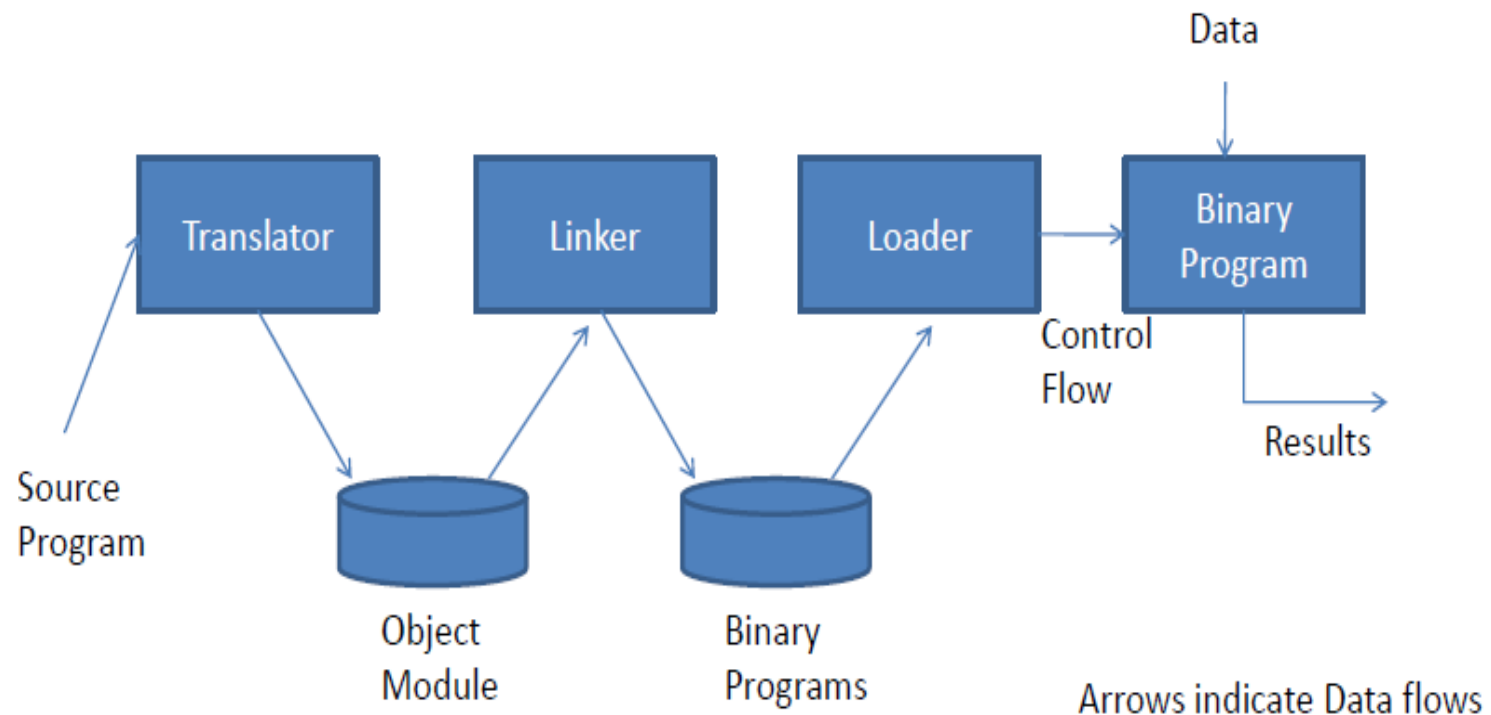


Linkers

Linker

- **Steps for Execution:**

1. **Translation** of program
2. **Linking** of program with other programs needed for its execution.
3. **Relocation** of the program to execute from the specific memory area allocated to it.
4. **Loading** of the program in the memory for the purpose of execution.





Linkers

- Links with other programs needed for its execution
- Processes a set of object modules to produce a ready-to-execute program form called binary program
- Loader loads this program into memory for execution



Linker

- A Linker is a system program that combines the code of a target program with codes of other programs and library routines



Object Module and Binary Program

- Object Module:
 - Contains target code (Machine language) of the program and information about other programs and library routines that it needs to invoke during its execution
- Binary Program:
 - Target code and other program routines combined together to form Binary program

Addresses

Address of a program entity may vary at different times:

1. Translation time address: Address assigned by translator (ORIGIN or START)
2. Linked address: Address assigned by linker
3. Load time address: Address assigned by loader

Addresses

Origin of a program may have to be changed by linker or loader because

1. Same set of translated addresses may have been used in different object modules, resulting in memory conflicts
2. OS MM module may require that a program should be allocated specific area of memory

Example:

	Statement	Address	Code
	START 500		
	ENTRY TOTAL		
	EXTRN MAX, ALPHA		
	READ A	500)	+ 09 0 540
LOOP		501)	
	.		
	.		
	MOVER AREG, ALPHA	518)	+ 04 1 000
	BC ANY, MAX	519)	+ 06 6 000
	.		
	.		
	BC LT, LOOP	538)	+ 06 1 501
	STOP	539)	+ 00 0 000
A	DS 1	540)	
TOTAL	DS 1	541)	
	END		

The translated
origin

The Translated Time
Address of LOOP

Relocation concept

- Program relocation is the process of modifying the addresses used in the address sensitive instructions of a program such that the program can execute correctly from the designated area of memory
- If linked origin \neq translated origin, relocation must be performed by the linker
- If load origin \neq linked origin, relocation must be performed by the loader



