UNIT III

A. **Linkers and Loaders**

We know…

- Source Program – Assembly Language
- Object Program - From assembler
  - Contains translated instructions and data values from the source program
- Executable Code - From Linker
- Loader - Loads the executable code to the specified memory locations and code gets executed.

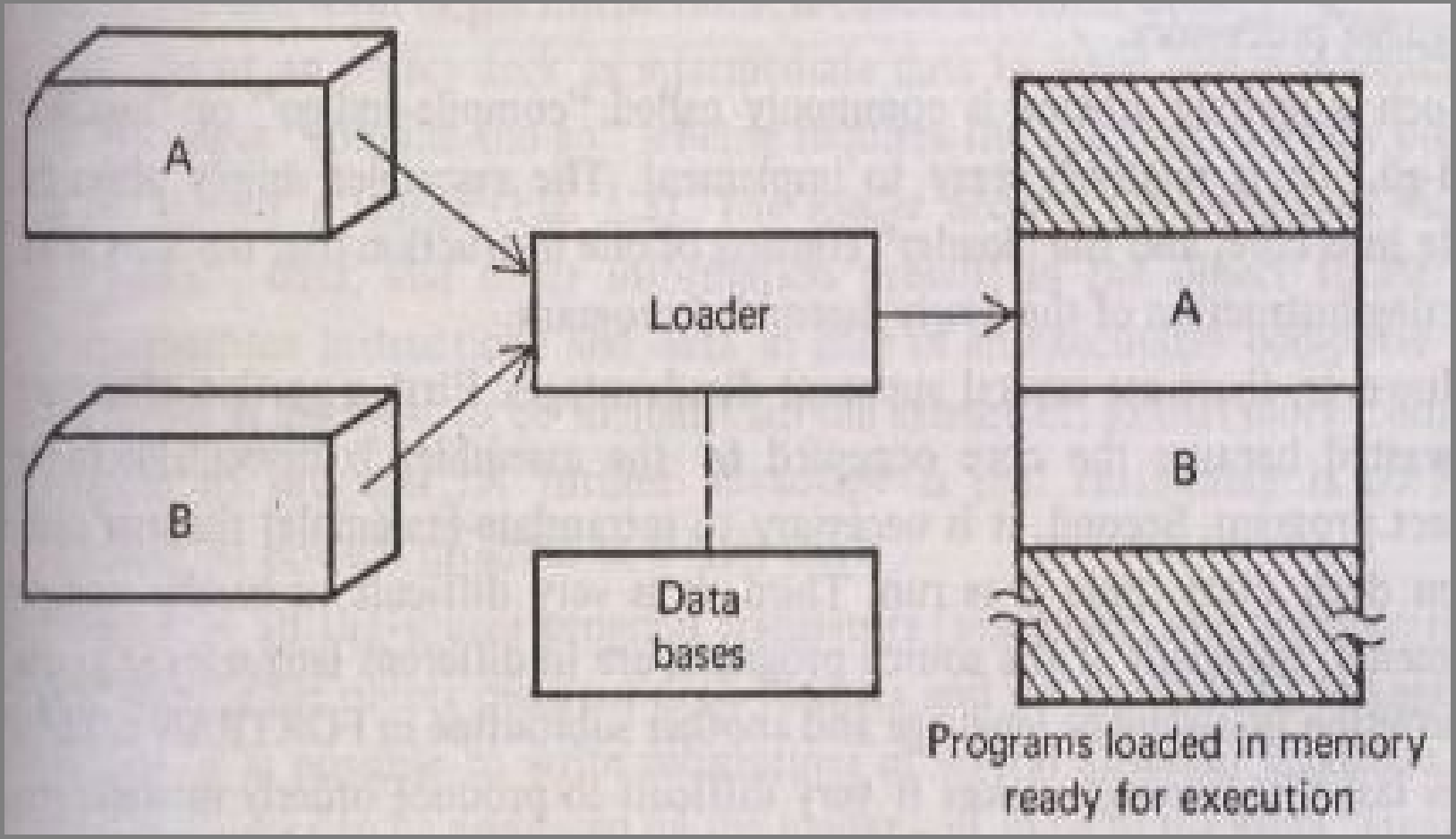
Basic Functions

- ***Allocation***: allocate space in memory for the programs
- ***Linking****:* Resolve symbolic references between object files
  - combines two or more separate object programs
  - supplies the information needed to allow references between them

