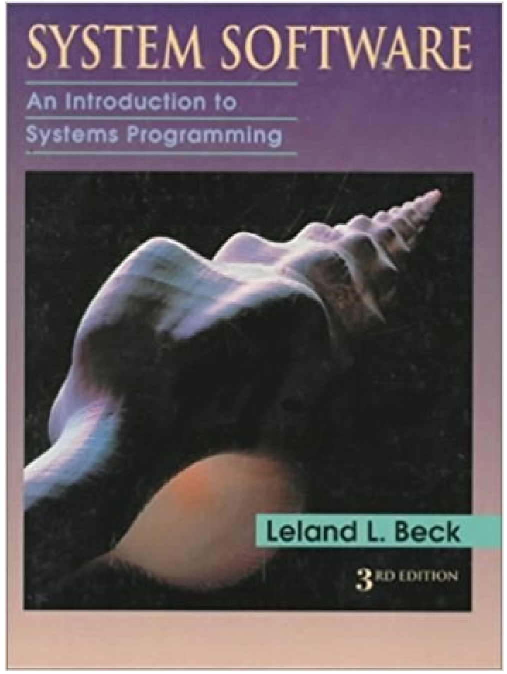
CSCI 30000
Systems Programming
Course Project



TA: Hongyan Zhou(zhou94@iu.edu)

Course Project: Introduction

- These slides will serve as a basic guide to help you get started
- This project will take a considerable amount of time to complete successfully
  - <u>If you have not already, please quickly make a plan with your group detailing how you will complete this project</u>
  - <u>There is no way you will be able to successfully complete this project if you put if off until the end of the semester</u>
- This project will enhance both your understanding of System Software and your skills as a programmer
- You need to start by reading the **first two chapters** of the textbook

Course Project: Goal

- **Design and Implement Project:** implement an assembler using **Java**, **C++, Perl, Tcl/Tk, others** (any one is OK)
  - I would suggest Java or Python, but you may choose whatever language reflects your team members' strengths
  - Irrespective of choice, you will be responsible for clearly outlining the details of how to successfully compile and run your code
- Implement an assembler which can process SIC/XE assembly programs and generate the corresponding object codes and object programs

Course Project: Overview

- In HW1 we saw how use an object code to calculate the corresponding SIC/XE instruction’s TA(target addresses), addressing modes, opcode and instruction format (**object code -> instruction details**)
- In this project you are tasked with discerning a SIC/XE program's instructions’ information and working to create the corresponding object codes **(instruction details -> object code)**

Course Project: Implementation Details

- Let us take a look at some assembler implementation details
- In an email you were given six sample SIC/XE assembly programs
    1. basic.txt
    2. functions.txt
    3. literals.txt
    4. program_blocks.txt
    5. control_sections.txt
    6. macros.txt
- You need to implement an assembler which can successfully process all these files
- Let us start by taking a closer look at functions.txt

