2}
```

	PROG	START	1000
		LDA	FIVE
		STA	ALPHA
		LDCH	CHARZ
		STCH	C1
	ALPHA	RESW	1
	FIVE	WORD	5
	CHARZ	BYTE	C’Z’
	C1	RESB	1
		END	

Thank You

Instruction Formats and Addressing Modes in SIC/XE

Prepared By: Dr. D. P. Singh

Sample Program

disp=TA-(PC)
=0030-0006-0003
=0030-0009
=0027- 0014
2'S COMPLEMENT
=FEC

	COPY	START	0	
0000	FIRST	STL(14)	RETADR	17202D
0003		LDB(68)	#LENGTH	69202D
		BASE	LENGTH	
0006	CLOOP	+JSUB(48)	RDREC	4B101036
000A		LDA(00)	LENGTH	032026
000D		COMP(28)	#0	290000
0010		JEQ(30)	ENDFIL	332007
0013		+JSUB(48)	WRREC	4B10105D
0017		J(3C)	CLOOP	3F2FEC

0001 01	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0010 1101**
0110 10	**0**	**1**	**0**	**0**	**1**	**0**	**0000 0010 1101**
0100 10	**1**	**1**	**0**	**0**	**0**	**1**	**0000 0001 0000 0011 0110**
0000 00	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0010 0110**
0010 10	**0**	**1**	**0**	**0**	**0**	**0**	**0000 0000 0000**
0011 00	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0000 0111**
0100 10	**1**	**1**	**0**	**0**	**0**	**1**	**0000 0001 0000 0101 1101**
0011 11	**1**	**1**	**0**	**0**	**1**	**0**	**1111 1110 1100**

SYMTAB	
FIRST	0000
CLOOP	0006
ENDFIL	001A
EOF	002D
RETADR	0030
LENGTH	0033
BUFFER	0036
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WRREC	105D
WLOOP	1062
OUTPUT	1076

Assembler directive BASE is used to inform the assembler about the content of base register

Sample Program cntd…

disp≡TA-(PC)

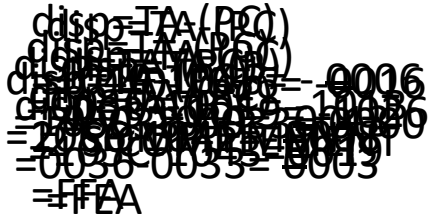
001A	ENDFIL	LDA(00)	EOF	032010
001D		STA(0C)	BUFFER	0F2016
0020		LDA(00)	#3	010003
0023		STA(0C)	LENGTH	0F200D
0026		+JSUB(48)	WRREC	4B10105D
002A		J (3C)	@RETADR	3E2003
002D	EOF	BYTE	C’EOF’	454F46
0030	RETADR	RESW	1	
0033	LENGTH	RESW	1	
0036	BUFFER	RESB	4096	
1036	RDREC	CLEAR(B4)	X	B410
1038		CLEAR(B4)	A	B400
103A		CLEAR(B4)	S	B440

0000 00	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0001 0000**
0000 11	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0001 0110**
0000 00	**0**	**1**	**0**	**0**	**0**	**0**	**0000 0000 0011**
0000 11	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0000 1101**
0100 10	**1**	**1**	**0**	**0**	**0**	**1**	**0000 0001 0000 0101 1101**
0011 11	**1**	**0**	**0**	**0**	**1**	**0**	**0000 0000 0011**

0100 0101 0100 1111 0100 0110

1011 0100	**0001**	**0000**
1011 0100	**0000**	**0000**
1011 0100	**0100**	**0000**

SYMTAB	
FIRST	0000
CLOOP	0006
ENDFIL	001A
EOF	002D
RETADR	0030
LENGTH	0033
BUFFER	0036
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WRREC	105D
WLOOP	1062
OUTPUT	1076

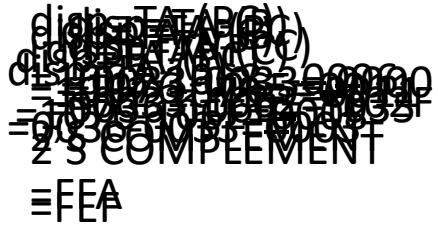
Sample Program  cntd…

103C		+LDT(74)	#4096	75101000
1040	RLOOP	TD(E0)	INPUT	E32019
1043		JEQ(30)	RLOOP	332FFA
1046		RD(D8)	INPUT	DB2013
1049		COMPR(A0)	A,S	A004
104B		JEQ(30)	EXIT	332008
104E		STCH(54)	BUFFER,X	57C003
1051		TIXR(B8)	T	B850
1053		JLT(38)	RLOOP	3B2FEA
1056	EXIT	STX(10)	LENGTH	134000
1059		RSUB(4C)		4F0000
105C	INPUT	BYTE	X’F1’	F1

0111 01	**0**	**1**	**0**	**0**	**0**	**1**	**0000 0001 0000 0000 0000**
1110 00	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0001 1001**
0011 00	**1**	**1**	**0**	**0**	**1**	**0**	**1111 1111 1010**
1101 10	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0001 0011**
1010 0000	**0000**	**0100**					
0011 00	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0000 1000**
0101 01	**1**	**1**	**1**	**1**	**0**	**0**	**0000 0000 0011**