Basic Functions

- ***Relocation**:* Adjust all address dependent locations, such as address constants, to correspond to the allocated space
  - modifies the object program so that it can be loaded at an address different from the location originally specified
- ***Loading**:* Physically place the machine instructions and data into memory

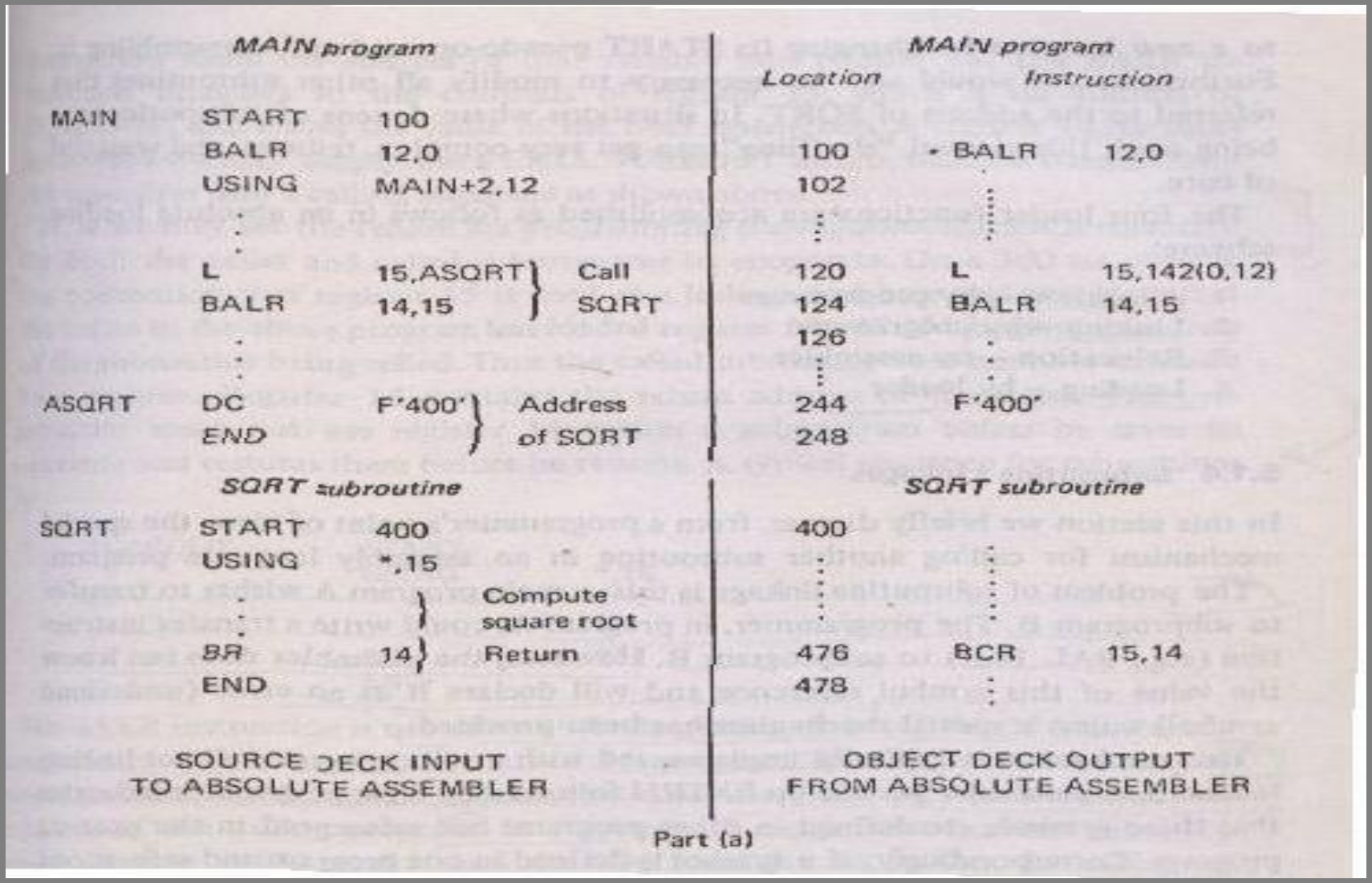
General loading scheme



Type of Loaders

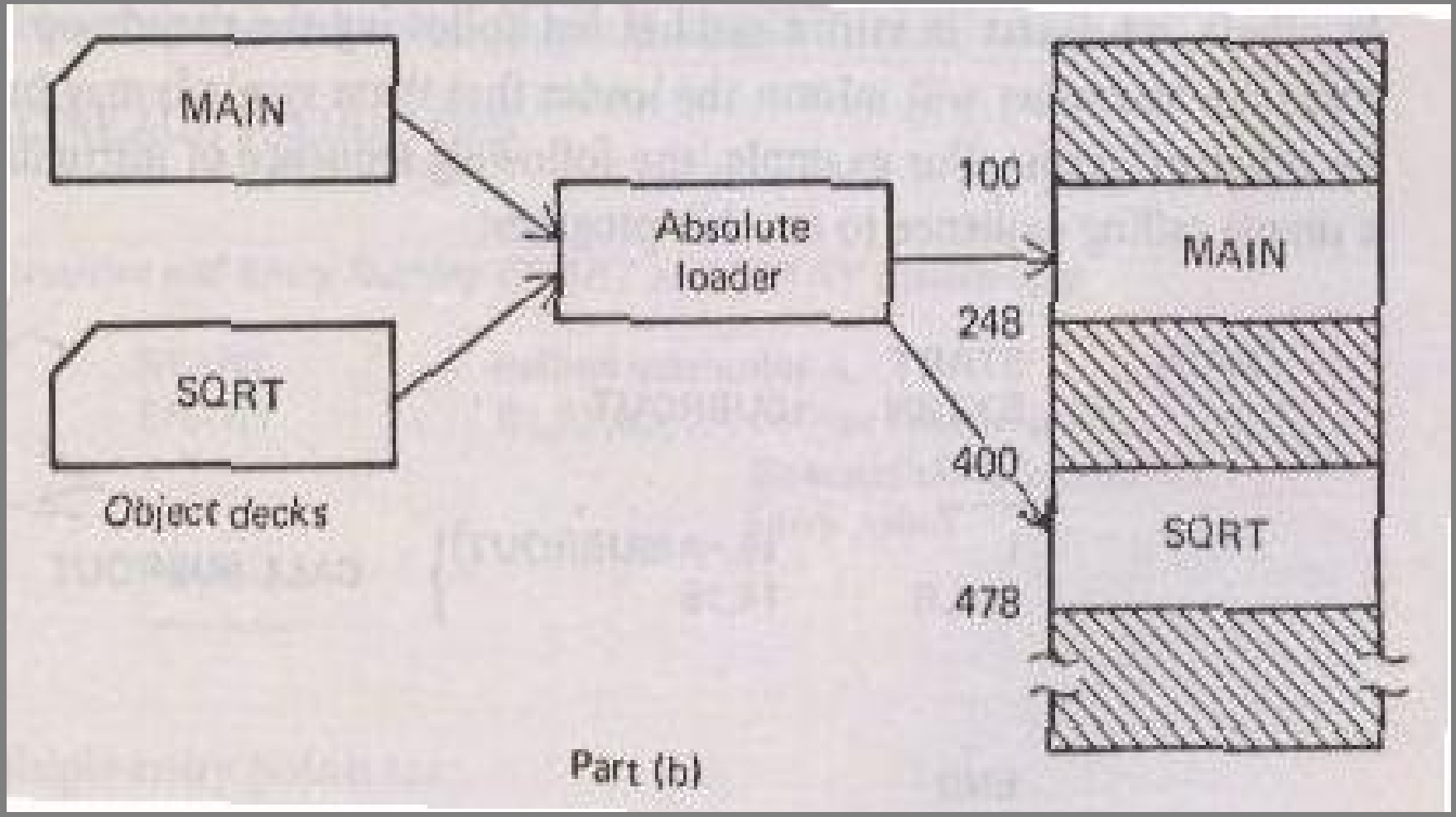
1. “Compile-and-go” loader
2. General loader scheme
3. Absolute loader
4. Subroutine linkage
5. Relocating loader (relative loader)
6. Direct linking loader
7. A binder and a module loader
8. Dynamic loading
9. Dynamic linking

Absolute Loader



Part (a)

Absolute Loader



Externals and Entries

- The assembler pseudo-op EXT followed by a list of symbols indicates that the symbols are defined in other programs but referenced in the present program
- If a symbol is defined in one program and referenced in others,
  - insert it into a symbol list following the pseudo-op ENT.

