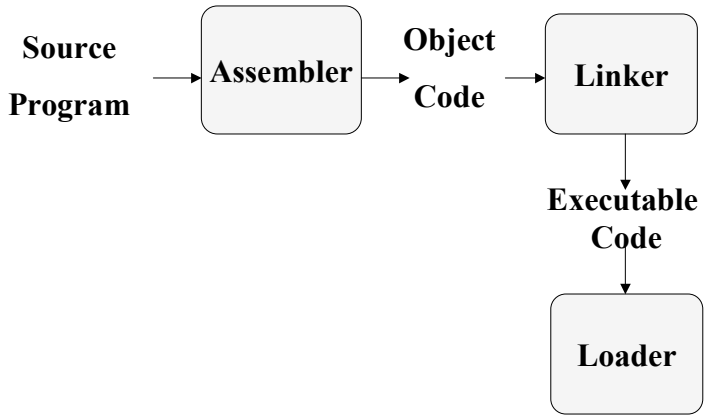
CS2422 Assembly Language & System Programming

November 30, 2006

Today's Topic

- Assembler: Basic Functions
  - Section 2.1 of Beck's "*System Software*" book.
- Reading Assignment: pages 43-52.

Role of Assembler



Example Program Fig. 2.1 (1/3)

Line		Source statement		
5	COPY	START	1000	
10	FIRST	STL	RETADR	SAVE RETURN ADDRESS
15	CLOOP	JSUB	RDREC	READ INPUT RECORD
20		LDA	LENGTH	TEST FOR EOF (LENGTH = 0)
25		COMP	ZERO	
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	THREE	SET LENGTH = 3
60		STA	LENGTH	
65		JSUB	WRREC	WRITE EOF
70		LDL	RETADR	GET RETURN ADDRESS
75		RSUB		RETURN TO CALLER
80	EOF	BYTE	C'EOF'	
85	THREE	WORD	3	
90	ZERO	WORD	0	
95	RETADR	RESW	1	
100	LENGTH	RESW	1	
105	BUFFER	RESB	4096	4096-BYTE BUFFER AREA

```
110   .
115   .         SUBROUTINE TO READ RECORD INTO BUFFER
120   .
125   RDREC     LDX     ZERO          CLEAR LOOP COUNTER
130             LDA     ZERO          CLEAR A TO ZERO
135   RLOOP     TD      INPUT         TEST INPUT DEVICE
140             JEQ     RLOOP         LOOP UNTIL READY
145             RD      INPUT         READ CHARACTER INTO REGISTER A
150             COMP    ZERO          TEST FOR END OF RECORD (X'00')
155             JEQ     EXIT          EXIT LOOP IF EOR
160             STCH    BUFFER,X      STORE CHARACTER IN BUFFER
165             TIX     MAXLEN        LOOP UNLESS MAX LENGTH
170             JLT     RLOOP           HAS BEEN REACHED
175   EXIT      STX     LENGTH        SAVE RECORD LENGTH
180             RSUB                  RETURN TO CALLER
185   INPUT     BYTE    X'F1'         CODE FOR INPUT DEVICE
190   MAXLEN    WORD    4096
195   .
200   .         SUBROUTINE TO WRITE RECORD FROM BUFFER
205   .
210   WRREC     LDX     ZERO          CLEAR LOOP COUNTER
215   WLOOP     TD      OUTPUT        TEST OUTPUT DEVICE
220             JEQ     WLOOP         LOOP UNTIL READY
225             LDCH    BUFFER,X      GET CHARACTER FROM BUFFER
230             WD      OUTPUT        WRITE CHARACTER
235             TIX     LENGTH        LOOP UNTIL ALL CHARACTERS
240             JLT     WLOOP           HAVE BEEN WRITTEN
245             RSUB                  RETURN TO CALLER
250   OUTPUT    BYTE    X'05'         CODE FOR OUTPUT DEVICE
255             END     FIRST
```

Example Program Fig. 2.1 (2/3)

- Purpose
  - Reads records from input device (code F1)
  - Copies them to output device (code 05)
  - At the end of the file, writes EOF on the output device, then RSUB to the operating system

Example Program Fig. 2.1 (3/3)

- Data transfer (RD, WD)
  - A buffer is used to store record
  - Buffering is necessary for different I/O rates
  - The end of each record is marked with a null character (00_{16})
  - The end of the file is indicated by a zero-length record
- Subroutines (JSUB, RSUB)
  - RDREC, WRREC
  - Save link register first before nested jump

Assembler Directives

- Pseudo-Instructions
  - Not translated into machine instructions
  - Providing information to the assembler
- Basic assembler directives
  - START
  - END
  - BYTE
  - WORD
  - RESB
  - RESW

Functions of a Basic Assembler

- Mnemonic code (or instruction name) → opcode.
- Symbolic operands (e.g., variable names) → addresses.
- Choose the proper instruction format and addressing mode.
- Constants → Numbers.
- Output to object files and listing files.