1011 1000	**0101**	**0000**					
0011 10	**1**	**1**	**0**	**0**	**1**	**0**	**1111 1110 1010**
0001 00	**1**	**1**	**0**	**1**	**0**	**0**	**0000 0000 0000**
0100 11	**1**	**1**	**0**	**0**	**0**	**0**	**0000 0000 0000**
1111 0001							

SYMTAB	
FIRST	0000
CLOOP	0006
ENDFIL	001A
EOF	002D
RETADR	0030
LENGTH	0033
BUFFER	0036
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WRREC	105D
WLOOP	1062
OUTPUT	1076

Sample Program cntd…



105D	WRREC	CLEAR(B4)	X	B410
105F		LDT(74)	LENGTH	774000
1062	WLOOP	TD(E0)	OUTPUT	E32011
1065		JEQ(30)	WLOOP	332FFA
1068		LDCH(50)	BUFFER,X	53C003
106B		WD(DC)	OUTPUT	DF2008
106E		TIXR(B8)	T	B850
1070		JLT(38)	WLOOP	3B2FEF
1073		RSUB(4C)		4F0000
1076	OUTPUT	BYTE	X'05'	05
		END	FIRST	

1011 0100	**0001**	**0100**

0111 01	**1**	**1**	**0**	**1**	**0**	**0**	**0000 0000 0000**
1110 00	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0001 0001**
0011 00	**1**	**1**	**0**	**0**	**1**	**0**	**1111 1111 1010**
0101 00	**1**	**1**	**1**	**1**	**0**	**0**	**0000 0000 0011**
1101 11	**1**	**1**	**0**	**0**	**1**	**0**	**0000 0000 1000**

1011 1000	**0101**	**0000**

0011 10	**1**	**1**	**0**	**0**	**1**	**0**	**1111 1110 1111**
0100 11	**1**	**1**	**0**	**0**	**0**	**0**	**0000 0000 0000**

0000 0101

SYMTAB	
FIRST	0000
CLOOP	0006
ENDFIL	001A
EOF	002D
RETADR	0030
LENGTH	0033
BUFFER	0036
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WRREC	105D
WLOOP	1062
OUTPUT	1076

Thank You

Program Relocation

Prepared By: Dr. D. P. Singh

Overview

- Due to the larger main memory of SIC/XE, several programs can be loaded and run at the same time.
- This kind of sharing machine among programs is known as multiprogramming.
- It results in more productive use of hardware.
- If we knew in advance exactly which programs were to be executed concurrently, we could assign the addresses when the programs were assembled so that they would fit together without overlap or wasted space.
- However it is not practical most of time

Overview cntd…

- We usually do not know exactly when jobs will be submitted, how long they will run etc.
- Because of this, it is desirable to be able to load a program into memory wherever there is a room for it.
- In this situation the actual starting address of the program is not known until the load time.

Absolute Program or Absolute Assembly

	PROG	START	1000	
1000		LDA	FIVE	00100F
1003		STA	ALPHA	0C100C
1006		LDCH	CHARZ	501012
1009		STCH	C1	541013
100C	ALPHA	RESW	1	
100F	FIVE	WORD	5	000005
1012	CHARZ	BYTE	C'Z'	5A
1013	C1	RESB	1	
		END		

Since assembler does not know the actual location where the program will be loaded, it cannot make the necessary changes in the addresses used by the program, however, the assembler can identify for the loader those parts of the object program that needs modification.

An object program that contains the information necessary to perform this kind of modification is called a **relocatable program.**

Loaded at 1000

1000	1001	1002	1003	1004	1005
00	10	0F	0C	10	0C
1006	1007	1008	1009	100A	100B
50	10	12	54	10	13
100C	100D	100E	100F	1010	1011
ALPHA			00	00	05
1012	1013				
5A	C1				

Loaded at 2000

2000	2001	2002	2003	2004	2005
00	10	0F	0C	10	0C
2006	2007	2008	2009	200A	200B
50	10	12	54	10	13
200C	200D	200E	200F	2010	2011
ALPHA			00	00	05
2012	2013				
5A	C1				

Program Relocation (SIC/XE Program)

	COPY	START	0	
0000	FIRST	STL(14)	RETADR	17202D
.				