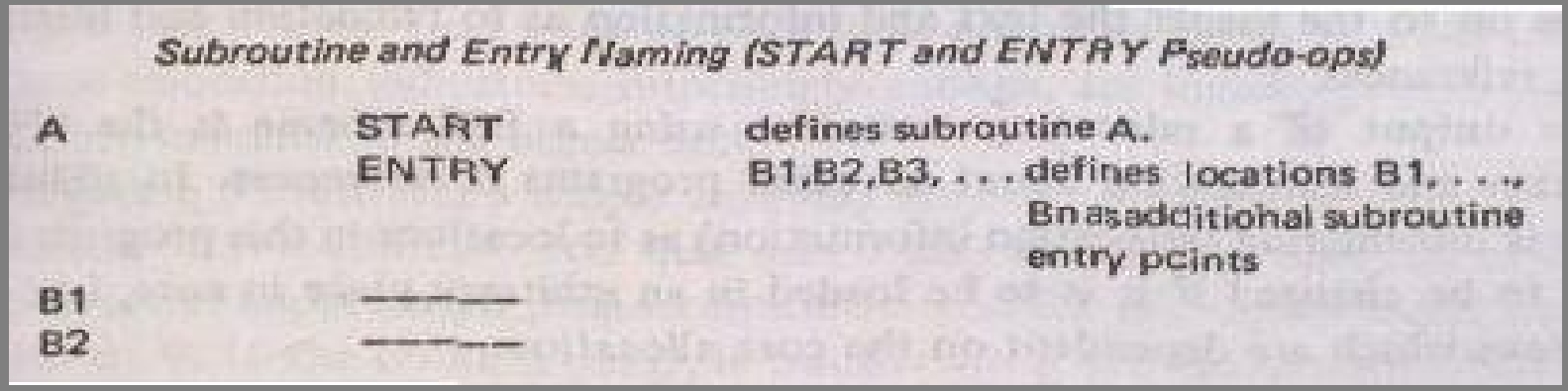
4. Subroutine linkage

```
MAIN    START
        EXTRN   SUBROUT
        ----
        ----
        L       15,=A(SUBROUT)  }
        BALR    14,15           }   CALL SUBROUT
          .
          .
          .
        END
```

```
SUBROUT START
        USING   *,15
          :
        BR      14
        END
```

Subroutine linkage

- Assembler linkage pseudo-ops



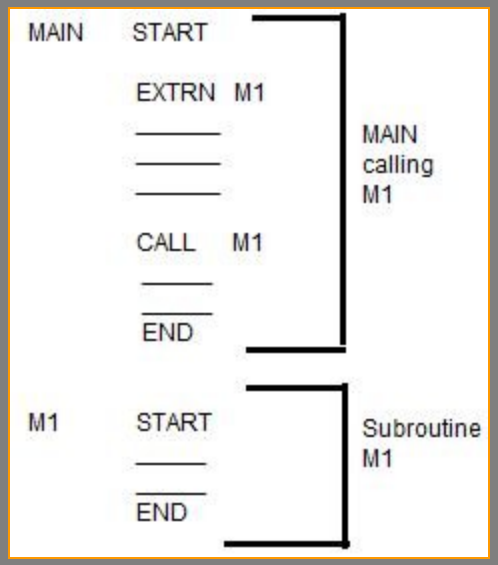
Subroutine and Entry Naming (START and ENTRY Pseudo-ops)

A	START	defines subroutine A.
	ENTRY	B1,B2,B3, . . . defines locations B1, . . ., Bn as additional subroutine entry points
B1	———	
B2	———	

- Uses of multiple entry points are
  - Common coding
  - Collecting together related routines for convenience
  - Better or convenient access to common data base

…contd..Subroutine Linkages

- **EXTRN Statements**: they list the SYMBOLS to which external references are made in current program unit
  - These SYMBOLS are defined in other program unit
  - Diagram shows MAIN and M1 in separate program units
  - Assembler cannot provide address of external symbols

