MIT-WPU
T.Y. B.Tech

System Software and Compilers

Course Objective & Course Outcomes

- **Course Objectives:**

1. To learn and understand different component of system software and fundamentals language processing activity.
2. To understand the process of converting assembly language program to machine language
3. To understand linking and loading concepts
4. Understand the basic concept of compiler design, and its different phases and tools.

- **Course Outcomes:**

1. Obtain knowledge in different component of systems software and fundamentals of language processing activity.
2. Design two pass assembler and Direct Linking Loaders.
3. Acquire knowledge in different phases and passes of Compiler.
4. Design different types of compiler tools to meet the requirements of the realistic constraints of compilers using LEX and YACC tools.

Text Books & Reference Books

Text Books:

1. Dhamdhere D., "Systems Programming and Operating Systems", McGraw Hill, ISBN 0 - 07 -463579 – 4.
2. A V Aho, R Sethi, J D Ullman, \Compilers: Principles, Techniques, and Tools", Pearson Edition, ISBN 81-7758-590-8.
3. John Donovan, “System Programming”, McGraw Hill, ISBN 978-0--07-460482-3.

Reference Books:

1. John. R. Levine, Tony Mason and Doug Brown, “Lex and Yacc”, O'Reilly, 1998, ISBN: 1- 56592-000-7.
2. Leland L. Beck, “System Software An Introduction to Systems Programming” 3rd Edition, Person Education, ISBN 81-7808-036-2.
3. Adam Hoover, “System Programming with C and Unix”, Pearson,2010

Module II

- Macro processor: Macro Definition, Macro expansion and nested macros
- **Loaders: Loader schemes: Types of loaders, direct linking loaders.**
- Linkers: Relocation and linking concepts, self-relocating programs, Static and dynamic link libraries.

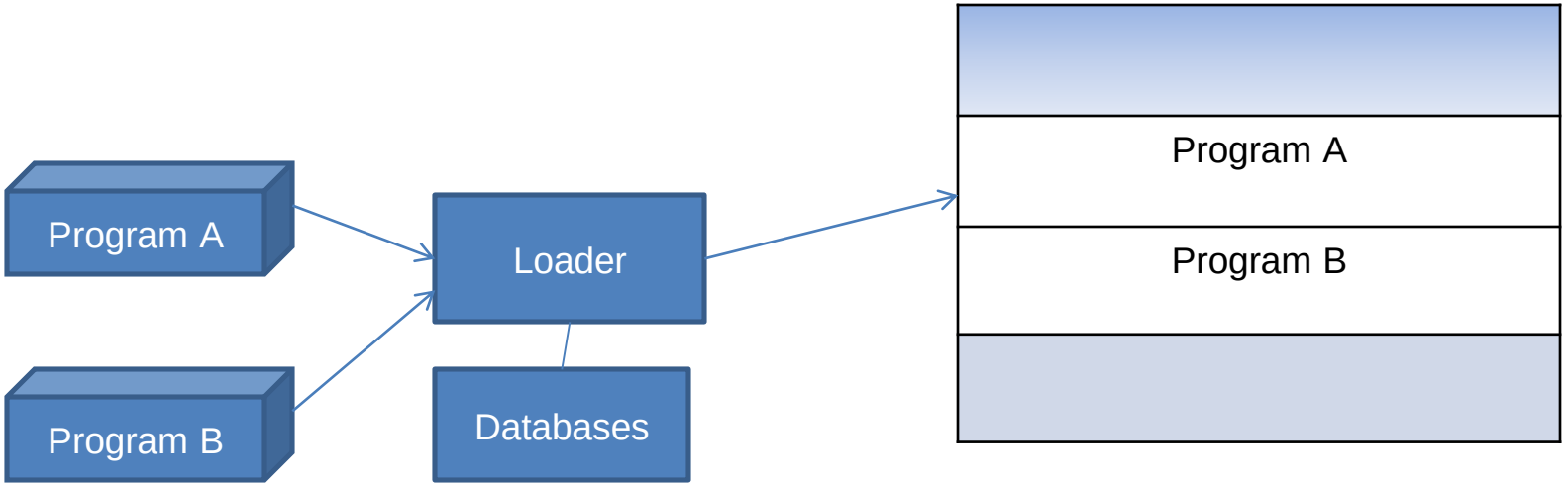
Loader

Loaders

Loader is a program that accepts the object program, prepares these programs for execution by the computer and initiates the execution of the program.