.				
0006	CLOOP	+JSUB(48)	RDREC	4B101036
000A		LDA(00)	LENGTH	032026
.				
0030	RETADR	RESW	1	
0033	LENGTH	RESW	1	
.				
.				
1036	RDREC	CLEAR(B4)	X	B410
.				

Loaded at 5000

5000	17202D	(STL RETADR)
.		
5006	4B101036	(+JSUB RDREC)
500A	032026	(LDA LENGTH)
.		
5030		←RETADR
5033		← LENGTH
.		
6036	B410	(CLEAR X) ← RDREC
.		

Note that no matter where the program is loaded, RDREC is always 1036 bytes past the starting address of the program.

0100 10	**1011 0100**	**0001**	**0000**	**011 0110**

TA=01036
(RDREC)
RDREC

Solution of Relocation Problem

- When the assembler generates the object code for the JSUB instruction which is considered, it will insert the address of RDREC relative to the start of the program.(this is reason, LOCCTR is initialized 0 for the assembly)
- The assembler will also produce a command for the loader, instructing it to add the beginning address of the program to the address field in the JSUB instruction at load time.
- Command for loader is part of the object program (written in modification record)

Thank You

Modification Record

Prepared By: Dr. D. P. Singh

Rules for Writing the Modification Record

Col 1	2	3	4	5	6	7	8	9
M	Starting Location of the address field to be modified, relative to the beginning of the program (in Hex)						Length of the address field to be modified, in half bytes(in Hex)	

Writing Relocatable Object Program

103C		+LDT(74)	#4096	75101000
105D	WRREC	CLEAR(B4)	X	B410
105F		LDT(74)	LENGTH	774000
1062	WLOOP	TD(E0)	OUTPUT	E32011
1065		JEQ(30)	WLOOP	332FFA
1068		LDCH(50)	BUFFER,X	53C003
106B		WD(DC)	OUTPUT	DF2008
106E		TIXR(B8)	T	B850
1070		JLT(38)	WLOOP	3B2FEF
1073		RSUB(4C)		4F0000
1076	OUTPUT	BYTE	X'05'	05
		END	FIRST	
103A		CLEAR(B4)	S	B440

H COPY 000000 001077

T 000000 1D 17202D 69202D 4B101036 032026 290000 332007 4B10105D 3F2FEC 032010

T 00001D 13 0F2016 010003 0F200D 4B10105D 3E2003 454F46

T 001036 1D B410 B400 B440 75101000 E32019 332FFA DB2013 A004 332008 57C003 B850

T 001053 1D 3B2FEA 134000 4F0000 F1 B410 774000 E32011 332FFA 53C003 DF2008 B850

T 001070 07 3B2FEF 4F0000 05

M 000007 05

M 000014 05

M 000027 05

E 000000

Thank You

Machine Independent Assembler Features

Prepared By: Dr. D. P. Singh

Outline

- Literals
- Symbol Defining Statements
- Expressions
- Program Blocks
- Control Sections and Program Linking

Literals

- Design idea
  - Let programmers to be able to write the value of a constant operand as a part of the instruction that uses it. Such an operand is called literal because the value is stated “literally” in the instruction.
  - This avoids having to define the constant elsewhere in the program and make up a label for it.
