A Simple Two-Pass Assembler
Main Functions

- Translate mnemonic operation codes to their machine language equivalents
- Assign machine addresses to symbolic labels used by the programmers

Depend heavily on the source language it translates and the machine language it produces.

E.g., the instruction format and addressing modes
Basic Functions
Example 2.1

Line numbers are not part of the program. They are for reference only.

Forward reference

Call subroutine

code
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>.</td>
<td>SUBROUTINE TO READ RECORD INTO BUFFER</td>
</tr>
<tr>
<td>120</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>RDREC</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>130</td>
<td>LDX</td>
<td>CLEAR A TO ZERO</td>
</tr>
<tr>
<td>135</td>
<td>LDA</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>140</td>
<td>TD</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>145</td>
<td>RLOOP</td>
<td>READ CHARACTER INTO REGISTER A</td>
</tr>
<tr>
<td>150</td>
<td>TD</td>
<td>TEST FOR END OF RECORD (X'00')</td>
</tr>
<tr>
<td>155</td>
<td>COMP</td>
<td>EXIT LOOP IF EOR</td>
</tr>
<tr>
<td>160</td>
<td>RD</td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>165</td>
<td>RD</td>
<td>LOOP UNLESS MAX LENGTH</td>
</tr>
<tr>
<td>170</td>
<td>STX</td>
<td>HAS BEEN REACHED</td>
</tr>
<tr>
<td>175</td>
<td>STX</td>
<td>SAVE RECORD LENGTH</td>
</tr>
<tr>
<td>180</td>
<td>EXIT</td>
<td>RETURN TO CALLER</td>
</tr>
<tr>
<td>185</td>
<td>INPUT</td>
<td>CODE FOR INPUT DEVICE</td>
</tr>
<tr>
<td>190</td>
<td>MAXLEN</td>
<td></td>
</tr>
<tr>
<td>195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE TO WRITE RECORD FROM BUFFER

LDX ZERO
TDX OUTPUT
JEQ WLOOP
LDCH BUFFER,X
WD OUTPUT
TIX LENGTH
ULT WLOOP
OUTPUT BYTE X’05’,
END FIRST
Purpose of Example 2.1 (COPY)

- It is a copy function that reads some records from a specified input device and then copies them to a specified output device
  - Reads a record from the input device (code F1)
  - Copies the record to the output device (code 05)
  - Repeats the above steps until encountering EOF.
  - Then writes EOF to the output device
  - Then call RSUB to return to the caller
RDREC and WRREC

• Data transfer
  – A record is a stream of bytes with a null character \((00_{16})\) at the end.
  – If a record is longer than 4096 bytes, only the first 4096 bytes are copied.
  – EOF is indicated by a zero-length record. (I.e., a byte stream with only a null character.
  – Because the speed of the input and output devices may be different, a buffer is used to temporarily store the record

• Subroutine call and return
  – On line 10, “STL RETADDR” is called to save the return address that is already stored in register L.
  – Otherwise, after calling RD or WR, this COPY cannot return back to its caller.
Assembler Directives

- Assembler directives are pseudo instructions
  - They will not be translated into machine instructions.
  - They only provide instruction/direction/information to the assembler.

- Basic assembler directives:
  - START:
    - Specify name and starting address for the program
  - END:
    - Indicate the end of the source program, and (optionally) the first executable instruction in the program.
Assembler Directives (cont’d)

– BYTE:
  • Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.

– WORD:
  • Generate one-word integer constant

– RESB:
  • Reserve the indicated number of bytes for a data area

– RESW:
  • Reserve the indicated number of words for a data area
An Assembler’s Job

• Convert mnemonic operation codes to their machine language codes
• Convert symbolic (e.g., jump labels, variable names) operands to their machine addresses
• Use proper addressing modes and formats to build efficient machine instructions
• Translate data constants into internal machine representations
• Output the object program and provide other information (e.g., for linker and loader)
Object Program Format

- Header
  - Col. 1: H
  - Col. 2~7: Program name
  - Col. 8~13: Starting address of object program (hex)
  - Col. 14-19: Length of object program in bytes (hex)

- Text
  - Col. 1: T
  - Col. 2~7: Starting address for object code in this record (hex)
  - Col. 8~9: Length of object code in this record in bytes (hex)
  - Col. 10~69: Object code, represented in hexa (2 col. per byte)

