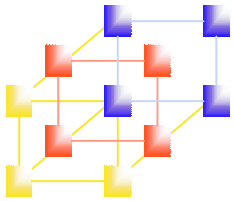
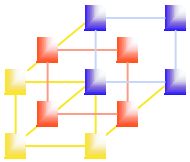


Chapter 3 Loaders and Linkers

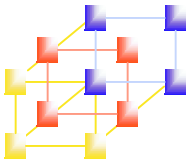
-- Basic Loader Functions



Three processes to run an object program

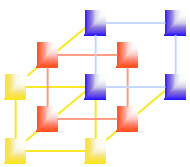
- Loading
  - Brings object program into memory
- Relocation
  - Modifies the object program so that it can be loaded at an address different from the location originally specified
- Linking
  - Combines two or more separate object programs and supplies information needed to allow cross-references.
- "Loader and linker" may be a single system program
  - Loader: loading and relocation
  - Linker: linking

} Loader



Absolute loader

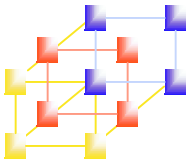
- No linking and relocation needed
- Records in object program perform
  - Header record
    - Check the Header record for program name, starting address, and length (available memory)
  - Text record
    - Bring the object program contained in the Text record to the indicated address
  - End record
    - Transfer control to the address specified in the End record



Loading an absolute program
Figure 3.1, pp. 125

```
HCOPY  00100000107A
T0010001E141033482039001036281030301015482061 3C100300102A0C103900102D
T00101E150C10364820610810334C0000454F46000003000000
T0020391E041030001030E0205D30203FD8205D2810303020575490392C205E38203F
T0020571C1010364C0000F001000041030E02079302064509039DC20792C1036
T002073073820644C000005
E001000
```

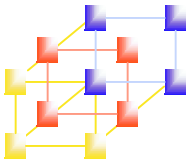
(a) Object program



Loading an absolute program
Figure 3.1, pp. 125

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
.	.	.	.	.
.	.	.	.	.

No text record



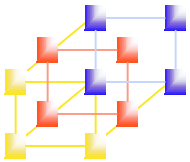
Algorithm for an absolute loader Figure 3.2, pp. 126

```
begin
   read Header record
   verify program name and length
   read first Text record
   while record type != 'E' do
            begin
               {if object code is in character form, convert 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
```

Algorithm for an absolute loader

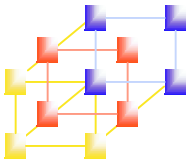
Most machines store object codes in binary form

- Less space and loading time
- Not good for reading



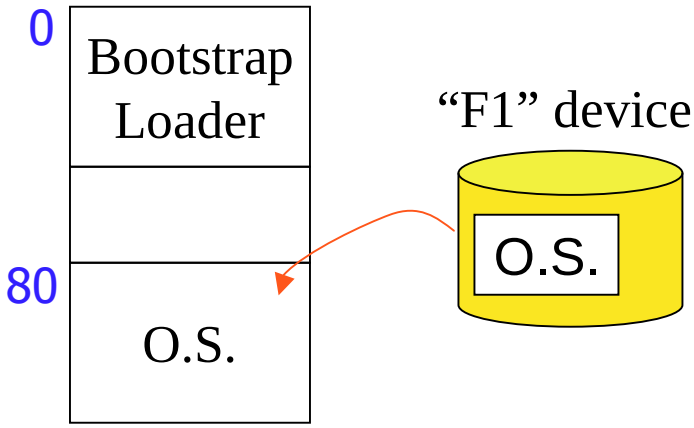
Object Code Representation

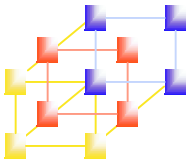
- Character form (e.g. Figure 3.1 (a))
  - Each byte of assembled code is given using its hexadecimal representation in character form
  - Easy to read by human beings
- Binary form
  - Each byte of object code is stored as a single byte
  - Most machine store object programs in a binary form



A simple bootstrap loader

- Bootstrap Loader (usually in ROM)
  - When a computer is first tuned on or restarted, a special type of absolute loader, the bootstrap loader loads the first program (usually O.S.) to be run into memory
- SIC bootstrap loader
  - The bootstrap itself begins at address 0
  - It loads the OS starting address 0x80
  - No header record or control information, the object code is consecutive bytes of memory
  - After load the OS, the control is transferred to the instruction at address 80.



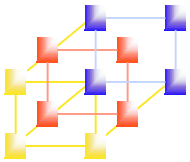


Algorithm for SIC/XE bootstrap loader

X ← 0x80 (the address of the next memory location to be loaded)
Loop until end of input
    A ← GETC (and convert from ASCII character code to the hexadecimal digit)
    save the value in the high-order 4 bits of S
    A ← GETC
    combine the value to form one byte A ← (A+S)
    (X) ← (A) (store one char.)
    X ← X + 1
End of loop

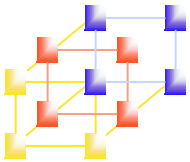
GETC A ← read one character from device F1
    if (A = 0x04) then jump to 0x80
    if A<48 then goto GETC
    A ← A-48 (0x30)
    if A<10 then return
    A ← A-7
    return

ASCII value of
`0~9 : 0x30~39`
`A~F : 0x41~46`



Bootstrap loader for SIC/XE -- Figure 3.3, pp. 128

```
BOOT    START   0           BOOTSTRAP LOADER FOR SIC/XE
.
. THIS BOOTSTRAP READS OBJECT CODE FROM DEVICE F1 AND ENTERS IT
. INTO MEMORY STARTING AT ADDRESS 80 (HEXADECIMAL). AFTER ALL OF
. THE CODE FROM DEVF1 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.
.
        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,X     ADD 1 TO MEMORY ADDRESS BEING LOADED
        J       LOOP    LOOP UNTIL END OF INPUT IS REACHED
```



Bootstrap loader for SIC/XE -- Figure 3.3, pp. 128

```
.
. SUBROUTINE TO 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 HCARACTERS 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
```