- Example

	001A	ENDFIL	LDA	=C’EOF’	032010
			LTORG		
	002D	*	=C’EOF’		454F46
	1062	WLOOP	TD	=X’05’	E32011

Literals vs Immediate Operands

- Immediate Operands
  - The operand value is assembled as part of the machine instruction

```
0020                    LDA     #3      010003
```

- Literals
  - The assembler generates the specified value as a constant at some other memory location

```
001A    ENDFIL  LDA     =C'EOF'         032010
```

- Compare

```
001A    ENDFIL          LDA             EOF             032010
002D    EOF             BYTE            C'EOF'          454F46
```

Literal Implementation

- Literal pools
  - Normally literals are placed into a pool at the end of the program
  - In some cases, it is desirable to place literals into a pool at some other location in the object program
    - Assembler Directive: LTORG
    - reason: keep the literal operand close to the instruction

Literal Implementation cntd…

- Duplicate literals
  - 1062 WLOOP TD =X’05’
  - 106B WD =X’05’
  - The assemblers should recognize duplicate literals and store only one copy of the specified data value
    - Comparison of the defining expression
      - Same literal name with different value, e.g. LOCCTR =*
    - Comparison of the generated data value
      - The benefits of using generate data value are usually not great enough to justify the additional complexity in the assembler

Literal Implementation cntd…

- LITTAB
  - literal name, the operand value and length, the address assigned to the operand
- Pass 1
  - build LITTAB with literal name, operand value and length, leaving the address unassigned
  - when LTORG statement is encountered, assign an address to each literal not yet assigned an address
- Pass 2
  - search LITTAB for each literal operand encountered
  - generate data values using BYTE or WORD statements
  - generate modification record for literals that represent an address in the program

Thank You

Machine Independent Assembler Features

Prepared By: Dr. D. P. Singh

Outline

- Literals
- Symbol Defining Statements
- Expressions
- Program Blocks
- Control Sections and Program Linking

Expressions

- Assemblers generally allow arithmetic expressions formed according to the normal rules using the operators +, -, *, /.
- Individual terms in the expression may be constants, user defined symbols, or special terms.
- Expressions can be classified as absolute expressions or relative expressions

  MAXLEN EQU BUFEND-BUFFER

  - BUFEND and BUFFER both are relative terms, representing addresses within the program
  - However the expression BUFEND-BUFFER represents an absolute value
- When relative terms are paired with opposite signs, the dependency on the program starting address is canceled out; the result is an absolute value

SYMTAB

- A relative expression is one in which all of the relative terms except one can be paired. The remaining unpaired relative term must have a positive sign.
- None of the relative terms may enter into a multiplication or division operation
- Expressions that do not meet the conditions given for either absolute or relative expressions should be flagged by the assembler as errors.
- Errors:
  - BUFEND+BUFFER
  - 100-BUFFER
  - 3*BUFFER
- The type of an expression
  - keep track of the types of all symbols defined in the program

Symbol	Type	Value
RETADR	R	30
BUFFER	R	36
BUFEND	R	1036
MAXLEN	A	1000

Thank You

Machine Independent Assembler Features

Prepared By: Dr. D. P. Singh

Outline

- Literals
- Symbol Defining Statements
- Expressions
- Program Blocks
- Control Sections and Program Linking

Program Blocks

- The source program logically contained subroutines, data areas etc. however, they were handled by the assembler as one entity, resulting in a single block of object program.
- Within this object program the generated machine instructions and data appeared in the same order as they were written in the source program.
- Many assemblers provide the features that allow the generated machine instructions and data to be appear in the object program in a different order from the corresponding source statements.
- Program Blocks refer to segments of code that are rearranged within a single object program unit

Program Blocks cntd…

- **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

Program Blocks- Implementation

- Pass 1
  - each program block has a separate location counter
  - each label is assigned an address that is relative to the start of the block that contains it