Course Project: functions.txt

Source statement

```
COPY    START   0               COPY FILE FROM INPUT TO OUTPUT
FIRST   STL     RETADR          SAVE RETURN ADDRESS
        LDB     #LENGTH         ESTABLISH BASE REGISTER
        BASE    LENGTH
CLOOP  +JSUB    RDREC           READ INPUT RECORD
        LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
        COMP    #0
        JEQ     ENDFIL          EXIT IF EOF FOUND
       +JSUB    WRREC           WRITE OUTPUT RECORD
        J       CLOOP           LOOP
ENDFIL  LDA     EOF             INSERT END OF FILE MARKER
        STA     BUFFER
        LDA     #3              SET LENGTH = 3
        STA     LENGTH
       +JSUB    WRREC           WRITE EOF
        J       @RETADR         RETURN TO CALLER
EOF     BYTE    C'EOF'
RETADR  RESW    1
LENGTH  RESW    1               LENGTH OF RECORD
BUFFER  RESB    4096            4096-BYTE BUFFER AREA
.
.       SUBROUTINE TO READ RECORD INTO BUFFER
.
RDREC   CLEAR   X               CLEAR LOOP COUNTER
        CLEAR   A               CLEAR A TO ZERO
        CLEAR   S               CLEAR S TO ZERO
       +LDT     #4096
RLOOP   TD      INPUT           TEST INPUT DEVICE
        JEQ     RLOOP           LOOP UNTIL READY
        RD      INPUT           READ CHARACTER INTO REGISTER A
        COMPR   A,S             TEST FOR END OF RECORD (X'00')
        JEQ     EXIT            EXIT LOOP IF EOR
        STCH    BUFFER,X        STORE CHARACTER IN BUFFER
        TIXR    T               LOOP UNLESS MAX LENGTH
        JLT     RLOOP             HAS BEEN REACHED
EXIT    STX     LENGTH          SAVE RECORD LENGTH
        RSUB                    RETURN TO CALLER
INPUT   BYTE    X'F1'           CODE FOR INPUT DEVICE
.
.       SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC   CLEAR   X               CLEAR LOOP COUNTER
        LDT     LENGTH
WLOOP   TD      OUTPUT          TEST OUTPUT DEVICE
        JEQ     WLOOP           LOOP UNTIL READY
        LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
        WD      OUTPUT          WRITE CHARACTER
        TIXR    T               LOOP UNTIL ALL CHARACTERS
        JLT     WLOOP             HAVE BEEN WRITTEN
        RSUB                    RETURN TO CALLER
OUTPUT  BYTE    X'05'           CODE FOR OUTPUT DEVICE
        END     FIRST
```

- This SIX/XE program should look familiar, it is from you book!
- functions.txt (pg. 55)

Course Project: functions.txt

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	
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		

- functions.txt solution (pg. 58)
- Loc and Object Code columns now present!