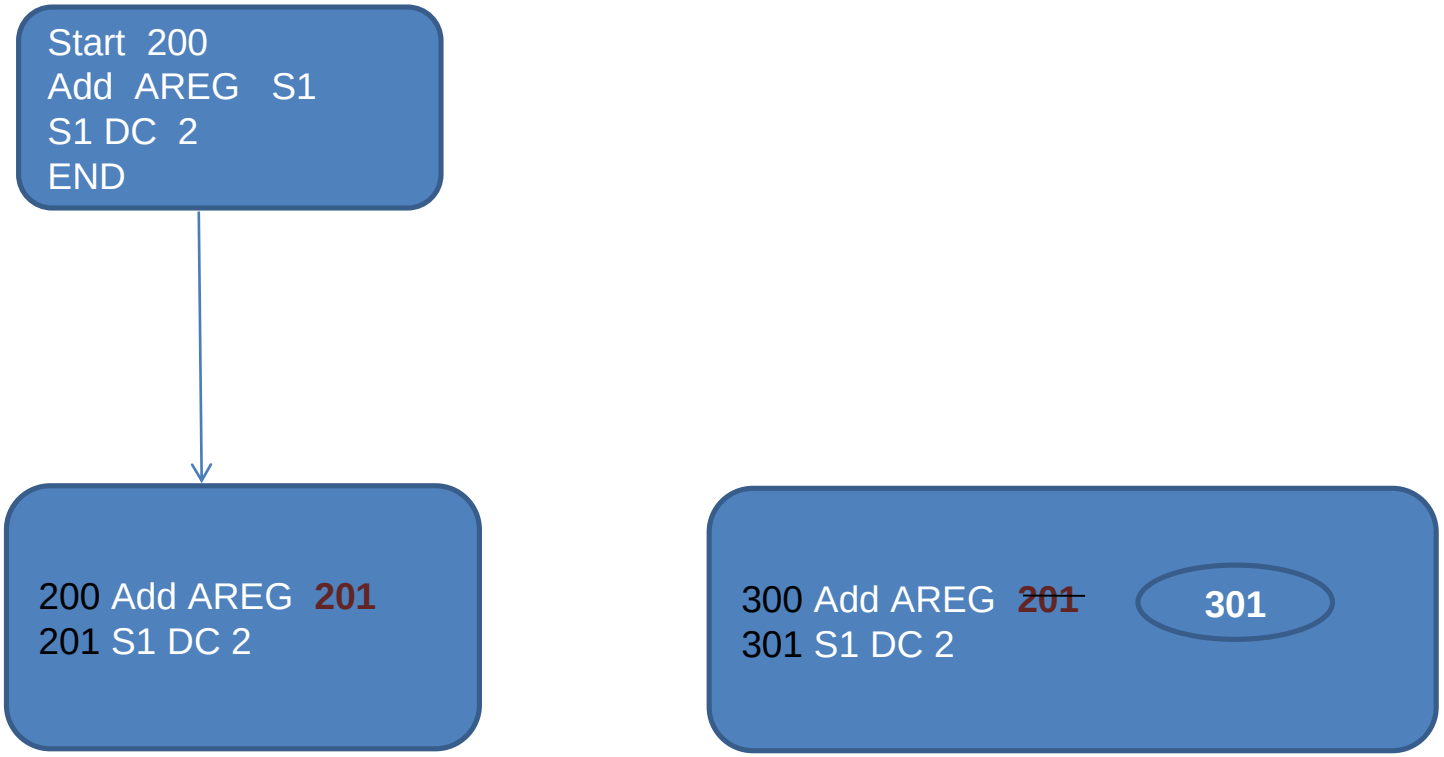
Functions of the loader

1. Allocate space in memory for the programs.(**Allocation**)
2. Resolve symbolic references between object decks.(**Linking**)
3. Adjust all addr dependent locations, such as addr constants, to correspond to the allocated space.(**Relocation**)
4. Physically place the m/c instr and data into memory.(**Loading**)

