SRES's Sanjivani College of Engineering, Kopargaon
(*An Autonomous Institute*)
Department of Computer Engineering

SPOS Lab Manual

Assignment No. 01

Title: Implementation of Pass 1 of Two Pass Assembler

Aim :
Design suitable data structures and implement pass-I of a two-pass assembler for pseudo-machine in Java using object oriented feature. Implementation should consist of a fewinstructions from each category and few assembler directives.

Inputs :
1. Assembly language program containing few instructions from each category
2. Opcode Table

Outputs :
1. Intermediate representation of assembly language program
2. Symbol table
3. Literal Table
4. Pool Table

Theory :
1. Processing of an assembly statement begins with the processing of its label field.
   If label of the statement is present then it is entered into the SYMTAB as a new entry. Also current LC value is entered as a address of the symbol
2. Class of mnemonic opcode (imperative, declaration or assembler directive) is determined by using class information from OPTAB
3. In case of an imperative statement, the length of the machine instruction is added to location counter. The length is also entered in the SYMTAB entry of the symbol (if any) defined in the statement
4. For a declaration or assembler directive statement, the routine mentioned in the mnemonic info field is called to perform appropriate processing of the statement
5. Assembler uses LITTAB and POOLTAB as follows: at any stage, the current literal pool is the last pool in LITTAB. On encountering an LTORG or END statement, literals in the current pool are allocated addresses starting with the current address in the location counter and address in the location counter is appropriately incremented.

Mnemonic operation codes for Imperative Statements (IS)

Instruction opcode	Assembly mnemonic	Remarks
00	STOP	Stop execution
01	ADD	Perform addition

02	SUB	Perform subtraction
03	MULT	Perform multiplication
04	MOVER	Move from memory to register
05	MOVEM	Move from register to memory
06	COMP	Compare and set condition code
07	BC	Branch on condition
08	DIV	Perform division
09	READ	Read into register
10	PRINT	Print contents of register

Codes for Assembler Directives (AD)

START	01
END	02
ORIGIN	03
EQU	04
LTORG	05

Codes for Declaration Statements (DL)

DC	01
DS	02

Pass I data structures

1. OPTAB A table of mnemonic opcodes and related information
2. SYMTAB Symbol table
3. LITTAB A table of literals used in the program
4. POOLTAB A table of information concerning literal pools

OPTAB

1. OPTAB contains the fields mnemonic opcode, class and mnemonic info.
2. The class field indicates whether the opcode corresponds to an imperative statements (IS), a declaration statement (DL), or an assembler directive (AD)
3. If an imperative , the mnemonic info field contains the pair (machine opcode, instruction length); otherwise it contains the id of a routine that handles the declaration or directive statement

Mnemonic opcode	Class	Machine Opcode	Length
STOP	IS	00	1
ADD	IS	01	1
SUB	IS	02	1
MULT	IS	03	1
MOVER	IS	04	1
MOVEM	IS	05	1
COMP	IS	06	1
BC	IS	07	1

DIV	IS	08	1
READ	IS	09	1
PRINT	IS	10	1
START	AD	01	--
END	AD	02	--
ORIGIN	AD	03	--
EQU	AD	04	--
LTORG	AD	05	--
DC	DL	01	--
DS	DL	02	--

Sample Example Showing Inputs and Outputs of Pass1 Assembler

Assembly Program			**Intermediate Program**	
	START	200		(AD, 01) (C, 200)
	MOVER	AREG, =’5	200	(IS, 04) (1) (L, 01)
	MOVEM	AREG , A	201	(IS, 05) (1) (S, 01)
LOOP	MOVER	AREG, A	202	(IS,04) (1) (S, 01)
	MOVER	CREG, B	203	(IS, 04) (3) (S,03)
	ADD	CREG, =’1’	204	(IS, 01) (3) (L, 02)
	...			
	BC	ANY, NEXT	210	(IS, 07) (6) (S, 04)
	LTORG		211	(AD, 05) (C, 5)
			212	(AD, 05) (C, 1)
	...			
NEXT	SUB	AREG, =’1	214	(IS, 02) (1) (L, 03)
	BC	LT, BACK	215	(IS, 07) (1) (S, 05)
LAST	STOP		216	(IS, 00)
	ORIGIN	LOOP+2		
	MULT	CREG, B	204	(IS, 03) (1) (S, 03)
	ORIGIN	LAST+1		
A	DS	1	217	(DL, 02) (C, 1)
BACK	EQU	LOOP		
B	DS	1	218	(DL, 02) (C, 1)
	END		219	(AD, 02) (C, 1)