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05
M^000014^05
M^000027^05
E^000000
```

- functions.txt object program solution (pg. 65)

Course Project: Implementation Details

- Cool! We can already see the answers ☺
- For many of the sample programs the solutions are given in the book
- As you implement assembler and add functionalities to successfully process the 6 sample programs, you can reference these solutions to see if you are on the right track

- This still doesn't tell us much about how to actually generate the displayed object code or location columns though ☹
- Let us try to figure out

Course Project: Implementation Details

- Our given functions.txt file has four tab delimited columns
  - Symbol Col, Operation Col, Operand Col and Comments Col
- Comments are indicated with a '.'
- In the solution a Loc Col and Object Code Col are added

Line	Source statement			
5	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
10	FIRST	STL	RETADR	SAVE RETURN ADDRESS
12		LDB	#LENGTH	ESTABLISH BASE REGISTER
13		BASE	LENGTH	
15	CLOOP	+JSUB	RDREC	READ INPUT RECORD
20		LDA	LENGTH	TEST FOR EOF (LENGTH = 0)
25		COMP	#0	
30		JEQ	ENDFIL	EXIT IF EOF FOUND
35		+JSUB	WRREC	WRITE OUTPUT RECORD
40		J	CLOOP	LOOP
45	ENDFIL	LDA	EOF	INSERT END OF FILE MARKER
50		STA	BUFFER	
55		LDA	#3	SET LENGTH = 3

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

Course Project: Implementation Details

- Pass 1 serve mainly to create the Loc Col and Symbol Table (SYMTAB)
- The Loc Col can be created by simply finding the program's starting address and tracking the amount of bytes that have been used by instructions
  - Length of instruction(e.g. 3 bytes)
  - Data area to be generated(e.g. buffer)
- The SYMTAB will keep track of the values in the Loc Column corresponding to each symbol in the Symbol Column

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

SYMTAB	
Name	LOCCTR
FIRST	1000
LOOP	1006
TABLE	1015
COUNT	1018
ZERO	101B
TOTAL	101E

Course Project: Implementation Details

- Pass 2 generates the Object Code column
- Take a second to re-read 2.2.1 Instruction Formats and Addressing Modes (pg. 57-61) to get an idea of what we will need to do

- First we should create an Operation Table (OPTAB)
  - The OPTAB will contain an entry for each instruction belonging to the SIC/XE instruction set
  - Each entry should contain the instruction's corresponding mnemonic, opcode, and format(s)
  - Re-read Appendix A

OPTAB

Mnemonic	Opcode
LDX	04
LDA	00
ADD	18
STA	0C
RSUB	4C
TIX	2C
JLT	38

Course Project: Implementation Details

- Once we have the OPTAB, we know an instruction's opcode and format
- Then (if necessary) we need to calculate the addressing mode information (n i x b p e bits)
- Finally (if necessary) we need to calculate the disp/TA information based on the addressing mode information

↑↑↑↑↑↑↑↑↑↑↑

- All this information will be used to calculate the instructions object code

Example 1

- Take a look in functions.txt at this line:

  FIRST   STL   RETADR   .SAVE RETURN ADDRESS

- First, from our OPTAB, STL : opcode 14 (hexadecimal)

  format   3/4 (3 or 4)

- This means the first byte of our obj code will contain the value 14 (0001 01~~00~~) plus the n and i bit values we find
- This also means we will either have a format 3 or 4 instruction

Calculate the n i x b p e bits

- n=1, i=1 indicates simple addressing(by default)
- n=1, i=0 indicates indirect addressing '@'
  - For example, in functions.txt find the J @RETADR line
- n=0, i=1 indicates immediate addressing '#'
  - For example, in functions.txt find the LDB #LENGTH line

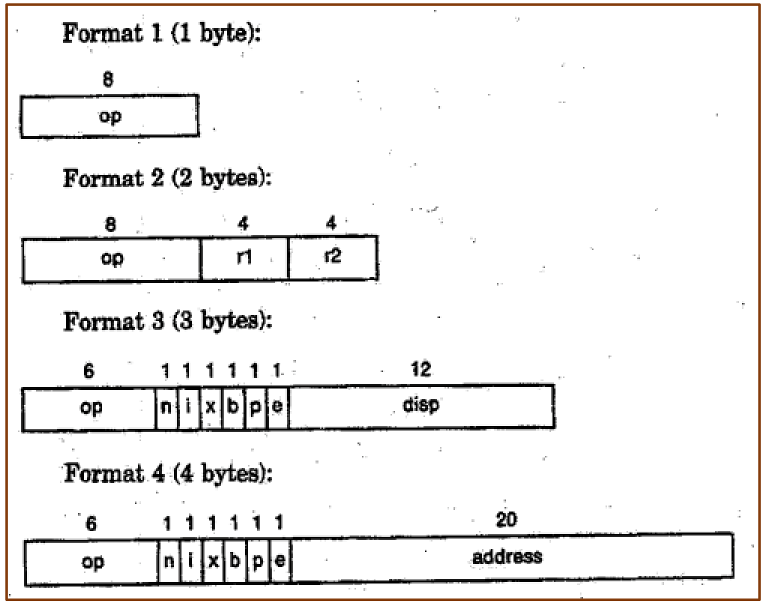
- x=1 indicates indexed addressing ', X'
  - For example, in functions.txt find the STCH BUFFER, X line

- b=1, p=0 indicates for base relative addressing
- b=0, p=1 indicates for program-counter relative addressing

- e=1 indicates a format 4 instruction '+'   (e=0 indicates format 3 instruction)

Bits: e

- Remember our SIC/XE addressing modes? (pg. 8 and 9)



- To indicate an instruction is format 4, the instruction will be preceded by a ‘+’
  - E.g, in functions.txt take a look at the 15TH line