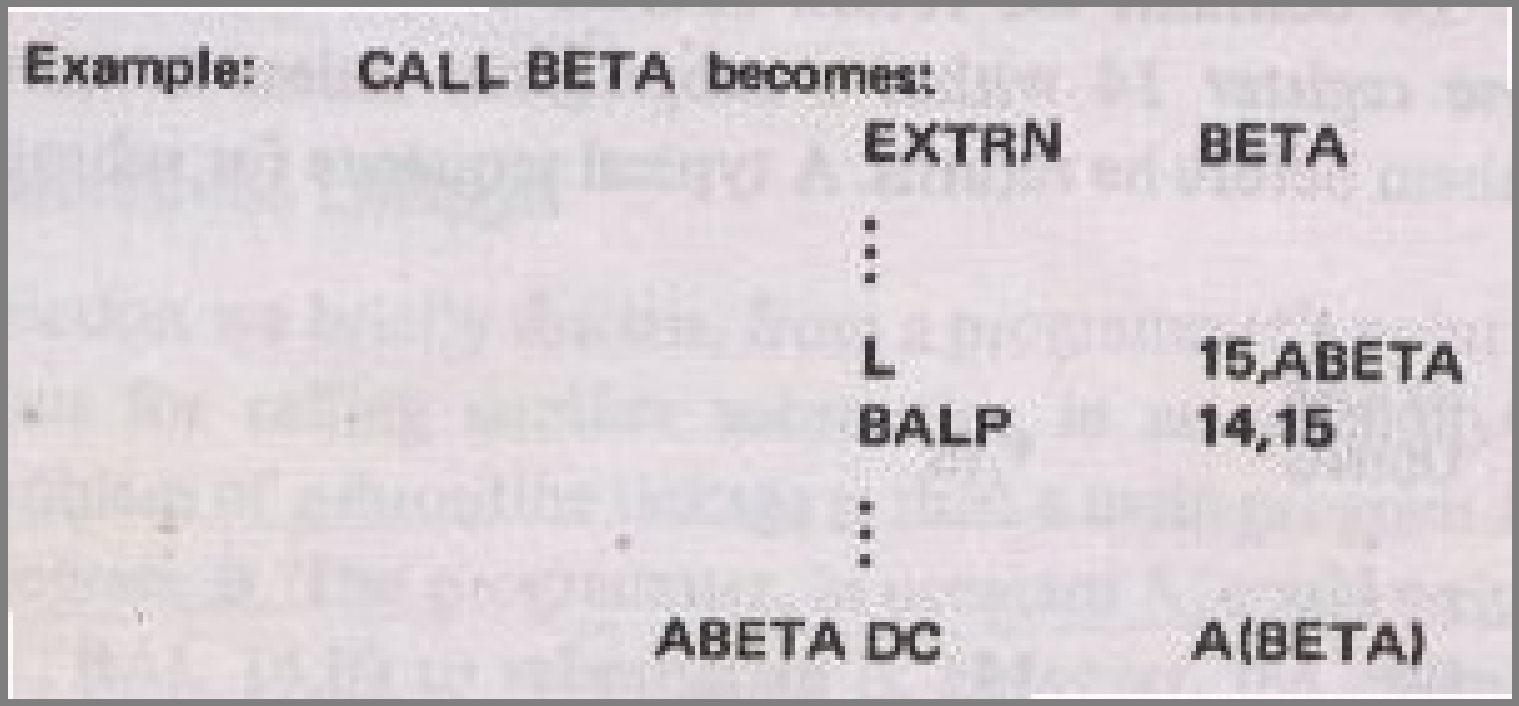


- **ENTRY Statements**: they list the public definitions of a program unit
  - Lists SYMBOLS defined the program unit which may be used in other program units

- External symbols are **UNRESOLVED** until **LINKING**

Subroutine linkage

- Subroutine reference (EXTRN pseudo-op)



5. Relocating Loaders

- To avoid possible reassembling of all subroutines when a single subroutine is changed , relocating Loader is used.
- Efficient sharing of the machine with larger memory and when several independent programs are to be run together
- Support the use of subroutine libraries efficiently

- Perform task of allocation & linking for programmer this loader is introduced
- It allows the programmer multiple procedure segments and single data segments

Relocation

- Compilers and assemblers generate the object code for each input module with a starting address of zero (Generally)
- Relocation is the process of assigning **load addresses** to different parts of the program by merging all sections of the same type into one section. The code and data section also are adjusted so they point to the correct runtime addresses.

What is RELOCATION?

