CS2422 Assembly Language & System Programming

December 26, 2006

Today's Topics

- 3.1 Basic Loader Functions
  - Design of an Absolute Loader
  - A Simple Bootstrap Loader
- 3.2 Machine-Dependent Loader Features
  - Relocation
  - Program Linking

Introduction

- To execute an object program, we needs
  - 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

Overview of Chapter 3

- Type of loaders
  - Assemble-and-go loader
  - Absolute loader (bootstrap loader)
  - Relocating loader (relative loader)
  - Direct linking loader
- Design options
  - Linkage editors
  - Dynamic linking
  - Bootstrap loaders

Assemble-and-go Loader

- Characteristic
  - The object code is stored in memory after assembly
  - Single JUMP instruction
- Advantage
  - Simple, developing environment
- Disadvantage
  - Whenever the assembly program is to be executed, it has to be assembled again
  - Programs have to be coded in the same language

Design of an Absolute Loader

- Absolute Program
  - Advantage
    - Simple and efficient
  - Disadvantage
    - The need for programmer to specify the actual address
    - Difficult to use subroutine libraries
- Program Logic

Fig. 3.2 Algorithm for an absolute loader

```
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
```

Loading of an absolute program (1/2)
Fig. 3.1(a)

```
HCOPY  001000001078
T001000 1E 141033482039001036281030301015482061 3C1003 00102A 0C1039 00102D
T00101E 15 0C1036482061 0810334C0000 454F46 000003 000000
T002039 1E 041030001030 E0205D 30203F D8205D 281030 302057 549039 2C205E 38203F
T002057 1C 1010364C0000 F1001000 41030E 02079 302064 509039 DC2079 2C1036
T002073 07 3820644C000005
E001000
```

(a) Object program

Loading of an absolute program (2/2)
Fig. 3.1(b)

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	← COPY
⋮	⋮	⋮	⋮	⋮	
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	
⋮	⋮	⋮	⋮	⋮	

(b) Program loaded in memory

Figure 3.1 Loading of an absolute program.

Object Code Representation

- Fig. 3.1(a)
  - Each byte of assembled code is given using its hexadecimal representation in character form
  - Easy to read by human beings
- In general
  - Each byte of object code is stored as a single byte
  - Most machine store object programs in a binary form
  - We must be sure that our file and device conventions do not cause some of the program bytes to be interpreted as control characters

A Simple Bootstrap Loader

- Bootstrap Loader
  - When a computer is first tuned on or restarted, a special type of absolute loader, called *bootstrap loader* is executed
  - This bootstrap loads the first program to be run by the computer -- usually an operating system
- Example (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

Bootstrap loader for SIC/XE

Begin
X=0x80 (the address of the next memory location to be loaded)
Loop
A←GETC (and convert it from the ASCII character code to the value of 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)
store the value (in A) to the address in register X
X←X+1
End

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

Fig. 3.3 Bootstrap loader for SIC/XE

```
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
.
. 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 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   RETURN TO CALLER
INPUT    BYTE      X'F1'    CODE FOR INPUT DEVICE
         END       LOOP
```

Figure 3.3 Bootstrap loader for SIC/XE.

Relocating Loaders

- Motivation
  - 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
- Two methods for specifying relocation
  - Modification record (Fig. 3.4, 3.5)
  - Relocation bit (Fig. 3.6, 3.7)
    - Each instruction is associated with one relocation bit
    - These relocation bits in a Text record is gathered into bit masks

Modification Record

- For complex machines
- Also called RLD specification
  - Relocation and Linkage Directory

Modification record
col 1: M
col 2-7: relocation address
col 8-9: length (halfbyte)
col 10: flag (+/-)
col 11-17: segment name

Fig. 3.4: An SIC/XE Example (from Fig. 2.6)