```
15        CLOOP     +JSUB     RDREC          READ INPUT RECORD
```

- Since STL has no preceding ‘+’ it must be format 3

Bits: b p

- The base relative vs. program-counter relative distinction is a bit harder to discern
- In the reading we learned program-counter relative is used by default, but, if the displacement (disp) we calculate is out of range, we resort to base relative

Mode	Indication	Target address calculation	
Base relative	b = 1, p = 0	TA = (B) + disp	($0 \leq$ disp ≤ 4095)
Program-counter relative	b = 0, p = 1	TA = (PC) + disp	($-2048 \leq$ disp ≤ 2047)

Bits: b p

- To get value of b & p, we need to calculate the displacement (disp)
- Remember we are looking at the line:

  FIRST STL RETADR .SAVE RETURN ADDRESS
- From out 1st pass, this line's corresponding Loc Col value is 0000
- From the reading, the PC register will contain the next Loc Col value (0003)
- We also know from our first pass-generated SYMTAB that the value corresponding to RETADR is 0030 (TA)

Loc	Source statement		
0000	COPY	START	0
0000	FIRST	STL	RETADR
0003		LDB	#LENGTH
		BASE	LENGTH
0006	CLOOP	+JSUB	RDREC
000A		LDA	LENGTH
000D		COMP	#0
0010		JEQ	ENDFIL
0013		+JSUB	WRREC
0017		J	CLOOP
001A	ENDFIL	LDA	EOF
001D		STA	BUFFER
0020		LDA	#3
0023		STA	LENGTH
0026		+JSUB	WRREC
002A		J	@RETADR
002D	EOF	BYTE	C'EOF'
0030	RETADR	RESW	1
0033	LENGTH	RESW	1
0036	BUFFER	RESB	4096

Bits: b p

- From the book we see, for pc-relative addressing:

  TA=(PC)+disp -> disp=TA-(PC)
- This means our disp=0030-0003, disp=002D (hexadecimal values)
- This is within our program-counter relative disp bounds

  (-2048 <= disp <= 2047) (decimal values)

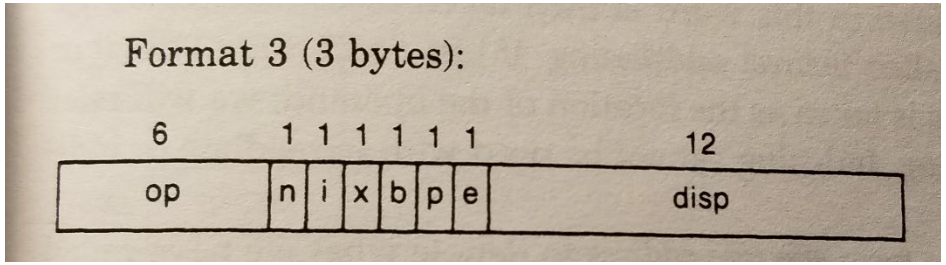
Mode	Indication	Target address calculation	
Base relative	$b = 1, p = 0$	$TA = (B) + disp$	$(0 \leq disp \leq 4095)$
Program-counter relative	$b = 0, p = 1$	$TA = (PC) + disp$	$(-2048 \leq disp \leq 2047)$

Generate object code

- We finally have all the information we need to form the object code for line:

  FIRST   STL   RETADR   .SAVE RETURN ADDRESS

- op (opcode)=14 or 0001 01~~00~~
- n=1,i=1,x=0,b=0,p=1,e=0 or 11 0010
  - We have simple addressing because we didn't have immediate or indirect addressing!
- disp=02D or 0000 0011 1101



Generate object code

FIRST STL RETADR .SAVE RETURN ADDRESS

- Object Code Generation:
  - First byte = opcode + ni -> 0001 0100 + 11 (or 14 + 3) -> 0001 0111 (or 17)

    So, our first byte = 0001 0111 (or 17)
  - Second byte first half = xbpe -> 0010 (or 2)

    So, our second byte first half = 0010 (or 2)
  - Second byte second half and third byte = disp -> 0000 0011 1101 (or 02D)

    So, our second byte second half and third byte = 0000 0011 1101 (or 02D)
  - Altogether, our 3-byte format 3 instruction is:

    0001 0111 0010 0000 0011 1101 (or 17202D)

Course Project: Implementation Details

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	
110		.			
115		.	SUBROUTINE TO 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		.	SUBROUTINE TO 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
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Figure 2.6 Program from Fig. 2.5 with object code.

Course Project: Implementation Details

- The overall process will be quite similar for format 4 instructions
