Pass-1 Assembler.

Page No.
Date

Aim - To Implement Pass-1 Assembler.

Problem statement - Design Suitable data structure and implement Pass-I of a two-pass Assembler for pseudo-machine in Java using object oriented features. Implementation should consist of a few instructions from each category and few assembler directives.

Theory :

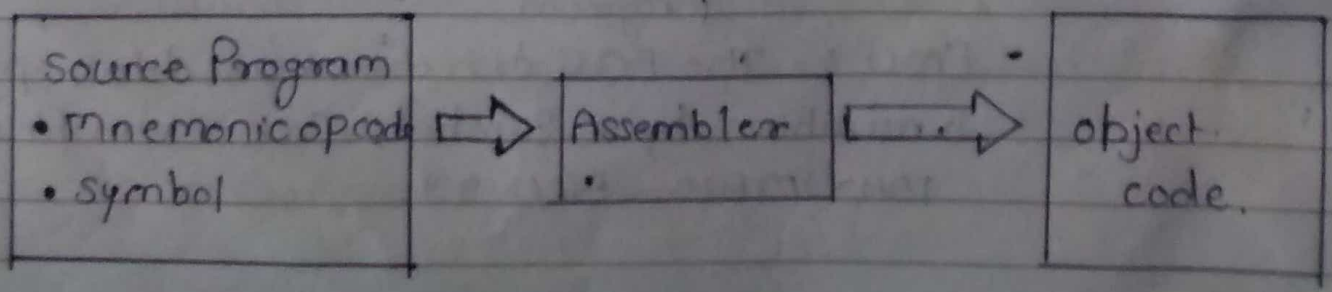
Assembler Languages

An assembly language is low-level programming languages for a computer, or other programmable device, in which there is a very strong correspondence between the language and the architectures machine code instructions.

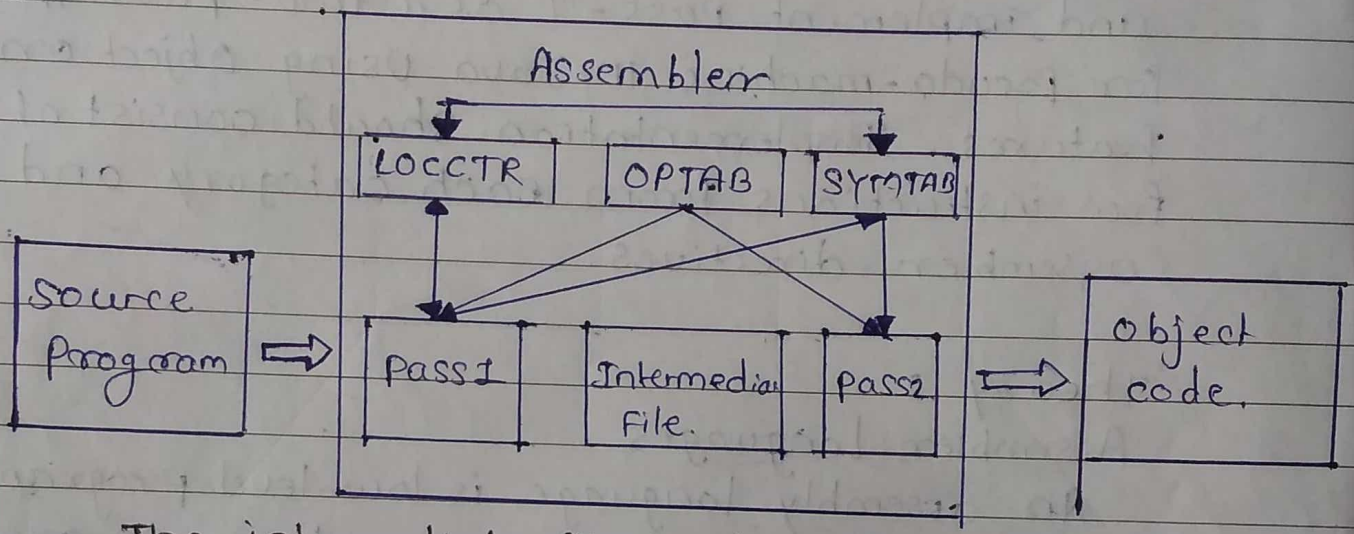
Assembly language uses a mnemonics to represent each low-level machine operations or opcode.

Assembler

Assembler language is converted into executable machine code by a utility program refferedas to as an assembler; the conversion process is refferred to as assembly; or assembling the code.



An Assembler is a translator that translate. an assembler program into a conventional m/c languages program.