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	

```
115          .       READ RECORD INTO BUFFER
120          .
125   1036   RDREC    CLEAR  X           B410
130   1038            CLEAR  A           B400
132   103A            CLEAR  S           B440
133   103C           +LDT    #4096       75101000
135   1040   RLOOP    TD     INPUT       E32019
140   1043            JEQ    RLOOP       332FFA
145   1046            RD     INPUT       DB2013
150   1049            COMPR  A,S         A004
155   104B            JEQ    EXIT        332008
160   104E            STCH   BUFFER,X    57C003
165   1051            TIXR   T           B850
170   1053            JLT    RLOOP       3B2FEA
175   1056   EXIT     STX    LENGTH      134000
180   1059            RSUB               4F0000
185   105C   INPUT    BYTE   X'F1'       F1
195          .
200          .       WRITE RECORD FROM BUFFER
205          .
210   105D   WRREC    CLEAR  X           B410
212   105F            LDT    LENGTH      774000
215   1062   WLOOP    TD     OUTPUT      E32011
220   1065            JEQ    WLOOP       332FFA
225   1068            LDCH   BUFFER,X    53C003
230   106B            WD     OUTPUT      DF2008
235   106E            TIXR   T           B850
             ...(omitted)
```

Fig. 3.5

```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200 74B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T001036 1DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T001053 1D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T001070 073B2FEF4F000005
M000007 05+COPY
M000014 05+COPY
M000027 05+COPY
E000000
```

Figure 3.5 Object program with relocation by Modification records.

Relocation Bit

- For simple machines
- Relocation bit
  - 0: No modification is necessary
  - 1: Modification is needed

Text record
col 1: T
col 2-7: starting address
col 8-9: length (byte)
col 10-12: relocation bits
col 13-72: object code

- Twelve-bit mask is used in each Text record
  - Since each text record contains less than 12 words
  - Unused words are set to 0
  - Any value that is to be modified during relocation must coincide with one of these 3-byte segments
    - e.g. line 210

Figure 3.6: Relocatable Program for SIC

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	140033
15	0003	CLOOP	JSUB	RDREC	481039
20	0006		LDA	LENGTH	000036
25	0009		COMP	ZERO	280030
30	000C		JEQ	ENDFIL	300015
35	000F		JSUB	WRREC	481061
40	0012		J	CLOOP	3C0003
45	0015	ENDFIL	LDA	EOF	00002A
50	0018		STA	BUFFER	0C0039
55	001B		LDA	THREE	00002D
60	001E		STA	LENGTH	0C0036
65	0021		JSUB	WRREC	481061
70	0024		LDL	RETADR	080033
75	0027		RSUB		4C0000
80	002A	EOF	BYTE	C'EOF'	454F46
85	002D	THREE	WORD	3	000003
90	0030	ZERO	WORD	0	000000
95	0033	RETADR	RESW	1	
100	0036	LENGTH	RESW	1	
105	0039	BUFFER	RESB	4096	

```
110           .
115           .        SUBROUTINE TO READ RECORD INTO BUFFER
120           .
125   1039    RDREC    LDX     ZERO          040030
130   103C             LDA     ZERO          000030
135   103F    RLOOP    TD      INPUT         E0105D
140   1042             JEQ     RLOOP         30103D
145   1045             RD      INPUT         D8105D
150   1048             COMP    ZERO          280030
155   104B             JEQ     EXIT          301057
160   104E             STCH    BUFFER,X      548039
165   1051             TIX     MAXLEN        2C105E
170   1054             JLT     RLOOP         38103F
175   1057    EXIT     STX     LENGTH        100036
180   105A             RSUB                  4C0000
185   105D    INPUT    BYTE    X'F1'         F1
190   105E    MAXLEN   WORD    4096          001000
195           .
200           .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205           .
210   1061    WRREC    LDX     ZERO          040030
215   1064    WLOOP    TD      OUTPUT        E01079
220   1067             JEQ     WLOOP         301064
225   106A             LDCH    BUFFER,X      508039
230   106D             WD      OUTPUT        DC1079
235   1070             TIX     LENGTH        2C0036
240   1073             JLT     WLOOP         381064
245   1076             RSUB                  4C0000
250   1079    OUTPUT   BYTE    X'05'         05
255                    END     FIRST
```

Fig. 3.7

```
HCOPY  000000001 07A
T0000001EFFC140033481039000036280030300015481061 3C0003 00002A0C0039 00002D
T00001E15E000C0036481061080033 4C0000454F46000003000000
T0001391EFFC040030000030E0105D30103FD8105D280030301057548039 2C105E38103F
T0010570A80010003 64C0000F1001000
T00106119FE0040030E01079301064508039DC10792C00363810644C000005
E000000
```

Figure 3.7 Object program with relocation by bit mask.