  - at the end of Pass 1, the latest value of the location counter for each block indicates the length of that block
  - the assembler can then assign to each block a starting address in the object program
- Pass 2
  - The address of each symbol can be computed by adding the assigned block starting address and the relative address of the symbol to that block

Example

LOC/	BLOCK	Source Statements			Object Code
LOC/	BLOCK	Source Statements			Object Code
0060	0		JLT	WLOOP	
0063	0		RSUB		
0007	1		USE	CDATA	
			LTORG		
0007	1	*	=C’EOF’		
000A	1	*	=X’05’		
			END	FIRST	
0058	0		LDCH	BUFFER,X	
005B	0		WD	=X’05’	
005E	0		TIXR	T	
0044	0		JLT	RLOOP	

BLOCK NAME	BLOCK NUMBER	ADDRESS	LENGTH
DEFAULT	0	0000	
CDATA	1		
CBLKS	2		

LOCCTR1: 0000

LOCCTR2: 0000

LOCCTR3: 0000

Symbol	Type	Value	Block
FIRST	R	0000	0
CLOOP	R	0003	0
ENDFIL	R	0015	0
RETADR	R	0000	1
LENGTH	R	0003	1
BUFFER	R	0000	2
BUFEND	R	1000	2
MAXLEN	A	1000	
RDREC	R	0027	0
RLOOP	R	0031	0
EXIT	R	0047	0
INPUT	R	0006	1
WRREC	R	004D	0
WLOOP	R	0052	0

Thank You

Program Block
Part 2

Prepared By: Dr. D. P. Singh

BLOCK NAME	BLOCK NUMBER	ADDRESS	LENGTH
DEFAULT	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

disp=TA-(PC)
=0000+0035-0021=0015
=FFA (2’S COMPLEMENT)

LOC/	BLOCK	Source Statements			Object Code								
0027	0		USE			0000 00	0	1	0	0	0	0	0000 0000 0011
0027	0	RDREC	CLEAR(B4)	X	B410	1011 0100	0001	0000	00 0100 1000				
0029	0		CLEAR(B4)	A	B400	1011 0100	0000	0000	00 0010 1001				
002B	0		CLEAR(B4)	S	B440	1011 0100	0100	0000	00 0011 1111				
002D	0		+LDT(74)	#MAXLEN	75101000	0111 01	0	1	0	0	0	1	0000 0001 0000 0000 0000
0031	0	RLOOP	TD(E0)	INPUT	E32038	1110 00	1	1	0	0	1	0	0000 0011 1000
0034	0		JEQ(30)	RLOOP	332FFA	0011 00	1	1	0	0	1	0	1111 1111 1010
0037	0		RD(D8)	INPUT	DB2032	1101 10	1	1	0	0	1	0	0000 0011 0010
003A	0		COMPR(A0)	A,S	A004	1010 0000	0000	0100	11 1110 1110				
003C	0		JEQ(30)	EXIT	332008	0011 00	1	1	0	0	1	0	0000 0000 1000
003F	0		STCH(54)	BUFFER,X	57A02F	0101 01	1	1	1	0	1	0	0000 0010 1111
0042	0		TIXR(B8)	T	B850	1011 1000	0101	0000					

Symbol	Type	Value	Block
FIRST	R	0000	0
CLOOP	R	0003	0
ENDFIL	R	0015	0
RETADR	R	0000	1
LENGTH	R	0003	1
BUFFER	R	0000	2
BUFEND	R	1000	2
MAXLEN	A	1000	
RDREC	R	0027	0
RLOOP	R	0031	0
EXIT	R	0047	0
INPUT	R	0006	1
WRREC	R	004D	0
WLOOP	R	0052	0

=C’EOF’	454F46	03	0007	1
=X’05’	05	01	000A	1

BLOCK NAME	BLOCK NUMBER	ADDRESS	LENGTH
DEFAULT	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

disp=TA-(PC)

= 0000+0052-0058=FFFA

=FFFA (2'S COMPLEMENT)

LOC/	BLOCK	Source Statements			Object Code
005E	0		TIXR(B8)	T	B850
0060	0		JLT(38)	WLOOP	3B2FEF
0063	0		RSUB(4C)		4F0000
0007	1		USE	CDATA	
			LTORG		
0007	1	*	=C'EOF'		454F46
000A	1	*	=X'05'		05
			END	FIRST	

0052	0	WLOOP	TD(E0)	=X'05'	E3201B
0055	0		JEQ(30)	WLOOP	332FFA
0058	0		LDCH(50)	BUFFER,X	53A016
005B	0		WD(DC)	=X'05'	DF2012

1011 1000				0101				0000												.1 1110 1010				
0011 10	1	1	0	0	1	0	1111 1110 1111																	
0100 11	1	1	0	0	0	0	0000 0000 0000																	
1011 0100				0001				0000																
0111 01	1	1	0	0	1	0	0000 0001 0111																	
1110 00	1	1	0	0	1	0	0000 0001 1011																	
0011 00	1	1	0	0	1	0	1111 1111 1010																	
0101 00	1	1	1	0	1	0	0000 0001 0110																	
1101 11	1	1	0	0	1	0	0000 0001 0010																	

Symbol	Type	Value	Block
FIRST	R	0000	0
CLOOP	R	0003	0
ENDFIL	R	0015	0
RETADR	R	0000	1
LENGTH	R	0003	1
BUFFER	R	0000	2
BUFEND	R	1000	2
MAXLEN	A	1000	
RDREC	R	0027	0
RLOOP	R	0031	0
EXIT	R	0047	0
INPUT	R	0006	1
WRREC	R	004D	0
WLOOP	R	0052	0

=C'EOF'	454F46	03	0007	1
=X'05'	05	01	000A	1

Thank You

Machine Independent Assembler Features

Prepared By: Dr. D. P. Singh

Outline

- Literals
- Symbol Defining Statements
- Expressions
- Program Blocks
- Control Sections and Program Linking

Symbol Defining Statement

- Labels on instructions or data areas
  - The value of such a label is the address assigned to the statement
- Defining symbols
  - symbol EQU value
  - value can be:
    - constant,
    - expression involving constants and previously defined symbols
- Uses
  - making the source program easier to understand
    - E.g. +LDT #4096
    - It can be written as

```
MAXLEN  EQU  4096
        +LDT #MAXLEN
```

  - no forward reference

Symbol Defining Statements cntd…

- Defining mnemonic names for Registers
  - e.g. suppose assembler expected register numbers instead of names in an instruction like RMO. This would require the programmer to write RMO 0,1 instead of RMO A,X. in such cases programmer could include a sequence of EQU statements like

```
A       EQU   0
X       EQU   1
L       EQU   2
```

- e.g.