Example Program with Object Code

Line	Loc	Source statement			Object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	

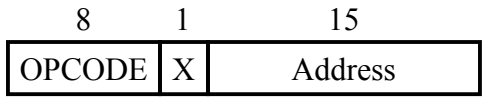
```
110           .
115           .        SUBROUTINE TO READ RECORD INTO BUFFER
120           .
125   2039    RDREC    LDX     ZERO          041030
130   203C             LDA     ZERO          001030
135   203F    RLOOP    TD      INPUT         E0205D
140   2042             JEQ     RLOOP         30203D
145   2045             RD      INPUT         D8205D
150   2048             COMP    ZERO          281030
155   204B             JEQ     EXIT          302057
160   204E             STCH    BUFFER,X      549039
165   2051             TIX     MAXLEN        2C205E
170   2054             JLT     RLOOP         38203F
175   2057    EXIT     STX     LENGTH        101036
180   205A             RSUB                  4C0000
185   205D    INPUT    BYTE    X'F1'         F1
190   205E    MAXLEN   WORD    4096          001000
195           .
200           .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205           .
210   2061    WRREC    LDX     ZERO          041030
215   2064    WLOOP    TD      OUTPUT        E02079
220   2067             JEQ     WLOOP         302064
225   206A             LDCH    BUFFER,X      509039
230   206D             WD      OUTPUT        DC2079
235   2070             TIX     LENGTH        2C1036
240   2073             JLT     WLOOP         382064
245   2076             RSUB                  4C0000
250   2079    OUTPUT   BYTE    X'05'         05
255                    END     FIRST
```

SIC Instruction Set (Review)

- Load/Store: LDA/STA, LDX/STX…etc.
- Arithmetic: ADD, SUB, MUL, DIV
- Compare: COMP
- Jump: J
- Conditional Jump: JLT, JEQ, JGT
- See Appendix A for the complete list.

SIC Instruction Format (again)

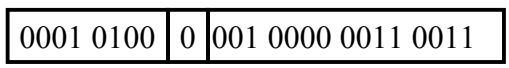
- Opcode: 8 bits
- Address: one bit flag (x) and 15 bits of address



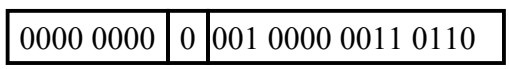
Examples

- Mnemonic code (or instruction name) → opcode.
- Examples:

  STL 1033 → opcode 14 10 33



  LDA 1036 → opcode 00 10 36

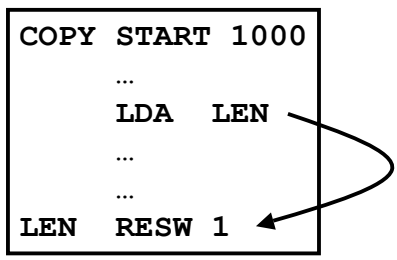


Symbolic Operands

- We're not likely to write memory addresses directly in our code.
  - Instead, we will define variable names.
- Other examples of symbolic operands:
  - Labels (for jump instructions)
  - Subroutines
  - Constants

Converting Symbols to Numbers

- Isn't it straightforward?
  - Isn't it simply the sequential processing of the source program, one line at a time?
  - Not so, if we have *forward references.*

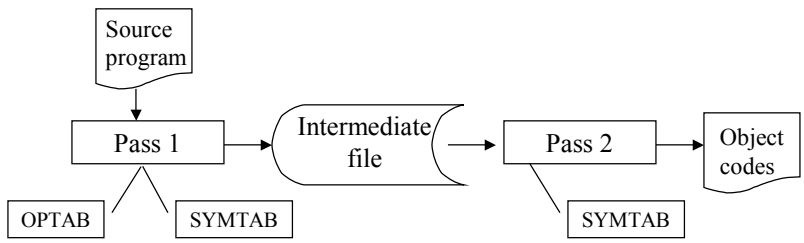


Two Pass Assembler

- Pass 1
  - Assign addresses to all statements in the program
  - Save the values assigned to all labels for use in Pass 2
  - Perform some processing of assembler directives
- Pass 2
  - Assemble instructions
  - Generate data values defined by BYTE, WORD
  - Perform processing of assembler directives not done in Pass 1
  - Write the object program and the assembly listing