Program Linking

- Goal
  - Resolve the problems with EXTREF and EXTDEF from different control sections (sec 2.3.5)
- Example
  - Program in Fig. 3.8 and object code in Fig. 3.9
  - Use modification records for both relocation and linking
    - Address constant
    - External reference

Fig. 3.8-1

```
Loc     Source statement                          Object code

0000    PROGA   START   0
                EXTDEF  LISTA,ENDA
                EXTREF  LISTB,ENDB,LISTC,ENDC
                .
                .
                .
0020    REF1    LDA     LISTA                     03201D
0023    REF2   +LDT     LISTB+4                   77100004
0027    REF3    LDX     #ENDA-LISTA               050014
                .
                .
                .
0040    LISTA   EQU     *
                .
                .
0054    ENDA    EQU     *
0054    REF4    WORD    ENDA-LISTA+LISTC          000014
0057    REF5    WORD    ENDC-LISTC-10             FFFFF6
005A    REF6    WORD    ENDC-LISTC+LISTA-1        00003F
005D    REF7    WORD    ENDA-LISTA-(ENDB-LISTB)   000014
0060    REF8    WORD    LISTB-LISTA               FFFFC0
                END     REF1
```

Fig. 3.9-1

```
HPROGA 000000000063
DLISTA 000040ENDA  000054
RLISTB ENDB  LISTC ENDC
:
T000020 0A 03201D 77100004 050014
:
T000054 0F 000014 FFFFF6 00003F 000014 FFFFC0
M000024 05 +LISTB
M000054 06 +LISTC
M000057 06 +ENDC
M000057 06 -LISTC
M00005A 06 +ENDC
M00005A 06 -LISTC
M00005A 06 +PROGA
M00005D 06 -ENDB
M00005D 06 +LISTB
M000060 06 +LISTB
M000060 06 -PROGA
E000020
```

Figure 3.9 Object programs corresponding to Fig. 3.8.

Fig. 3.8-2

Loc		Source statement		Object code
0000	PROGB	START	0	
		EXTDEF	LISTB,ENDB	
		EXTREF	LISTA,ENDA,LISTC,ENDC	
		.		
		.		
		.		
0036	REF1	+LDA	LISTA	03100000
003A	REF2	LDT	LISTB+4	772027
003D	REF3	+LDX	#ENDA-LISTA	05100000
		.		
		.		
		.		
0060	LISTB	EQU	*	
		.		
		.		
0070	ENDB	EQU	*	
0070	REF4	WORD	ENDA-LISTA+LISTC	000000
0073	REF5	WORD	ENDC-LISTC-10	FFFFF6
0076	REF6	WORD	ENDC-LISTC+LISTA-1	FFFFFF
0079	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	FFFFF0
007C	REF8	WORD	LISTB-LISTA	000060
		END		

Figure 3.8 Sample programs illustrating linking and relocation.

Fig. 3.9-2

```
HPROGB 000000000007F
DLISTB 000060ENDB  000070
RLISTA ENDA  LISTC ENDC
.
.
T0000360B0310000077202705100000
.
.
T0000700F000000FFFFF6FFFFFFFFFF0000060
M00003705+LISTA
M00003E05+ENDA
M00003E05-LISTA
M00007006+ENDA
M00007006-LISTA
M00007006+LISTC
M00007306+ENDC
M00007306-LISTC
M00007606+ENDC
M00007606-LISTC
M00007606+LISTA
M00007906+ENDA
M00007906-LISTA
M00007C06+PROGB
M00007C06-LISTA
E
```