- Process of modifying the addresses used in the "address sensitive instructions" so that the program executes correctly from any designated area of the memory
  - Sample of (memory) address sensitive instruction:-

    MOVER AREG, X

    BC ANY, L1

Meaning of these 3 Origins : Translated, Linked and Loaded

- Translated Origin :
  - Address specified by Programmer (ORIGIN)
- Linked Origin:
  - Address specified by Linker when producing the machine code
- Loaded Origin:
  - Address specified by Loader when loading program for execution

Relocation

- Relocating loaders or relative loaders:
  - loaders that allow for program relocation.
- Two methods for specifying relocation as part of the object program:
- **1. A Modification record**
  - describe each part of the object code that must be changed when the program is relocated

Second Method

- **Bit mask:** A relocation bit/byte associated with each word of object code
  - S for Absolute: does not need modification
  - R for Relative: needs relocation
  - X for external.

Relocating Loaders

- Allocating subroutines to prevent reassembling the code every time a subroutine changes
- Binary Symbolic Subroutine (BSS) Loader
  - The program length information is for allocation.
  - Bit Mask is used for relocation
  - The transfer vector is used to solve the problem of linking

…contd..Relocation concepts

- Program Relocatability : Ability to load and execute program in into arbitrary location in memory
- 2 basic type of relocation: (Based on when mapping of virtual address to physical address is done)
  - Static Relocation
  - Dynamic Relocation

Binary Symbolic Subroutine Loader

- The assembler assembles Provides the loader
  - Object program + relocation information
  - Prefixed with information about all other program it references (transfer vector).
  - The length of the entire program
  - The length of the transfer vector portion

Transfer Vector

- A transfer vector consists of
  - addresses containing names of the subroutines referenced by the source program
  - if a Square Root Routine (SQRT) was referenced and was the first subroutine called, the first location in the transfer vector could contain the symbolic name SQRT.
  - The statement calling SQRT would be translated into a branch to the location of the transfer vector associated with SQRT

The loader

- loads the text and the transfer vector
- loads each subroutine identified in the transfer vector.
- place a transfer instruction to the corresponding subroutine in each entry in the transfer vector.
  - The execution of the call SQRT statement result in a branch to the first location in the transfer vector
  - which contains a branch to the location of SQRT.

BSS Scheme Disadvantages

1. the transfer vector increases the size of the object program in memory
2. the BSS loader does not facilitate access to data segments that can be shared
   - the transfer vector linkage is only useful for transfers
   - not well suited for loading or storing external data (data located in another procedure segment)

6. Direct linking loader

- A general relocatable loader
- It allows the programmer multiple procedure segments and multiple data segments and gives complete freedom in referencing data or instructions contained in other program
- It provides flexible intersegment referencing and accessing ability, while at the same time allowing independent translations of programs

Direct Linking Loader

- The assembler provides
  1. The length of segment
  2. A list of all entries and their relative location within the segment
  3. A list of all external symbols
  4. Information as to where address constants are loaded in the segment and a description of how to revise their values.
  5. The machine code translation of the source program and the relative addresses assigned

Direct-linking loader

- Four sections of the object deck
  - External Symbol Dictionary cards (ESD)
  - Instructions and data cards, called “text” of program (TXT)
  - Relocation And Linkage Directory cards (RLD)
  - End card (END)

Example

```
John    START
        ENTRY   RESULT
        EXTERNAL                                SUM
        LOAD    1,POINTER       0   LOAD    1,48
        LD-ADDR 15 ASUM         4   LD-ADDR 15,56
        BALR    14 15           8   BALR    14 15
        STORE   l RESULT        12  STORE   1,52
        HLT                     1C  HLT     0,0

TABLE   DC      1,7,9,10,13     28  0001
                                2A  0007
                                2C  0009
                                30  000A
                                34  000D

POINTER DATA    ADDR(TABLE)     48  0028
RESULT  NUM     0               52  0000
ASUM    DATA    ADDR(SUM)       56  ????    EXTERNAL
        END
```

Assembler records

- External Symbol Dictionary (ESD) record: Entries and Externals
- (TXT) records control the actual object code translated version of the source program.
- The Relocation and Linkage Directory (RLD) records relocation information
- The END record specifies the starting address for execution

ESD and RLD

- SD: Segment Definition
- Local Definition
- External Reference

ESD records

symbol	type	Rel-location	Length
JOHN	SD	0	64
RESULT	LD	52	
SUM	ER	—	—

RLD record

Symbol	Flag	Length	Relative location
JOHN	+	4	48
SUM	+	4	56

Disadvantages of Direct Linking

- It is necessary to allocate, relocate, link, and load all of the subroutines each time in order to execute a program
  - loading process can be extremely time consuming.
- Though smaller than the assembler, the loader absorbs a considerable amount of space
  - Dividing the loading process into two separate programs a binder and a module loader can solve these problems.