```
BASE  EQU   R1
INDEX EQU   R3
```

ORG (Origin)

- Indirectly assign values to symbols
- Reset the location counter to the specified value

  ORG value
- Value can be:
  - constant, expression
- No forward reference
- Example
  - SYMBOL: 6bytes
  - VALUE: 1word
  - FLAGS: 2bytes

	SYMBOL	VALUE	FLAGS
STAB			
(100 entries)			
	⋮	⋮	⋮
	⋮	⋮	⋮
	⋮	⋮	⋮

ORG cntd…

- Using EQU statements

```
STAB      RESB   1100
SYMBOL    EQU    STAB
VALUE     EQU    STAB+6
FLAG      EQU    STAB+9
```

  This would allow us to write LDA VALUE,X

- Using ORG statements

```
STAB      RESB   1100
          ORG    STAB
SYMBOL    RESB   6
VALUE     RESW   1
FLAGS     RESB   2
          ORG    STAB+1100
```

Thank You

Program Blocks
Part 3

Prepared By: Dr. D. P. Singh

Writing Object Program

LOC/	BLOCK	Source Statements			Object Code
005E	0		TIXR(B8)	T	B850
0060	0		JLT(38)	WLOOP	3B2FEF
0063	0		RSUB(4C)		4F0000
0007	1		USE	CDATA	
			LTORG		
0007	1	*	=C'EOF'		454F46
000A	1	*	=X'05'		05
			END	FIRST	
0052	0	WLOOP	TD(E0)	=X'05'	E3201B
0055	0		JEQ(30)	WLOOP	332FFA
0058	0		LDCH(50)	BUFFER,X	53A016
005B	0		WD(DC)	=X'05'	DF2012

H COPY 000000 001071

T 000000 1E 172063 4B2021 032060 290000 332006 4B203B 3F2FEE 032055 0F2056 010003

T 00001E 09 0F2048 4B2029 3E203F

T 000027 1D B410 B400 B440 75101000 E32038 332FFA DB2032 A004 332008 57A02F B850

T 000044 09 3B2FEA 13201F 4F0000

T 00006C 01 F1

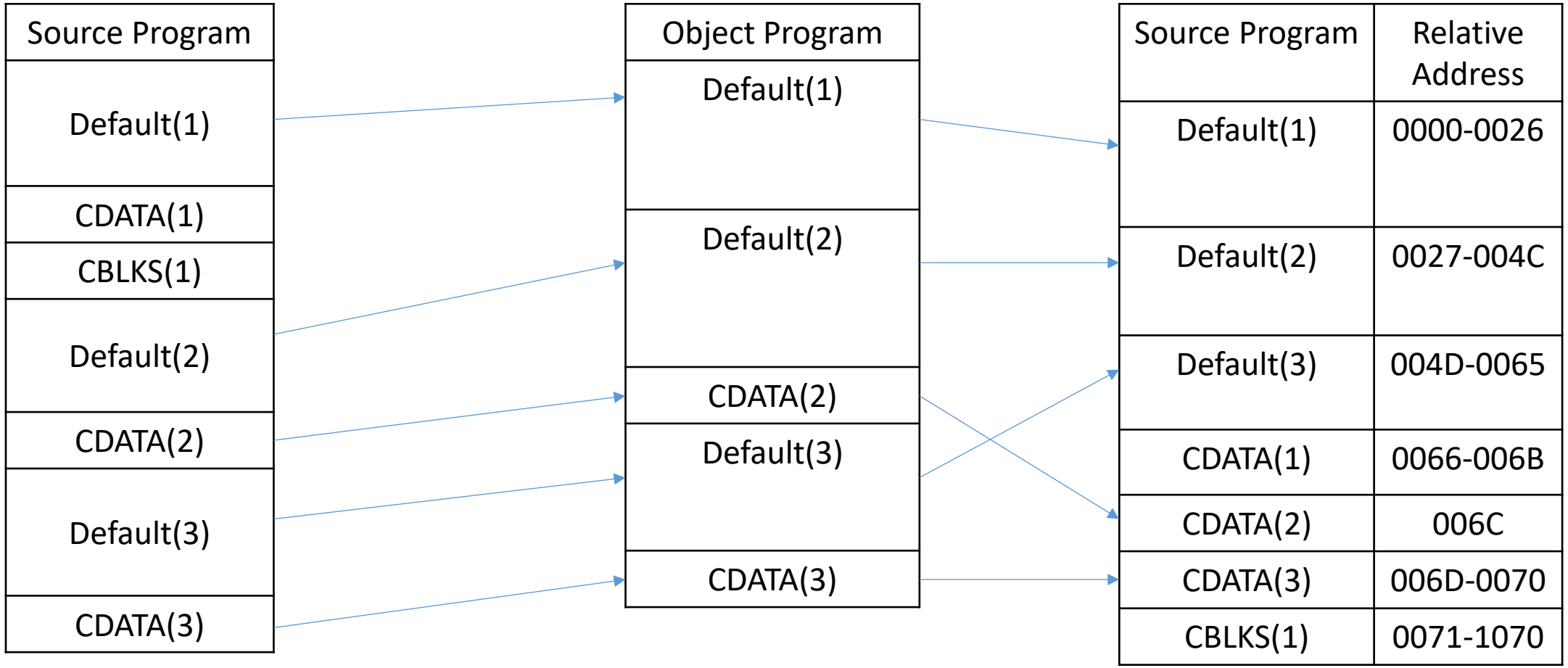
T 00004D 19 B410 772017 E3201B 332FFA 53A016 DF2012 B850 3B2FEF 4F0000

T 00006D 04 454F46 05

E 000000

BLOCK NAME	BLOCK NUMBER	ADDRESS	LENGTH
DEFAULT	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Tracing of Program Block through Assembly and Loading Process



Thank You

Control Section and Program Linking Part 1

Prepared By: Dr. D. P. Singh

Overview

- Control Sections
  - Is a part of the program that maintains its identity after assembly; each such control section can be loaded and relocated independently of the others.
  - are most often used for subroutines or other logical subdivisions of a program
  - the programmer can assemble, load, and manipulate each of these control sections separately
  - instruction in one control section may need to refer to instructions or data located in another control section
  - because of this, there should be some means for linking control sections together