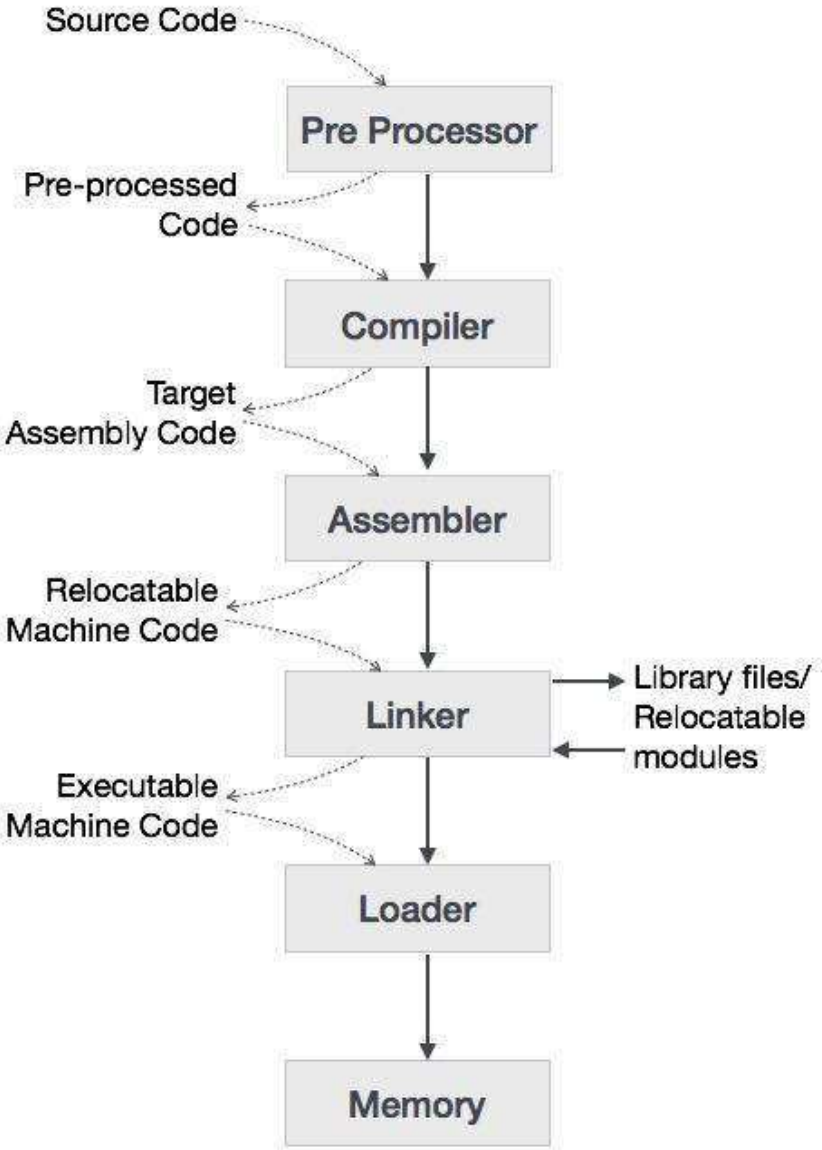


Programs loaded in memory ready for execution

Concept of Relocation



Language Processing System



Different Types of Loader Schemes

1. Compile and Go Loaders
2. General Loader Scheme
3. Absolute Loaders
4. Relocating Loaders
5. Direct Linking Loaders

1. Compile and Go Loaders

-Assembler places the code into core

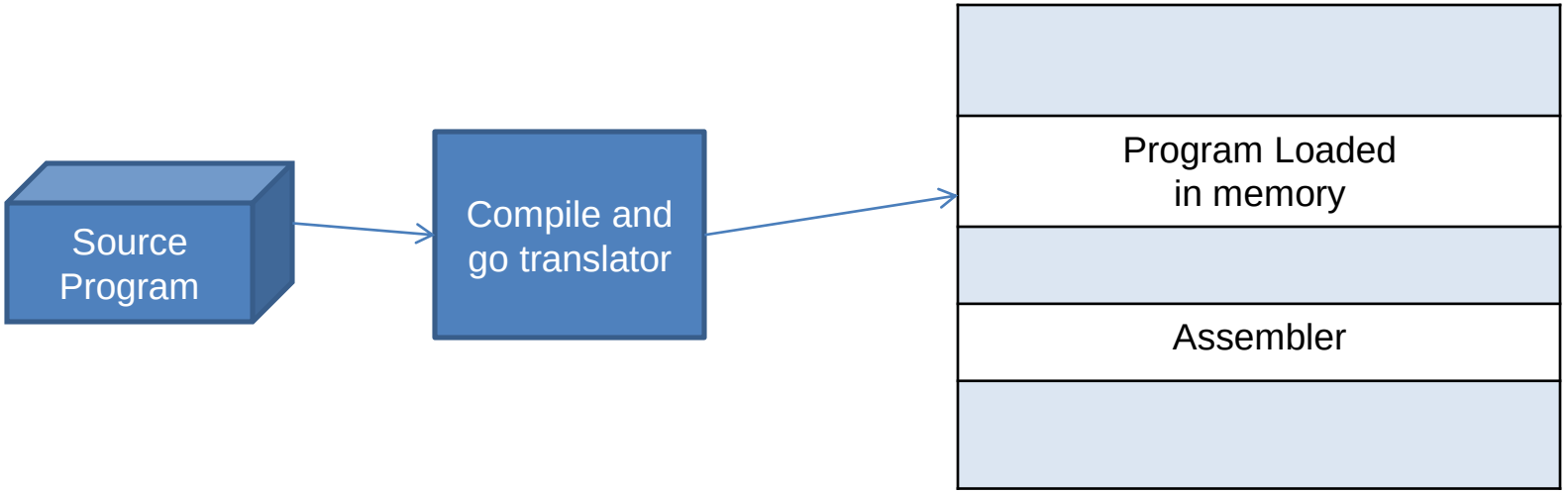