The intermediate file include Each source Statement, assigned address and error indicators.

Assembler Directives -

- Assembler Directives are peseudo instruction
  - They will not be translated into machine instructions.
  - They only provides instruction /direction /information to the assembler.
- Basic assembler directives:
  - START : Specify name and starting address for the program.
  - END : indicates the end of the source program.
  - EQU : The EQU directives used to replace a number by a symbol. For examples : MAXIMUM EQU 99.

Page No.
Date

Three Main Data Structure.
- operation Code Table (OPTAB)
- Location Counter (LOCCTR)
- Symbol Table (SYMTAB).

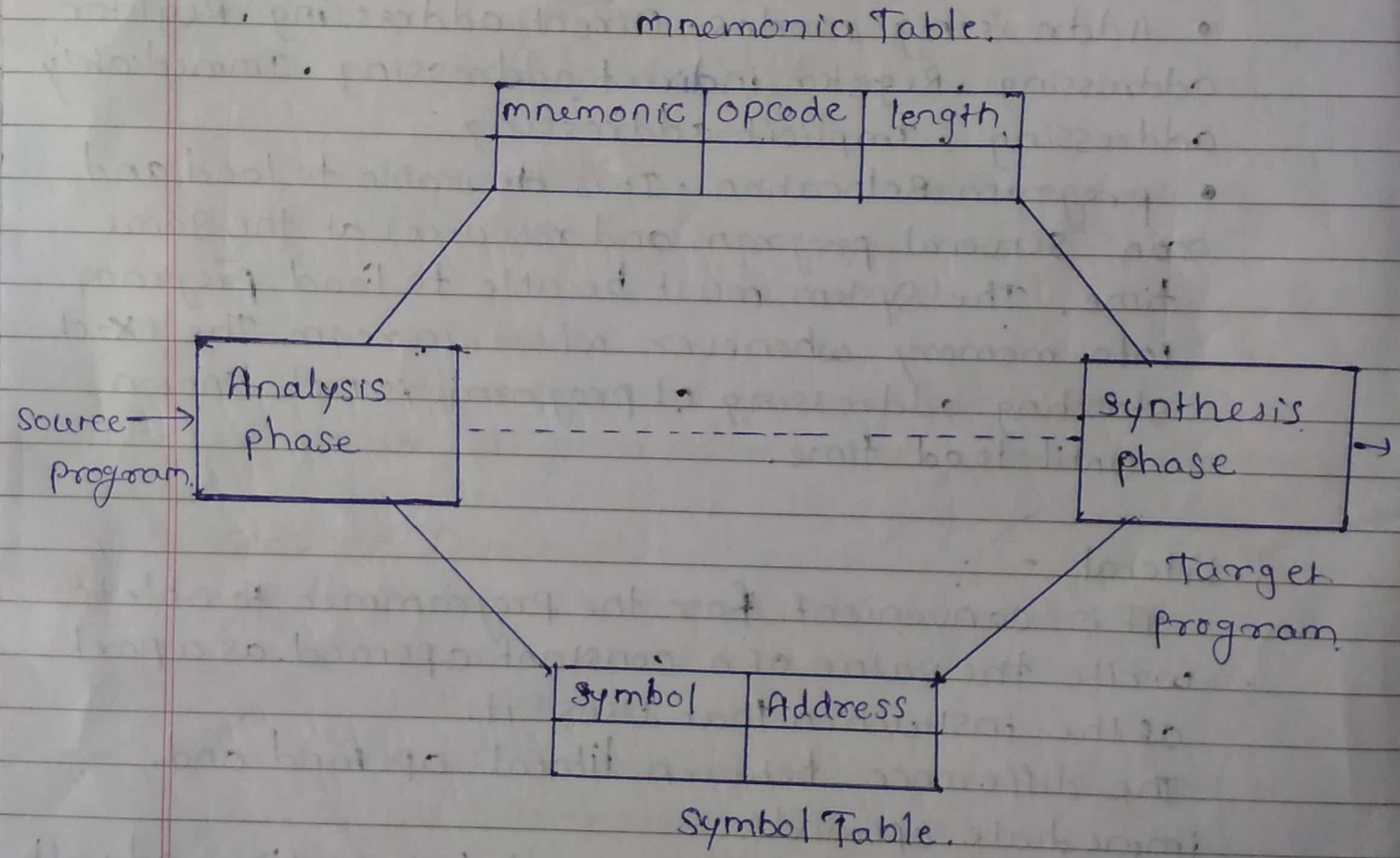
Instruction format -
- Addressing modes : Direct addressing . Register addressing . Register indirect addressing . Immediately addressing . Implicit addressing
- program Relocation . It is desirable to load and run several program and resources at the same time. The system must be able to load program into memory whenever where in room. The exact starting addressing of program is not known until load time.

