



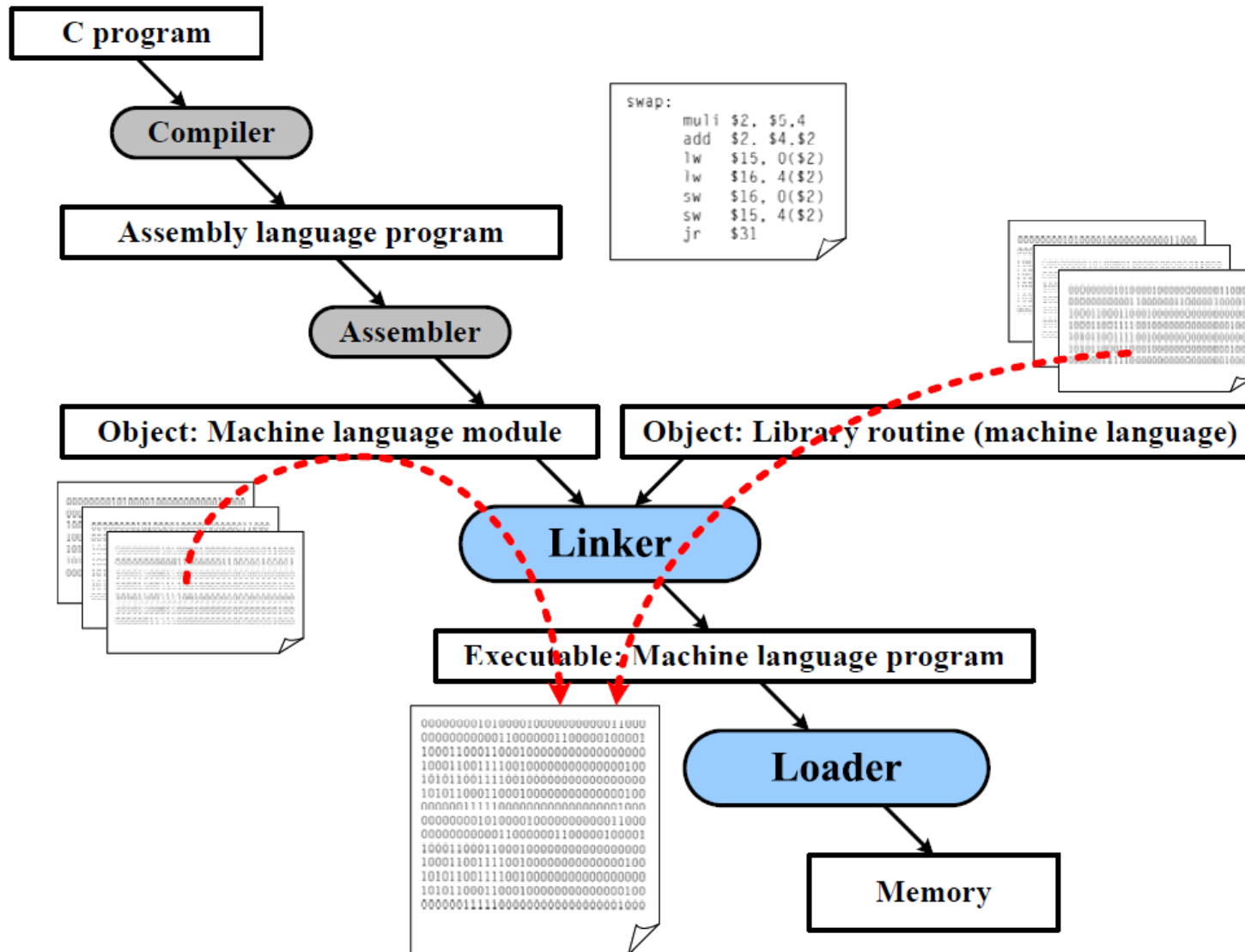
CS202 – System Software

Dr. Manish Khare

Linker and Loader




Need for Linker and Loader





➤ To execute an object program, we need:

- **Relocation** - which modifies the object program so that it can be loaded at an address different from the location originally specified
- **Linking** - which combines two or more separate object programs and supplies the information needed to allow references between them
- **Loading and Allocation** - which allocates memory location and brings the object program into memory for execution

- 
- The system software which performs linking operation is called **linker**.
 - The system software which loads the object program into memory and starts its execution is called **loader**.
 - Linkers and loaders perform several related but conceptually separate actions.