- This process will differ slightly with the different addressing modes
- Base relative addressing will require you to keep track of what is loaded into the B register for use in your disp calculation
- Immediate and indirect addressing effects also need to be considered!

Example 2

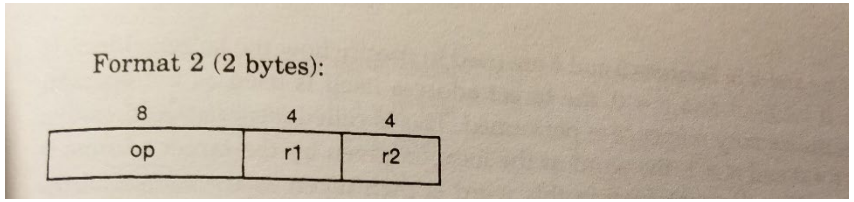
- Let us take a look at a format 2 instruction
- Look at functions.txt and find the line:

COMPR A,S .TEST for End Of Record (X'00')

- From our OPTAB we see COMPR has opcode of A0 (hexadecimal value) and that it is a format 2 instruction

Course Project: Implementation Details

- So, we already have the op section ready (A0 or 1010 0000) for the first byte
- We need to get the r1 and r2 values for the second byte



- The book lists all the register values for us in the first chapter

Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
PC	8	Program counter; contains the address of the next instruction to be fetched for execution
SW	9	Status word; contains a variety of information, including a Condition Code (CC)

Mnemonic	Number	Special use
B	3	Base register; used for addressing
S	4	General working register—no special use
T	5	General working register—no special use
F	6	Floating-point accumulator (48 bits)

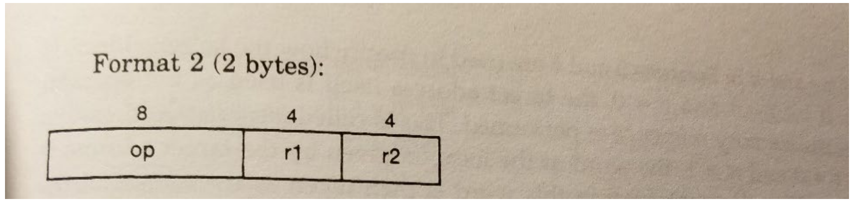
Course Project: Implementation Details

COMPR A,S .TEST for End Of Record (X'00')

Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
PC	8	Program counter; contains the address of the next instruction to be fetched for execution
SW	9	Status word; contains a variety of information, including a Condition Code (CC)

Mnemonic	Number	Special use
B	3	Base register; used for addressing
S	4	General working register—no special use
T	5	General working register—no special use
F	6	Floating-point accumulator (48 bits)

- We need the A (0 hexadecimal) and S (4 hexadecimal) register values for r1 and r2 respectively



Generate object code

COMPR A,S .TEST for End Of Record (X'00')

- Object Code Generation:
  - First byte = opcode -> 1010 0000 (or A0)

    1st byte = 1010 0000 (or A0)
  - Second byte first half = r1 -> A -> 0000 (or 0)

    2nd byte first half = 0000 (or 0)
  - Second byte second half = r2 -> S -> 0100 (or 4)

    2nd byte second half = 0100 (or 4)
  - Altogether, our 2-byte format 2 instruction is:

    1010 0000 0000 0100 (or A004)

Course Project: Implementation Details

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	
110		.			
115		.	SUBROUTINE TO 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		.	SUBROUTINE TO 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
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Figure 2.6 Program from Fig. 2.5 with object code.

What else?

- With each of the sample programs there will need to be added functionality your assembler will need to support
  - literals.txt adds literals
  - program-blocks.txt adds program blocks
  - etc.
- You also should create object programs using the object code you generate
  - I would suggest leaving this part until you can support all six test files
- Your assembler should support the six test programs
- To test for robustness, your assembler will be tested using files you do not have access to