External Definition and References

- External definition
  - **EXTDEF name [, name]**
  - EXTDEF names symbols that are defined in this control section and may be used by other sections
- External reference
  - **EXTREF name [,name]**
  - EXTREF names symbols that are used in this control section and are defined elsewhere

Note: A separate LOCCTR is used for each control section

Example

001D		JLT	RLOOP
0020	EXIT	+STX	LENGTH
0024		RSUB	
0027	INPUT	BYTE	X’F1’
0028	MAXLEN	WORD	BUFEND-BUFFER
	WRERC	CSECT	
		EXTREF	LENGTH, BUFFER
0000		CLEAR	X
0002		+LDT	LENGTH
0006	WLOOP	TD	=X’05’
0009		JEQ	WLOOP
000C		+LDCH	BUFFER,X
0010		WD	=X’05’
0013		TIXR	T
0015		JLT	WLOOP
0018		RSUB	
		END	FIRST
001B	*	=X’05’	

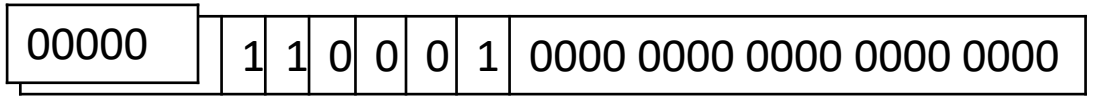
Thank You

Control Section and Program Linking Part 2

Prepared By: Dr. D. P. Singh

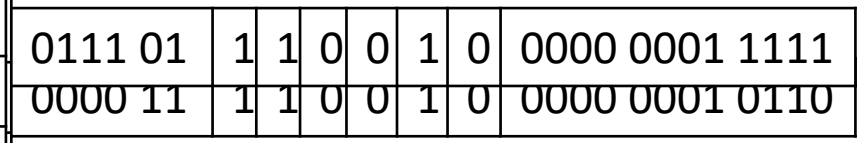
Finding Object Code

001D		JLT	RLOOP
0020	EXIT	+STX	LENGTH
0024		RSUB	
0027	INPUT	BYTE	X'F1'
0028	MAXLEN	WORD	BUFEND-BUFFER
	WRERC	CSECT	
		EXTREF	LENGTH, BUFFER
0000		CLEAR	X
0002		+LDT	LENGTH
0006	WLOOP	TD	=X'05'
0009		JEQ	WLOOP
000C		+LDCH	BUFFER,X
0010		WD	=X'05'
0013		TIXR	T
0015		JLT	WLOOP
0018		RSUB	
		END	FIRST
001B	*	=X'05'	



disp=TA-(PC)

=0028-0009=001F



Thank You

Control Section and Program Linking Part 3

Prepared By: Dr. D. P. Singh

Implementation

- The assembler must include information in the object program that will cause the loader to insert proper values where they are required
- Two more record are used in the Object Program
- Define Record
  - Col. 1 D
  - Col. 2-7 Name of external symbol defined in this control section
  - Col. 8-13 Relative address within this control section (in Hex)
  - Col.14-73 Repeat information in Col. 2-13 for other external symbols
- Refer Record
  - Col. 1 R
  - Col. 2-7 Name of external symbol referred to in this control section
  - Col. 8-73 Name of other external reference symbols

Modification Record (Revised)

- Col. 1 M
- Col. 2-7 Starting address of the field to be modified (in Hex)
- Col. 8-9 Length of the field to be modified, in half-bytes (in Hex)
- Col. 10 Modification Flag (+ or -)
- Col.11-16 External symbol whose value is to be added to or subtracted from the indicated field

- Note: control section name is automatically an external symbol, i.e. it is available for use in Modification records.