-Loader consists of one instr that transfers to the starting instr of the newly assembled program

-easy to implement

-portion of memory is wasted because of assembler

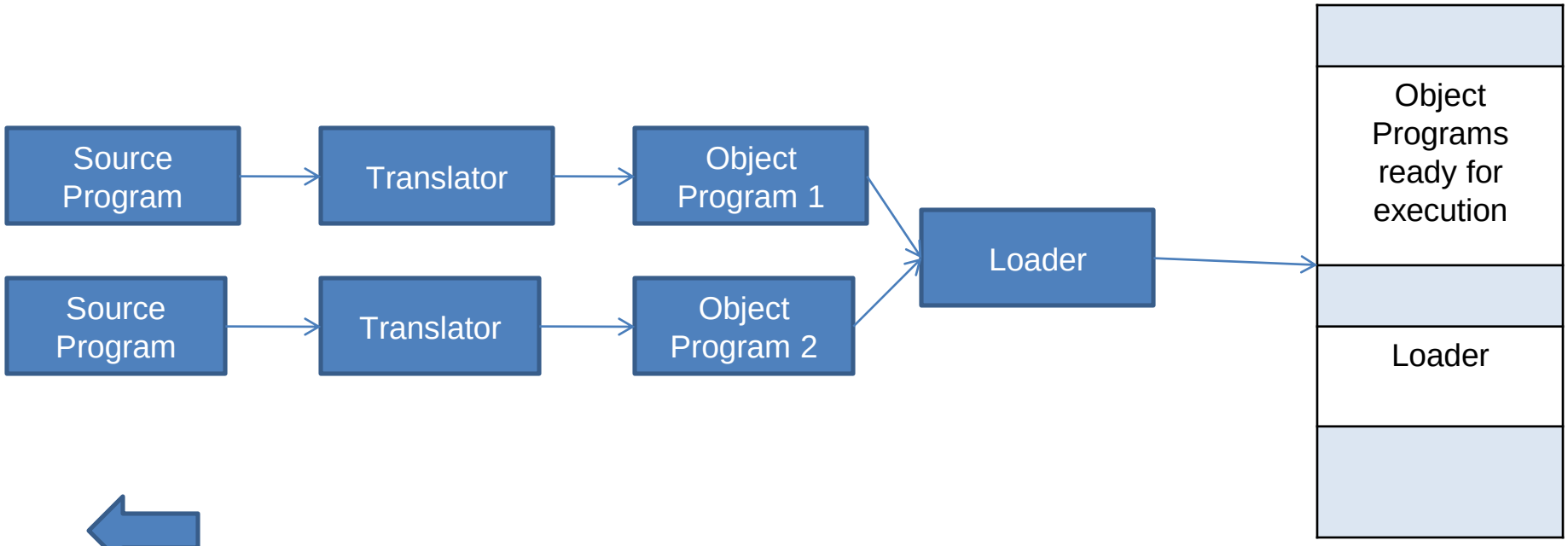
-every time the program is run it has to be retranslated

-difficult to handle multiple segments



2. General Loader Scheme

- As loader is smaller than assembler more memory is available
- Reassembling of program is not required to run the program later.
- Loader is present in memory.