- End
  - Col. 1: E
  - Col. 2~7: Address of first executable instruction in object program (hex)
<table>
<thead>
<tr>
<th>Line</th>
<th>Loc</th>
<th>Source statement</th>
<th>Object code</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1000</td>
<td>START</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>FIRST</td>
<td>RETADR</td>
</tr>
<tr>
<td>15</td>
<td>1003</td>
<td>CLOOP</td>
<td>RDREC</td>
</tr>
<tr>
<td>20</td>
<td>1006</td>
<td>LDA</td>
<td>LENGTH</td>
</tr>
<tr>
<td>25</td>
<td>1009</td>
<td>COMP</td>
<td>ZERO</td>
</tr>
<tr>
<td>30</td>
<td>100C</td>
<td>JEQ</td>
<td>ENDFIL</td>
</tr>
<tr>
<td>35</td>
<td>100F</td>
<td>JSUB</td>
<td>WRREC</td>
</tr>
<tr>
<td>40</td>
<td>1012</td>
<td>J</td>
<td>CLOOP</td>
</tr>
<tr>
<td>45</td>
<td>1015</td>
<td>ENDFIL</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1018</td>
<td>STA</td>
<td>EOF</td>
</tr>
<tr>
<td>55</td>
<td>101B</td>
<td>LDA</td>
<td>BUFFER</td>
</tr>
<tr>
<td>60</td>
<td>101E</td>
<td>STA</td>
<td>THREE</td>
</tr>
<tr>
<td>65</td>
<td>1021</td>
<td>JSUB</td>
<td>LENGTH</td>
</tr>
<tr>
<td>70</td>
<td>1024</td>
<td>LDL</td>
<td>RETADR</td>
</tr>
<tr>
<td>75</td>
<td>1027</td>
<td>RSUB</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>102A</td>
<td>EOF</td>
<td>BYTE</td>
</tr>
<tr>
<td>85</td>
<td>102D</td>
<td>THREE</td>
<td>C’EOF’</td>
</tr>
<tr>
<td>90</td>
<td>1030</td>
<td>ZERO</td>
<td>3</td>
</tr>
<tr>
<td>95</td>
<td>1033</td>
<td>RETADR</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>1036</td>
<td>LENGTH</td>
<td>1</td>
</tr>
<tr>
<td>105</td>
<td>1039</td>
<td>BUFFER</td>
<td>RESW</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td>RESW</td>
</tr>
<tr>
<td>Line</td>
<td>Assembly Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>------------------------------</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>SUBROUTINE TO READ RECORD INTO BUFFER</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>2039</td>
<td>RDREC</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td></td>
<td>LDX ZERO</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>203C</td>
<td>LDA ZERO</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>203F</td>
<td>RLOOP</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>2042</td>
<td>TD INPUT</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>2045</td>
<td>RD INPUT</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>2048</td>
<td>COMP ZERO</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>204B</td>
<td>JEQ EXIT</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>204E</td>
<td>STCH BUFFER,X</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>2051</td>
<td>TIX MAXLEN</td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>2054</td>
<td>JLT RLOOP</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>2057</td>
<td>EXIT</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>205A</td>
<td>RSUB</td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>205D</td>
<td>INPUT BYTE 'F1'</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>205E</td>
<td>MAXLEN WORD 4096</td>
<td></td>
</tr>
<tr>
<td>195</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Binary Values:
- 041030
- 001030
- E0205D
- 30203F
- D8205D
- 281030
- 302057
- 549039
- 2C205E
- 38203F
- 101036
- 4C0000
- F1
- 001000
SUBROUTINE TO WRITE RECORD FROM BUFFER

LDX ZERO
TD OUTPUT
JEQ WLOOP
LDCH BUFFER, X
WD OUTPUT
TIX LENGTH
JLT WLOOP
RSUB WLOOP
OUTPUT BYTE X'05'
END FIRST
The Object Code for COPY

H COPY  001000 00107A
T 001000 1E 141033 482039 001036 281030 301015 482061 3C1003
     00102A 0C1039 00102D
T 00101E 15 0C1036 482061 081044 4C0000 454F46 000003 000000
T 002039 1E 041030 001030 E02079 302064 509039 DC2079 2C1036
T 002073 07 382064 4C0000 05
E 001000

There is no object code corresponding to addresses 1033-2038. This storage is simply reserved by the loader for use by the program during execution.
Two Pass Assembler

• Pass 1
  – Assign addresses to all statements in the program
  – Save the values (addresses) assigned to all labels (including label and variable names) for use in Pass 2 (deal with forward references)
  – Perform some processing of assembler directives (e.g., BYTE, RESW, these can affect address assignment)

• Pass 2
  – Assemble instructions (generate opcode and look up addresses)
  – 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
A Simple Two Pass Assembler Implementation

Source program

Pass 1

Mnemonic and opcode mappings are referenced from here

OPTAB

SYMTAB

Pass 2

Label and address mappings enter here

SYMTAB

Object codes

Label and address mappings are referenced from here

READ (Label, opcode, operand)
Three Main Data Structures

• Operation Code Table (OPTAB)
• Location Counter (LOCCTR)
• Symbol Table (SYMTAB)
OPTAB (operation code table)

• Content
  – The mapping between mnemonic and machine code. Also include the instruction format, available addressing modes, and length information.

• Characteristic
  – Static table. The content will never change.

• Implementation
  – Array or hash table. Because the content will never change, we can optimize its search speed.

• In pass 1, OPTAB is used to look up and validate mnemonics in the source program.
• In pass 2, OPTAB is used to translate mnemonics to machine instructions.
Location Counter (LOCCTR)

- This variable can help in the assignment of addresses.
- It is initialized to the beginning address specified in the START statement.
- After each source statement is processed, the length of the assembled instruction and data area to be generated is added to LOCCTR.
- Thus, when we reach a label in the source program, the current value of LOCCTR gives the address to be associated with that label.
Symbol Table (SYMTAB)

• Content
  – Include the label name and value (address) for each label in the source program.
  – Include type and length information (e.g., int64)
  – With flag to indicate errors (e.g., a symbol defined in two places)

• Characteristic
  – Dynamic table (i.e., symbols may be inserted, deleted, or searched in the table)

• Implementation
  – Hash table can be used to speed up search
  – Because variable names may be very similar (e.g., LOOP1, LOOP2), the selected hash function must perform well with such non-random keys.
The Pseudo Code for 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}
The Pseudo Code for 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
          end
        end
      end
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
  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}