Writing Object Program (Main Control Section)

		LTORG			
0030	*	=C’EOF’		454F46	
0033	BUFFER	RESB	4096		
1033	BUFEND	EQU	*		172027
1000	MAXLEN	EQU	BUFEND-BUFFER		4B100000
					032023
000A		COMP	#0		290000
000D		JEQ	ENDFIL		332007
0010		+JSUB	WRREC		4B100000
0014		J	CLOOP		3F2FEC
0017	ENDFIL	LDA	=C’EOF’		032016
001A		STA(0C)	BUFFER		0F2016
001D		LDA	#3		010003
0020		STA	LENGTH		0F200A
0023		+JSUB	WRREC		4B100000
0027		J	@RETADR		3E2000
002A	RETADR	RESW	1		
002D	LENGTH	RESW	1		

H COPY 000000 001033

D BUFFER 000033 BUFEND 001033 LENGTH 00002D

R RDREC WRREC

T 000000 1D 172027 4B100000 032023 290000 332007 4B100000 3F2FEC 032016 0F2016

T 00001D 0D 010003 0F200A 4B100000 3E2000

T 000030 03 454F46

M 000004 05+RDREC

M 000011 05+WRREC

M 000024 05+WRREC

E 000000

Writing Object Program (RDREC Control Section)

	RDREC	CSECT		
		EXTREF	BUFFER, LENGTH, BUFEND	
0000		CLEAR	X	B410
0002		CLEAR	A	B400
0004		CLEAR	S	B440
0006		LDT(74)	MAXLEN	77201F
0009	RLOOP	TD	INPUT	E3201B
000C		JEQ	RLOOP	332FFA
000F		RD	INPUT	DB2015
0012		COMPR	A,S	A004
0014		JEQ	EXIT	332009
0017		+STCH	BUFFER,X	57900000
001B		TIXR	T	B850
001D		JLT	RLOOP	3B2FE9
0020	EXIT	+STX	LENGTH	13100000
0024		RSUB		4F0000
0027	INPUT	BYTE	X'F1'	F1
0028	MAXLEN	WORD	BUFEND-BUFFER	000000

H RDREC 000000 00002B

R BUFFER LENGTH BUFEND

T 000000 1D B410 B400 B440 77201F E3201B 332FFA DB2015 A004 332009 57900000 B850

T 00001D 0E 3B2FE9 13100000 4F0000 F1 000000

M 000018 05+BUFFER

M 000021 05+LENGTH

M 000028 06+BUFEND

M 000028 06-BUFFER

E

Writing Object Program (WRREC Control Section)

	WRERC	CSECT		
		EXTREF	LENGTH, BUFFER	
0000		CLEAR	X	B410
0002		+LDT	LENGTH	77100000
0006	WLOOP	TD	=X’05’	E32012
0009		JEQ	WLOOP	332FFA
000C		+LDCH	BUFFER,X	53900000
0010		WD	=X’05’	DF2008
0013		TIXR	T	B850
0015		JLT	WLOOP	3B2FEE
0018		RSUB		4F0000
		END	FIRST	
001B	*	=X’05’		05

H WRREC 000000 00001C

R LENGTH BUFFER

T 000000 1C B410 77100000 E32012 332FFA 53900000 DF2008 B850 3B2FEE 4F0000 05

M 000003 05+LENGTH

M 00000D 05+BUFFER

E

Thank You