3. Absolute Loader

-Same as “compile and go “ loader except data is punched on cards instead of memory.

-Loader accepts m/c language text and places it into memory at the location specified by the assembler

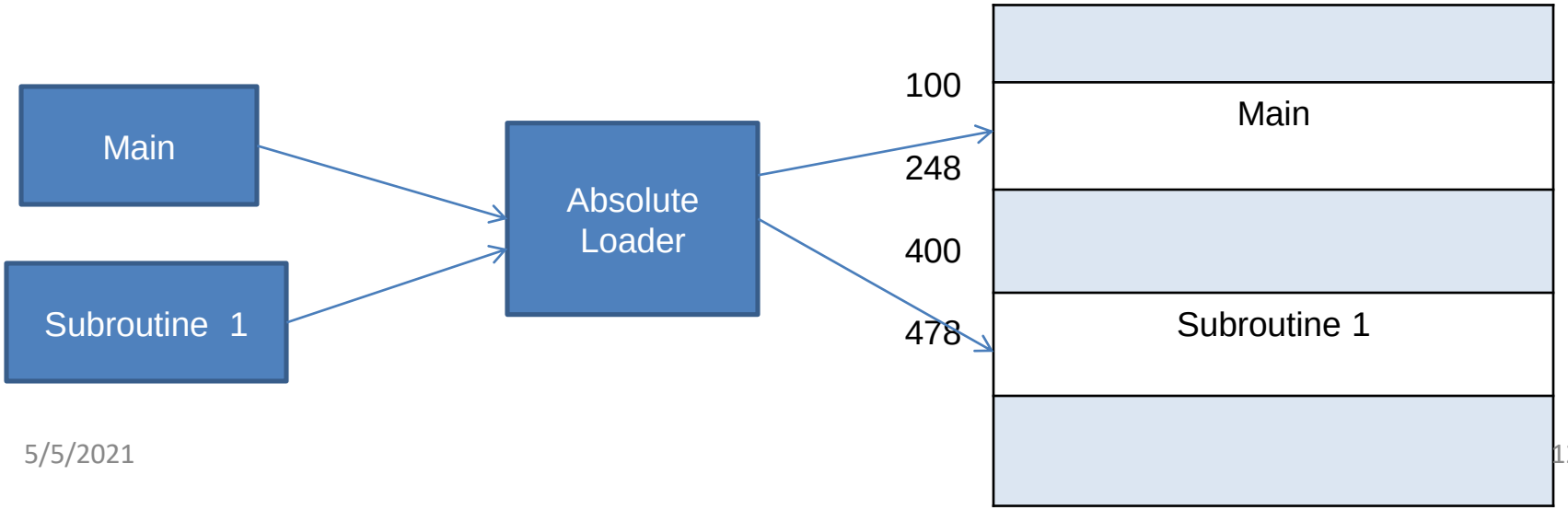
Advantages

-More memory is available, -simple to implement

Disadvantages

-Programmer must specify address to the assembler where the program is loaded.

-In case of multiple subroutines, programmer has to remember address of each subroutine.



Relocating Loader (Binary Symbolic Subroutine)

-To **avoid** possible **reassembling** of all subroutines when a single subroutine is changed.

-To perform **task of allocation and linking** for the programmer.

-Allows **many procedure segments** but only one data segment.

-Translated code segments and the information regarding relocation and intersegment references is passed to the loader.

Information provided by the assembler to the BSS loader.

- ***Transfer Vector***
- Contains the address and names related to the subroutines referenced in the program.
- Total length of the program
- length of transfer vector
- ***Relocation Bits***
- relocation bit is associated with every instruction
- Relocation bits can be 0 or 1.
- If 1 then address field needs relocation
- If 0 then address field does not need relocation

ST	14	SAVE
ST	14	SAVE
Relocation Bit=0		Relocation Bit=1

4. Relocating Loader (Binary Symbolic Subroutine)

- In BSS
- All four functions of loader (allocation, linking, relocation and loading)are performed automatically by the BSS loader.

- **Relocation bits** are used to solve the problem of relocation.

- The **transfer vector** is used to solve the problem of linking.

- The **program length** information is used to solve the problem of allocation.