SYMTAB

A SYMTAB entry contains the fields symbol entry number, symbol name, address and length

symbol entry number	Symbol Name	Symbol address	Symbol length
1	A	217	1
2	LOOP	202	1
3	B	218	1
4	NEXT	214	1
5	BACK	202	1
6	LAST	216	1

LITTAB

a LITTAB entry contains the fields literal entry nymber, literal value and address

Literal entry number	Literal value	address
1	=’5’	211
2	=’1’	212
3	=’1’	219

POOLTAB

a POOLTAB entry contains the fields pool number, literal number and number of literal in pool

Pool number	First literal entry number	Number of literals in pool
1	1	2
2	3	1
3	4	0

Intermediate Code

- The intermdiate code consists of a sequence of intermediate code units (IC units). Each IC unit consists of following three fields
  - Address
  - Representation of mnemonic opcode
  - Representation of operands

- The mnemonic opcode field contains a pair of the form
  (statement class, code)
  where class can be one of IS, AD and DL
  For IS class code is instruction opcode in machine language.
  For AD and DL class code is ordinal number within class

- **Intermediate code for IS class Statements**
- The first operand is represented by single digit number in the range 1...4 that represents a CPU register
  AREG 1

BREG	2
CREG	3
DREG	4

- OR it is a condition code in the range 1...6

LT	1
LE	2
EQ	3
GT	4
GE	5
ANY	6

- The second operand is a memory operand, is represented by a pair of the form
  (operand class, code)
  where operand class is one of C, S and L standing for constant, symbol and literal, respectively
- For a constant, the code field contains the constant value. For a symbol or literal the code field contains the entry number of the operand in SYMTAB or LITTAB

The assembler implements pass1 by using following algorithm. It uses following data structures OPTAB, SYMTAB, LITTAB and POOLTAB

LC : Location Counter
littab_ptr : Points to an entry in LITTAB
pooltab_ptr : Points to an entry in POOLTAB

Algorithm for Pass 1 of a two-pass assembler

1. LC = 0; (default value)
   littab_ptr = 1;
   pooltab_ptr = 1;
   POOLTAB[1].first = 1;
   POOLTAB[1].#literals = 0;

2. While the next statement is not an END statement

   a) If a symbol is present in the label field then
      this_label = symbol in the label field;
      Make an entry (this_label, <LC>, --) in SYMTAB

   b) If an LTORG statement then
      i) If POOLTAB[pooltab_ptr].#literals > 0 then
         Process the entries LITTAB [POOLTAB[pooltab_ptr].first]....LITTAB [littab_ptr-1] to allocate memory to the literal, put address of the allocated memory area in the address field of the LITTAB entry, and update the address contained in location counter accordingly.
      ii) pooltab_ptr = pooltab_ptr + 1;
      iii) POOLTAB [pooltab_ptr].first = littab_ptr;

POOLTAB [pooltab_ptr]. #literals = 0;

c) If a START and ORIGIN statement then
LC = value specified in operand field

d) If an EQU statement then
i) this_addr = value of <address specification>
ii) Correct the SYMTAB entry for this_label to (this_label, this_addr, 1)

e) If a declaration statement then
i) Invoke the routine whose id is mentioned in the mnemonic info field, This routine returns code and size.
ii) If a symbol is present in the label field, correct the symtab entry for this_label to (this_label, <LC>, size)
iii) LC = LC + size;
iv) Generate intermediate code for the declaration statement.

f) If an imperative statement then
i) code = machine opcode from the mnemonic info field of OPTAB
ii) LC = LC + instruction length from the mnemonic info field of OPTAB;
iii) If operand is a literal then
this_literal = literal in operand field;
if POOLTAB[pooltab_ptr]. #literals = 0 or this_literal does not match any literal in the range LITTAB [POOLTAB [pooltab_ptr]].first ... LITTAB[littab_ptr – 1] then
LITTAB[littab_ptr].value = this_literal;
POOLTAB[pooltab_ptr].#literals = POOLTAB[pooltab_ptr].#literals + 1
littab_ptr = littab_ptr + 1;
else (i.e. operand is a symbol)
this_entry = SYMTAB entry number of operand;
Generate intermediate code for the imperative statement

3. (Processing of END statement)
a) Perform actions i) -iii) of Step 2(b)
b) Generate intermediate code for the END statement

Conclusion: In this assignment we have implemented pass I of assembler. Information from opcode table is used to process input assembly file and to generate Intermediate code, symbol table, literal table and pool table.

References: Systems Programming & Operating Systems by D M Dhamdhere

Prepared by
Prof.N.G.Pardeshi
Subject Teacher

Approved by
Dr. D.B.Kshirsagar
HOD