Basic Loader Function

- Fundamental functions of a loader are **Bringing an object program into memory and starting its execution.**
- Two basic loader designs
 - 1. Absolute Loader
 - 2. Bootstrap Loader

Design of Absolute Loader

- An absolute loader is a loader that places absolute code into main memory beginning with the initial address(absolute address) assigned by the assembler.
- No address manipulation is performed.
- That is there is no need for relocation and linking because the program will be loaded into the location specified in the program.

Loading of absolute program

```

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


```

(a) Object program

Memory address	Contents			
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮
0FF0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
1000	14103348	20390010	36281030	30101548
1010	20613C10	0300102A	0C103900	102D0C10
1020	36482061	0810334C	0000454F	46000003
1030	000000xx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮
2030	xxxxxxxx	xxxxxxxx	xx041030	001030E0
2040	205D3020	3FD8205D	28103030	20575490
2050	392C205E	38203F10	10364C00	00F10010
2060	00041030	E0207930	20645090	39DC2079
2070	2C103638	20644C00	0005xxxx	xxxxxxxx
2080	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮

← COPY

(b) Program loaded in memory



➤ For a simple absolute loader, all functions are accomplished in a single pass as follows:



- 1) The **Header record** of object programs is checked to verify that the correct program has been presented for loading.
- 2) As each **Text record** is read, the object code it contains is moved to the indicated address in memory.
- 3) When the **End record** is encountered, the loader jumps to the specified address to begin execution of the loaded program.



➤ Algorithm for absolute loader

begin

 read Header record

 verify program name and length

 read first Text record

 while record type \neq E

 begin

 //if object code is in character form, convert it into internal representation

 move object code to specified location in memory

 read next object program record

 end

 jump to address specified in End record

end

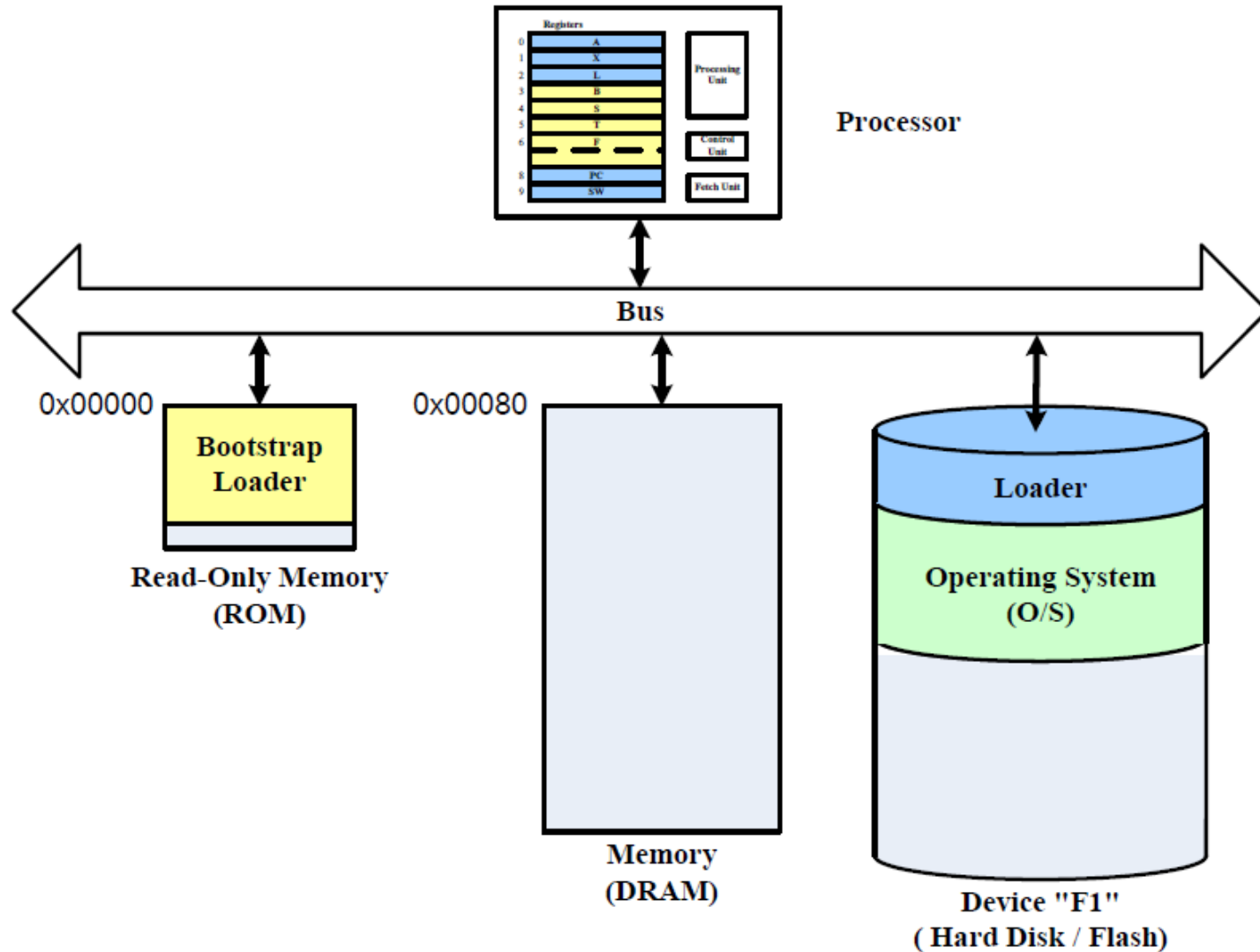
Hex code	ASCII character	Hex Code	ASCII character	Hex code	ASCII character	Hex code	ASCII character
00	NUL	20	SP	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	\$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(48	H	68	h
09	HT	29)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[7B	{
1C	FS	3C	<	5C	\	7C	
1D	GS	3D	=	5D]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL



➤ Advantages and disadvantages of absolute loader

- The advantage of absolute loader is that it is simple and efficient, but the need for programmer to specify the actual address restricts the flexibility.
- As a result we cannot run several independent programs together, sharing memory between them.
- Another disadvantage is that it is difficult to use subroutine libraries while using an absolute loader.

A Simple Bootstrap Loader



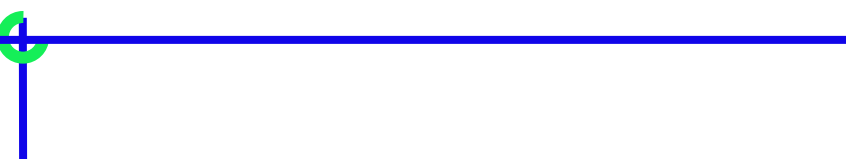
A Simple Bootstrap Loader

- Given an idle computer with no program in memory, how do we get things started?
- Two solutions are there.
 - 1. On some computers, an absolute loader program is permanently resident in a read-only memory (ROM). When some hardware signal occurs, the machine begins to execute this ROM program. This is referred to as a bootstrap loader.
 - 2. On some computers, there's a built-in hardware which read a fixed-length record from some device into memory at a fixed location. After the read operation, control is automatically transferred to the address in memory. .
- When a computer is first turned on or restarted, a special type of absolute loader, called a **bootstrap loader**, is executed.
- This bootstrap loader loads the first program to be run by the computer – usually an operating system.

Working of a SIC Bootstrap loader

SIC uses the above mentioned second method.

- The bootstrap begins at address 0 in the memory of the machine.
- It loads the operating system at address 80.
- Each byte of object code to be loaded is represented on device F1 as two hexadecimal digits just as it is in a Text record of a SIC object program.
- The object code from device F1 is always loaded into consecutive bytes of memory, starting at address 80.
- The main loop of the bootstrap keeps the address of the next memory location to be loaded in register X.
- After all of the object code from device F1 has been loaded, the bootstrap jumps to address 80, which begins the execution of the program that was loaded.
- Much of the work of the bootstrap loader is performed by the subroutine GETC.

- 
- GETC is used to read and convert a pair of characters from device F1 representing
 - 1 byte of object code to be loaded. For example, two bytes = C “D8” → ‘4438’H converting to one byte ‘D8’H.
 - The resulting byte is stored at the address currently in register X, using STCH instruction that refers to location 0 using indexed addressing.
 - The TIXR instruction is then used to add 1 to the value in X.