Two Pass Assembler

- Read from input line
  - LABEL, OPCODE, OPERAND



Two Pass Assembler – Pass 1

Pass 1:

```
begin
   read first input line
   if OPCODE = 'START' then
       begin
           save #[OPERAND] as starting address
           initialize LOCCTR to starting address
           write line to intermediate file
           read next input line
       end {if START}
   else
       initialize LOCCTR to 0
   while OPCODE ≠ 'END' do
       begin
           if this is not a comment line then
               begin
                   if there is a symbol in the LABEL field then
                       begin
                           search SYMTAB for LABEL
                           if found then
                               set error flag (duplicate symbol)
                           else
                               insert (LABEL,LOCCTR) into SYMTAB
                       end {if symbol}
                   search OPTAB for OPCODE
                   if found then
                       add 3 {instruction length} to LOCCTR
                   else if OPCODE = 'WORD' then
                       add 3 to LOCCTR
                   else if OPCODE = 'RESW' then
                       add 3 * #[OPERAND] to LOCCTR
                   else if OPCODE = 'RESB' then
                       add #[OPERAND] to LOCCTR
                   else if OPCODE  = 'BYTE' then
                       begin
                           find length of constant in bytes
                           add length to LOCCTR
                       end {if BYTE}
                   else
                       set error flag (invalid operation code)
               end {if not a comment}
           write line to intermediate file
           read next input line
       end {while not END}
   write last line to intermediate file
   save (LOCCTR - starting address) as program length
end {Pass 1}
```

Figure 2.4(a) Algorithm for Pass 1 of assembler.

Two Pass Assembler – Pass 2

Pass 2:

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
       begin
           write listing line
           read next input line
       end {if START}
   write Header record to object program
   initialize first Text record
   while OPCODE ≠ 'END' do
       begin
           if this is not a comment line then
               begin
                   search OPTAB for OPCODE
                   if found then
                       begin
                           if there is a symbol in OPERAND field then
                               begin
                                   search SYMTAB for OPERAND
                                   if found then
                                       store symbol value as operand address
                                   else
                                       begin
                                           store 0 as operand address
                                           set error flag (undefined symbol)
                                       end
                               end {if symbol}
                           else
                               store 0 as operand address
                           assemble the object code instruction
                       end {if opcode found}
                   else if OPCODE = 'BYTE' or 'WORD' then
                       convert constant to object code
                   if object code will not fit into the current Text record then
                       begin
                           write Text record to object program
                           initialize new Text record
                       end
                   add object code to Text record
               end {if not comment}
           write listing line
           read next input line
       end {while not END}
   write last Text record to object program
   write End record to object program
   write last listing line
end {Pass 2}
```

Figure 2.4(b) Algorithm for Pass 2 of assembler.

Data Structures

- Operation Code Table (OPTAB)
- Symbol Table (SYMTAB)
- Location Counter(LOCCTR)

OPTAB (operation code table)

- Content
  - Mnemonic, machine code (instruction format, length) etc.
- Characteristic
  - Static table
- Implementation
  - Array or hash table, easy for search

SYMTAB (symbol table)

- Content
  - Label name, value, flag, (type, length) etc.
- Characteristic
  - Dynamic table (insert, delete, search)
- Implementation
  - Hash table, non-random keys, hashing function

COPY	**1000**
FIRST	**1000**
CLOOP	**1003**
ENDFIL	**1015**
EOF	**1024**
THREE	**102D**
ZERO	**1030**
RETADR	**1033**
LENGTH	**1036**
BUFFER	**1039**
RDREC	**2039**

One-Pass Assemblers

- Forward references can be resolved in One-Pass Assemblers too!
- Add a linked list to the Symbol Table to keep track of unresolved references. (See p.95)
- We will discuss 1-pass assembler again in the future (Section 2.4.1)

Object Program

- Header
  - Col. 1 H
  - Col. 2~7 Program name
  - Col. 8~13 Starting address (hex)
  - Col. 14-19 Length of object program in bytes (hex)
- Text
  - Col.1 T
  - Col.2~7 Starting address in this record (hex)
  - Col. 8~9 Length of object code in this record in bytes (hex)
  - Col. 10~69 Object code (69-10+1)/6=10 instructions
- End
  - Col.1 E
  - Col.2~7 Address of first executable instruction (hex) (END program_name)

Fig. 2.3

H COPY 001000 00107A
T 001000 1E 141033 482039 001036 281030 301015 482061 ...
T 00101E 15 0C1036 482061 081044 4C0000 454F46 000003 000000
T 002039 1E 041030 001030 E0205D 30203F D8205D 281030 …
T 002057 1C 101036 4C0000 F1 001000 041030 E02079 302064 …
T 002073 07 382064 4C0000 05
E 001000 ←starting address