Fig. 3.8-3

Loc		Source statement		Object code
0000	PROGC	START	0	
		EXTDEF	LISTC,ENDC	
		EXTREF	LISTA,ENDA,LISTB,ENDB	
		.		
		.		
		.		
0018	REF1	+LDA	LISTA	03100000
001C	REF2	+LDT	LISTB+4	77100004
0020	REF3	+LDX	#ENDA-LISTA	05100000
		.		
		.		
		.		
0030	LISTC	EQU	*	
		.		
		.		
0042	ENDC	EQU	*	
0042	REF4	WORD	ENDA-LISTA+LISTC	000030
0045	REF5	WORD	ENDC-LISTC-10	000008
0048	REF6	WORD	ENDC-LISTC+LISTA-1	000011
004B	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000000
004E	REF8	WORD	LISTB-LISTA	000000
		END		

Figure 3.8 (*cont'd*)

Fig. 3.9-3

```
HPROGC 000000000051
DLISTC 000030ENDC  000042
RLISTA ENDA  LISTB ENDB
.
.
T000018 0C 03 10000077100004 05100000
.
.
T000042 0F 000030 000008 000011 000000000000
M000019 05+LISTA
M00001D 05+LISTB
M000021 05+ENDA
M000021 05-LISTA
M000042 06+ENDA
M000042 06-LISTA
M000042 06+PROGC
M000048 06+LISTA
M00004B 06+ENDA
M00004B 06-LISTA
M00004B 06-ENDB
M00004B 06+LISTB
M00004E 06+LISTB
M00004E 06-LISTA
E
```

Figure 3.9 (*cont'd*)

Program Linking Example

- Fig. 3.10
- Load address for control sections
  - PROGA 004000 63
  - PROGB 004063 7F
  - PROGC 0040E2 51
- Load address for symbols
  - LISTA: PROGA+0040=4040
  - LISTB: PROGB+0060=40C3
  - LISTC: PROGC+0030=4112
- REF4 in PROGA
  - ENDA-LISTA+LISTC=14+4112=4126
  - T0000540F000014FFFFF600003F000014FFFFC0
  - M00005406+LISTC

Fig. 3.10(a)

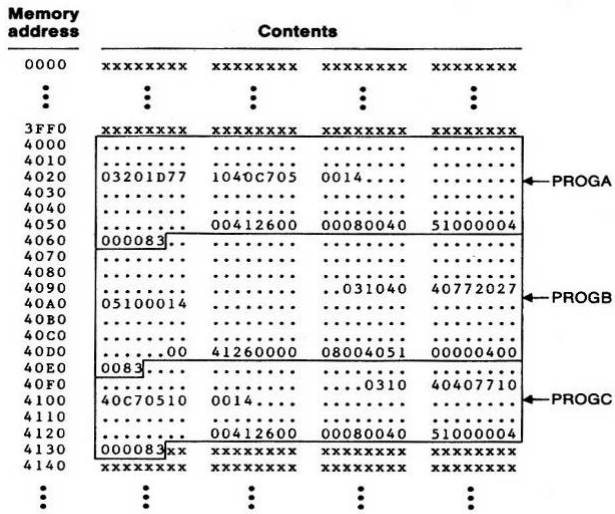

Figure 3.10(a) Programs from Fig. 3.8 after linking and loading.

Fig. 3.10(b)

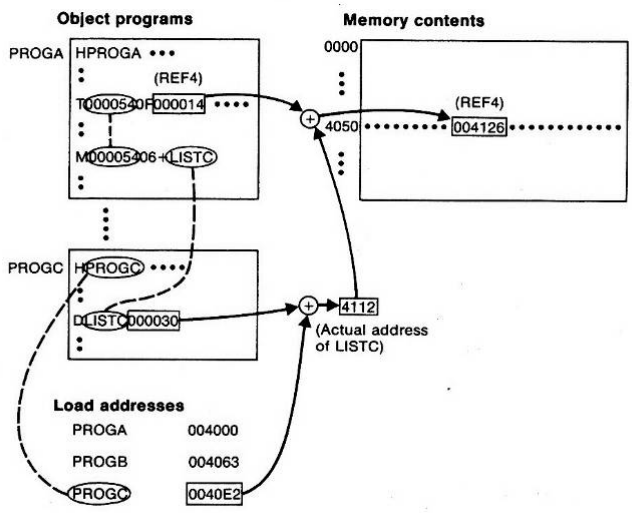

Figure 3.10(b) Relocation and linking operations performed on REF4 from PROGA.