Bootstrap Loader for SIC/XE

- This bootstrap main function reads object code from device F1 and enters it into memory starting at address 80 (hexadecimal).
- After all of the code from dev F1 has been seen entered into memory, the bootstrap executes a jump to address 80 to begin execution of the program just loaded.
- Register X contains the next address to be loaded.

BOOT	START	0	
	CLEAR	A	CLEAR REGISTER A TO ZERO
	LDX	#128	INITIALIZE REGISTER X TO HEX 80
LOOP	JSUB	GETC	READ HEX DIGIT FROM PROGRAM BEING LOADED
	RMO	A, S	SAVE IN REGISTER S
	SHIFTL	S, 4	MOVE TO HIGH ORDER 4 BITS OF BYTE
	JSUB	GETC	GET NEXT HEX DIGIT
	ADDR	S, A	COMBINE DIGITS TO FORM ONE BYTE
	STCH	0, X	STORE AT ADDRESS IN REGISTER X
	TIXR	X	ADD 1 TO MEMORY ADDRESS BEING LOADED
	JUMP	LOOP	LOOP UNTIL END OF INPUT IS REACHED

- GETC subroutine read one character from input device and convert it from ASCII code to hexadecimal digit value. The converted digit value is returned in register A. When an end of file is read, control is transferred to the starting address (hex 80)

GETC	TD	INPUT	TEST INPUT DEVICE
	JEQ	GETC	LOOP UNTIL READY
	RD	INPUT	READ CHARACTER
	COMP	#4	IF CHARACTER IS HEX 04 (END OF FILE) ,
	JEQ	80	JUMP TO START OF PROGRAM JUST LOADED
	COMP	#48	COMPARE TO HEX 30 (CHARACTER '0')
	JLT	GETC	SKIP CHARACTERS LESS THAN '0'
	SUB	#48	SUBTRACT HEX 30 FROM ASCII CODE
	COMP	10	IF RESULT IS LESS THAN 10 , CONVERSION IS
	JLT	RETURN	COMPLETE. OTHERWISE, SUBTRACT 7 MORE
	SUB	#7	(FOR HEX DIGITS 'A' THROUGH 'F')
RETURN	RSUB		RETURN TO CALLER
INPUT	BYTE	X'F1'	CODE FOR INPUT DEVICE
			END LOOP


MACHINE DEPENDENT LOADER FEATURES

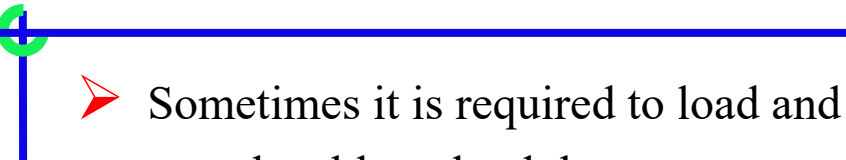
➤ The features of loader that depends on machine architecture are called machine dependent loader features. It includes:

- 1. Program Relocation
- 2. Program Linking

Program Relocation (Relocating Loader)

- The absolute loader has several disadvantages. One of the most obvious is the need for the programmer to specify the actual address at which it will be loaded into memory.
- On a simple computer with a small memory the actual address at which the program will be loaded can be specified easily.
- On a larger and more advanced machine, we often like to run several independent programs together, sharing memory between them. We do not know in advance where a program will be loaded. Hence we write relocatable programs instead of absolute ones.

- 
- Writing absolute programs also makes it difficult to use **subroutine libraries** efficiently. This could not be done effectively if all of the subroutines had preassigned absolute addresses.
 - The need for program relocation is an indirect consequence of the change to larger and more powerful computers. The way relocation is implemented in a loader is also dependent upon machine characteristics.

- 
- Sometimes it is required to load and run several programs at the same time. The system must be able to load these programs wherever there is place in the memory. Therefore the exact starting address is not known until the load time.
 - In an absolute program the starting address to which the program has to be loaded is mentioned in the program itself using the START directive. So the address of every instruction and labels are known while assembling itself.
 - **Loaders that has the capability to perform relocation are called relocating loaders or relative loaders.**
 - There are two methods for specifying relocation in object program
 - 1. Modification Record (for SIC/XE)
 - 2. Relocation Bit (for SIC)