A binder and a module loader

- Classes of binders
  - Core image builder
    - The specific core allocation of the program is performed at the time that the subroutines are bound together.
  - Linkage editor
    - Can keep track of the relocation information so that the resulting load module, can be further relocated & loaded anywhere in core.

…contd..

- Linking Loader : performs linking and relocation at load time
- Other alternatives
- Linkage editors: which performs linking prior to load time
- Dynamic linking: where linking function is performed at execution time

7. A binder and a module loader

- A binder is a program that performs the same functions as the direct-linking loader in “binding” subroutines together, but rather than placing the relocated and linked text directly into memory, it outputs the text as a file or card deck.
- This output file is in a format ready to be loaded which is called as load module.
- The binder essentially performs the functions of allocation, relocation, and linking; the module loader performs the function of loading.
- A sophisticated binder is linkage editor

Dynamic loading

- In order for overlay structure to work it is necessary for the module loader to load the various procedures as they are needed.
- The portion of the loader that actually intercepts the "calls" and loads the necessary procedure is called the overlay supervisor or flipper.
- This overall scheme is called dynamic loading or load-on-call (LOCAL).

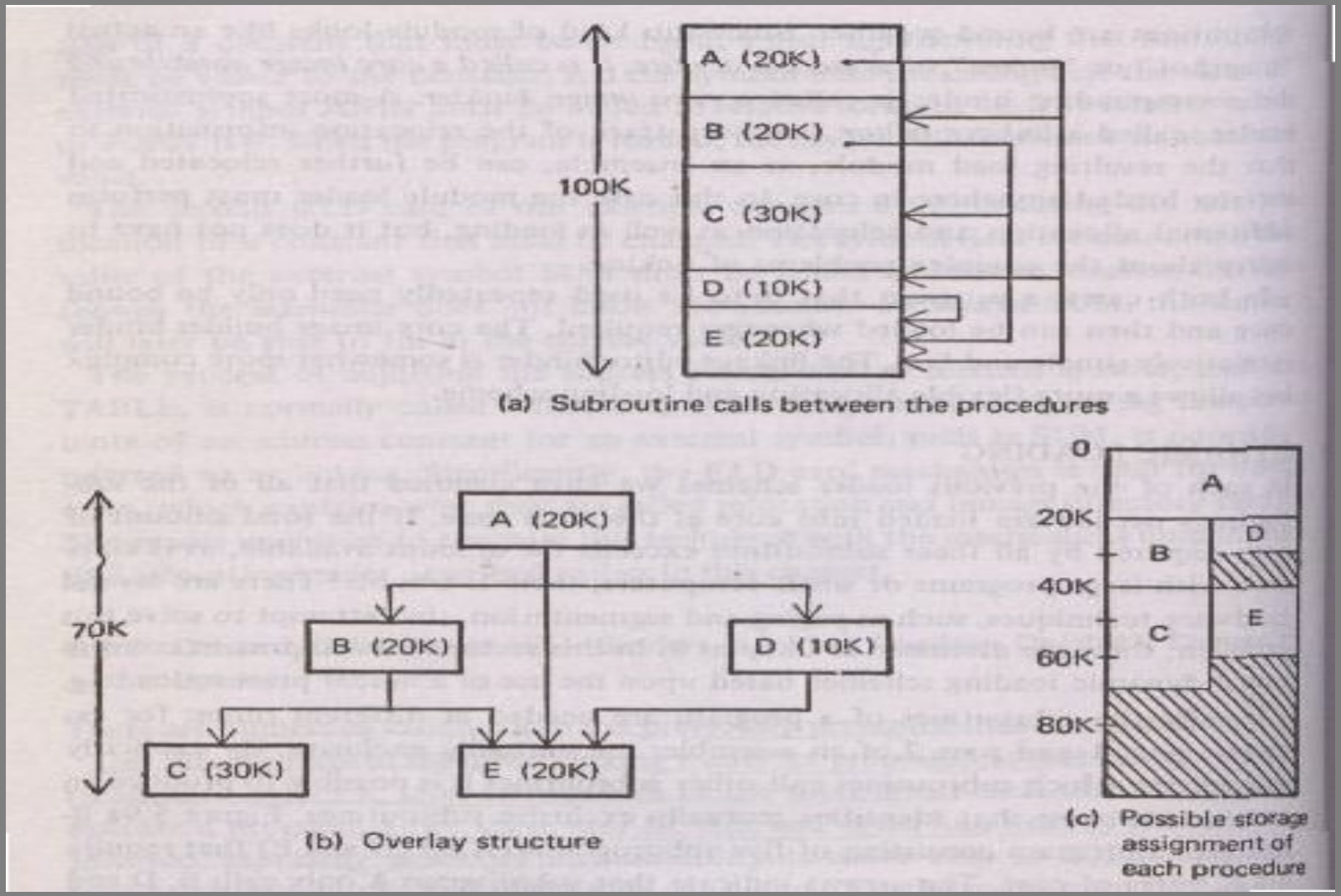
Overlay Structures

- What is Overlay?
  - Part of a program which has same load origin, as some other part of program
- When is it used?
  - When system does not support virtual memory

  And static linking/loading requires all subroutines to be in main memory at the same time

  And total memory required exceeds the amount available

8. Dynamic loading



9. Dynamic linking

- Loading and linking of external references are postponed until execution time
- Assembler produces text, binding and relocation information from a source language deck.
- The loader loads only the main program.