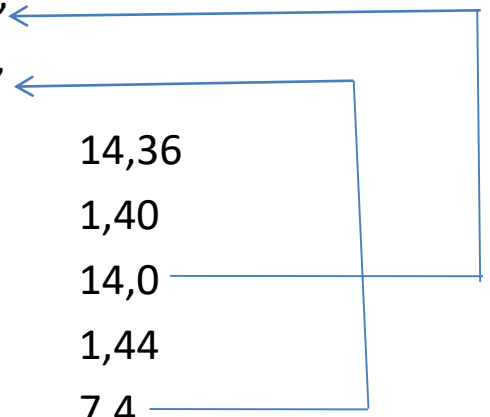
4. Relocating Loader (Contd..)

Source program

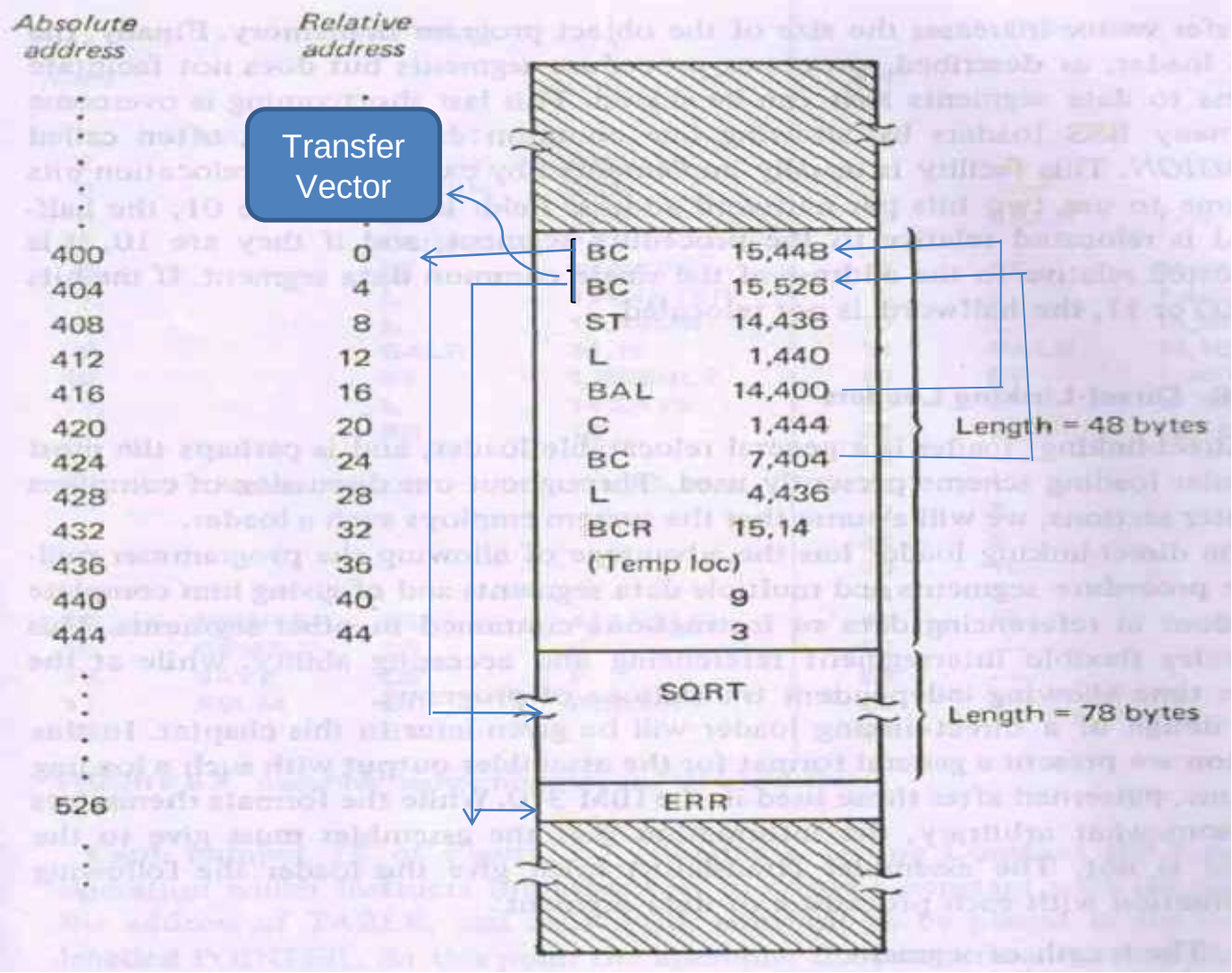
Program Length = 48 bytes

Transfer Vector = 8 bytes

			Rel. Addr.	Rel Bits	Object Code	
MAIN	START					
	EXTRN	SQRT	0	00	'SQRT'	
	EXTRN	ERR	4	00	'ERRb'	
	ST	14,SAVE	8	01	ST	14,36
	L	1,=F'9'	12	01	L	1,40
	BAL	14,SQRT	16	01	BAL	14,0
	C	1,=F'3'	20	01	C	1,44
	BNE	ERR	24	01	BC	7,4
	L	14,SAVE	28	01	L	14,36
	BR	14	32	0	BCR	15,14
			34	0	(Skipped for alignment)	
SAVE	DS	F	36	00	(Temp location)	
	END		40	00	9	
			44	00	3	



4. Relocating Loader (Contd..)



Disadvantages of Relocating Loader

- The transfer vector linkage is only useful for transfers and **not well suited for loading** or storing external data.
- The transfer vector **increases the size of the object program** in memory.
- BSS loader processes procedure segments but **does not facilitate access to data segments** that can be shared.

5. Direct Linking Loader

- *Flexible intersegment referencing and accessing ability.*
- *Allows independent translation of programs.*

Information provided by the assembler with each procedure or data segment

- *Length of the segment.*
- *List of symbols and relative locations.*
- List of symbols not defined but referenced.
- Information where address constants are located.
- M/c translation of source program and relative address assigned.

Assembler produces 4 types of cards in the object deck.

ESD → External Symbol Dictionary.

TXT → Actual Object Code.

RLD → Relocation and Linkage Directory.

END → End of object deck.

5. Direct Linking Loader(contd…)

ESD cards

- Contains info related to all the symbols defined and referenced in the program.