Sample Assembly Language Program and generated code (Program P)

```
START      500
ENTRY      TOTAL
EXTERN     MAX,ALPHA
READ       A           500 +400)  09 00  540+400=940
LOOP                               501+400)  901
-----
MOVER      AREG, ALPHA  518+400)  918           04 01 000
BC         ANY, MAX     519+400)  919           07 06 000
-----
BC         LT, LOOP     538)           07 01 501
STOP                               539)           00 00 000
A          DS          1           540)
TOTAL     DS          1           541)
END
```



Performing Relocation

t_origin_p - translated origin of program P

l_origin_p - linked origin of program P

t_{symb} - translation time address of a symbol
 $symb$

l_{symb} - link time address of a symbol $symb$

Relocation factor of P is defined as

$$relocation_factor_p = l_origin_p - t_origin \quad \text{-----1}$$

Performing Relocation

$$relocation_factor_p = l_origin_p - t_origin \quad \text{-----1}$$

$$t_{symb} = t_origin_p + d_{symb} \quad \text{-----2}$$

$$l_{symb} = l_origin_p + d_{symb} \quad \text{-----3}$$

From 1 and 3

$$l_{symb} = t_origin_p + relocation_factor_p + d_{symb} \quad \text{-----4}$$

From 2 and 4

$$l_{symb} = t_{symb} + relocation_factor_p \quad \text{-----5}$$



For Program P:

- Translated origin is 500
- Suppose $I_origin = 900$
- $Relocation_factor = 900 - 500 = 400$
- Relocation will be performed for instructions with translated time address 500 and 538
 - For instruction with translated time address 500:
address 540 in operand field will change to $(540 + 400) = 940$
 - For instruction with translated time address 538:
address 501 in operand field will change to $(501 + 400) = 901$

Linking

- Program interacts with another program unit using its instructions & data in its own instructions
- Public definitions & external references required
 - ENTRY: Public definitions
 - A symbol defined in a program unit that may be referenced in other program unit.
 - EXTRN: External references
 - A reference to a symbol that is not defined in the program unit containing the reference (defined in other program)
- Linking is the process of binding an external reference to the correct link time address



Program Q

START 200

ENTRY ALPHA

--

ALPHA DS 25 231) 00 00 025

END

Linkers

Object module contains all info necessary to relocate & link program with other programs

1. Header : has translated origin, size & execution start address
2. Program : has machine code
3. Relocation table (RELOCTAB) : each entry contains translated address of an address sensitive instruction
4. Linking table (LINKTAB): contains PD/EXT symbols

Linker generates the linked addresses of all the symbols & instructions



RELOCTAB and LINKTAB

- Program P:
- RELOCTAB:
 - 500) 09 00 540
 - 538) 07 01 501
- LINKTAB:
 - TOTAL PD
 - MAX EXT
 - ALPHA EXT



RELOCTAB and LINKTAB

- Program Q:
- LINKTAB:
 - ALPHA PD

NTAB:

Symbol	Linked addr
P	900
TOTAL	941
Q	942
ALPHA	973

Linking of Program P and Q

Program P:

```

START      500
            ENTRY      TOTAL
            EXTERN      MAX,ALPHA
            READ        A
            LOOP
            ----
            MOVER      AREG, ALPHA
            BC ANY, MAX
            -----
            BC LT, LOOP
            STOP
A          DS 1
TOTAL     DS      1
            END

```

900) 09 00 940
901)
918) 04 01 973
919) 07 06 000
938) 07 01 901
939) 00 00 000
940)
941)

Program Q:

```

START      200
            ENTRY      ALPHA
            --
ALPHA     DS      25
            END

```

942)
973) 00 00 025



Self-Relocating Programs

Programs can be classified into

1. Non relocatable programs

- cannot be executed in any memory area other than its translated origin

- due to lack of information pertaining to address sensitive instructions in program

2. Relocatable programs

- Has info available related to address sensitive instructions in program

3. Self-relocating programs



Self-Relocating Programs

- Performs relocation of its own address sensitive instructions
- 2 provisions for this
 1. Table containing address sensitive instructions exists as part of program
 2. Relocating logic: Code to perform relocation of address sensitive instructions also exists as part of program
- Can execute in any area of memory
- Useful in time sharing operating systems



Static Link Libraries

- Is the process of copying all library modules used in the program into the final executable image.
- This is performed by the linker and it is done as the last step of the compilation process.
- The linker combines library routines with the program code in order to resolve external references, and to generate an executable image suitable for loading into memory.
- When the program is loaded, the operating system places into memory a single file that contains the executable code and data.
- This statically linked file includes both the calling program and the called program.
- Statically linked files are significantly larger in size because external programs are built into the executable files.

Static Link Libraries

- If any of the external programs change then they have to be recompiled and re-linked again else the changes won't reflect in existing executable file.
- Statically linked program takes constant load time every time it is loaded into the memory for execution
- Programs that use statically-linked libraries are usually faster
- In statically-linked programs, all code is contained in a single executable module. Therefore, they never run into compatibility issues.

Dynamic Link Libraries

- In dynamic linking the names of the external libraries (shared libraries) are placed in the final executable file while the actual linking takes place at run time when both executable file and libraries are placed in the memory.
- Several programs use a single copy of an executable module.
- Is performed at run time by the operating system
- Only one copy of shared library is kept in memory. This significantly reduces the size of executable programs, thereby saving memory and disk space
- Individual shared modules can be updated and recompiled. This is one of the greatest advantages dynamic linking offers.



Dynamic Link Libraries

- Load time might be reduced if the shared library code is already present in memory.
- Programs that use shared libraries are usually slower than those that use statically-linked libraries.
- Dynamically linked programs are dependent on having a compatible library. If a library is changed (for example, a new release may change a library), applications might have to be reworked to be made compatible with the new version of the library. If a library is removed from the system, programs using that library will no longer work.