Example of a Direct-linking loader

Source card reference	Relative address		Sample program (source deck)		
1	0	PG1	START		(a) Procedure PG1
2			ENTRY	PG1ENT1,PG1ENT2	
3			EXTRN	PG2ENT1,PG2	
4	20	PG1ENT1	=		
5	30	PG1ENT2	=		
6	40		DC	A(PG1ENT1)	
7	44		DC	A(PG1ENT2+15)	
8	48		DC	A(PG1ENT2-PG1ENT1-3)	
9	52		DC	A(PG2)	
10	56		DC	A(PG2ENT1+PG2-PG1ENT1+4)	
11			END		
12	0	PG2	START		(b) Procedure PG2
13			ENTRY	PG2ENT1	
14			EXTRN	PG1ENT1,PG1ENT2	
15	16	PG2ENT1	=		
16	24		DC	A(PG1ENT1)	
17	28		DC	A(PG1ENT2+15)	
18	32		DC	A(PG1ENT2-PG1ENT1-3)	
19			END		

FIGURE 5.12 Sample procedures PG1 and PG2

Example of Direct-linking loader

ESD cards

Source card reference	Name	Type	ID	Relative address	Length
1	PG1	SD	01	0	60
2	PG1ENT1	LD	—	20	—
2	PG1ENT2	LD	—	30	—
3	PG2	ER	02	—	—
3	PG2ENT1	ER	03	—	—

TXT cards
(only the interesting ones, i.e. those involving address constants)

Source card reference	Relative address	Contents	Comments
6	40-43	20	
7	44-47	45	= 30 + 15
8	48-51	7	= 30-20-3
9	52-55	0	unknown to PG1
10	56-59	-16	= -20 + 4

RLD cards

Source card reference	ESD ID	Length (bytes)	Flag + or −	Relative address
6	01	4	+	40
7	01	4	+	44
9	02	4	+	52
10	03	4	+	56
10	02	4	+	56
10	01	4	−	56

FIGURE 5.13 Object deck program PG1

Example of a Direct-linking loader

ESD cards

Source card reference	Name	Type	ID	ADDR	Length
12	PG2	SD	01	0	36
13	PG2ENT1	LD	—	16	—
14	PG1ENT1	ER	02	—	—
14	PG1ENT2	ER	03	—	—

TXT cards
(only the interesting ones)

Source card reference	Relative address	Contents
16	24-27	0
17	28-31	15
18	32-25	-3

RLD cards

Source card reference	ESD ID	Length flag (bytes)	Flag + or −	Relative address
16	02	4	+	24
17	03	4	+	28
18	03	4	+	32
18	02	4	−	32

FIGURE 5.14 Object deck program PG2

TRUE or FALSE

- 1.In absolute loader, linking is done by programmer.
- 2. In compile and go loader linking is performed by loader.
- 3 loader loads & executes object program.

Comment

- Static linking Dynamic linking
- Static binding leads to more efficient execution of a program than dynamic bindings.
- Why are library routines usually relocatable ?
- What would happen if these routines are made
- Non relocatable?