- Values for ESD cards are

  SD (Segment Definition) → name on START card

  LD (Local Definition) → Specified on ENTRY card

  ER (External Reference) → specified on EXTRN card

TXT cards

- Contains actual object code translated version of program.

RLD cards

- The location constant that needs to be changed due to relocation
- By what is has to be changed
- The operation to be performed(+/-)

END cards

- End of object deck and specifies the starting address for execution if the assembled routine is the main program.

5. Direct Linking Loader(contd…)

Card No	ALP	Rel Loc	Translation
1.	JOHN START		
2.	ENTRY RESULT		
3.	EXTRN SUM		
4.	BALR 12, 0	0	BALR 12,0
5.	USING *, 12		[12]<-- current value of LC
6	ST 14, SAVE	2	ST 14, 54(0,12)
7.	L 1, POINTER	6	L 1, 46(0,12)
8.	L 15, ASUM	10	L 15, 58(0,12)
9.	BALR 14, 15	14	BALR 14, 15
10.	ST 1, RESULT	16	ST 1, 50(0,12)
11.	L 14, SAVE	20	L 14, 54(0,12)
12.	BR 14	24	BCR 15, 14
13.	TABLE DC F '1, 7, 9, 10, 3'	28 32 36 40 44	1 7 9 10 3
14.	POINTER DC A(TABLE)	48	28
15.	RESULT DS F	52	-
16.	SAVE DS F	56	-
17.	ASUM DC A(SUM)	60	?
18.	END	64	

Offset + [index reg] + [base reg]
54 + 0 + [12] = 54 + 0 + 2 = 56

ESD And RLD Cards

ESD Cards				
Ref No	Symbol	Type	Relative Loc	Length
1.	JOHN	SD	0	64
2.	RESULT	LD	52	-
3.	SUM	ER	-	-

RLD Cards				
Ref No	Symbol	Flag	Length	Rel Loc
14	JOHN	+	4	48
17	SUM	+	4	60

TXT Cards

TXT Cards			
Ref No	**Rel Loc**	**Object Code**	
4	0	BALR	12,0
6	2	ST	14, 54(0,12)
7	6	L	1, 46(0,12)
8	10	L	15, 58(0,12)
9	14	BALR	14, 15
10	16	ST	1, 50(0,12)
11	20	L	14, 54(0,12)
12	24	BCR	15, 14
13	28	1	
13	32	7	
13	36	9	
13	40	10	
13	44	3	
14	48	28	
17	60	0	