Literal -
It is convenient for the programmer to able to write the value of a constant operand as a part of the instruction that uses it.
The difference between literal operand and immediate operands.
- For literal operand we use '=' as prefix, and with immediately operand we use '#' as prefix.
- During immediately addressing, the operand value is assembled as part of the machine instruction, and there is no memory reference.
- with a literal, the assembler generates the specified value as a constant at some other memory locations.

Page No.
Date

One-pass assembler

- A onepass assembler passes over source files exactly onece, inthe same pass collecting the label, resolving furture reference and doing the actual assembly.



Forward references in one pass transfer Assembler.

- omits the operand address if the symbol has not yet been defined.
- Enters this undefined symbol into SYMTAB and indicates that it is undefined.
- Add the address of the operand address to a list of forward references associated with the SYMTAB entery.

Page No.
Date

Data structures for Assembler:
opcode table
Looked up for the translations mnemonic code

key : mnemonic code.
Hashing is usually used onces prepared, the table is not changed efficient lookup is desired sices mnemonics code is predefined, the hashing functions can be turned a priori The table may have the instructions format and length to decide where to put opcode bits, operand bits, offeset bits.

For variable instruction size
Used to calculates the address
Symbol table.
Stored and lookup to assign address to labels.

effecient insertion and retrival is needed.
deletion does not occurs.
Difficulties in hashing

non random keys
problem.

the size varies widely
pass1: loop until the end of the program
1. Read in a line of assembly code.
2. Assign an address to line

increament N(word addressign or byte address)

Page No.
Date

in symbol tables
4. Process assembler directives.

Constant declarations.
Space reservations.

Algorithm for Pass 1 assembler:

begin
    if starting address is given
        LOCCTR = starting address;
    else
        LOCCTR = 0;
    while OPCODE != END do ;; or EOF
        begin
        read a line from the code.
        if there is a label;
            if this label in in SYMTAB, then error.
        else insert (label, LOCCTR) into SYMTAB.
        Search OPTAB for the op code.
        if found.
            LOCCTR += N ;;
        else if this in assembly directive.
            update LOCCTR as directed.
        else error.
            write line to intermediates files
        end
    Program size = LOCCTR - starting address;
    end

Page No.
Date

Algorithm 4.1 (Assembler first pass):

1. loc_cntr := 0 (default value)
   pooltab_ptr := 1; POOLTAB[1] := 1;
   littab_ptr := 1;
2. While next statement is not an END statement
   (a) If label is present then.
      this_label := Symbol in label field;
      Enter (this_label, loc_cntr) in SYMTAB.
   (b) If an LTORG statement then
      (i) process literals LITTAB[POOLTAB[pooltab_ptr]] ... LITTAB[lit_tab_ptr-1] to allocate memory and put the address in the address field. Update loc_cntr accordingly.
      (ii) pooltab_ptr := pooltab_ptr + 1;
      (iii) POOLTAB[pooltab_ptr] := littab_ptr;
   (c) If a START or ORIGIN statement then.
      loc_cntr := value specified in operand field;
   (d) If an EQU statement then
      (i) this_addr := value of <address_spec>;
      (ii) Correct the symtab entry for this_label to (this_label, this_addr);
   (e) If a declaration statement then
      (i) code := code of declaration statement;
      (ii) size := size of memory area required by DC/DS.
      (iii) loc_cntr := loc_cntr + size.
      (iv) Generate IC (DL, code).
   (f) If an Imperative Statement then
      (i) code := machine opcode from OPTAB;

(ii) loc_cntr := loc cntr + Instruction length from OPTAB;

(iii) If operand is a literal then.

this_literal := literal on operand field;

LITTAB (littab_Ptr) := this_literal;

littab_ptr := littab_ptr + 1;

else (i.e operand is symbol)

this_entry := SYMTAB entry number of operand;

Generate IC (IS code) (S: this_entry);

3. (processing of END statement)

(a) perform Step 2 (b).

(b) Generates IC

(c) Go to pass II

Input.

```
START 200
MOVER AREG = '4'
MOVEM AREG, A
MOVER BREG = '1'
LOOP MOVER CREG, B.
 LTORG
ADD CREG = '6'
 STOP
A DS 1
B DS 1
END.
```

Expected output: Symbol table

```
A.    208
LOOP  203
B     209.
```

Intermediate code.

```
AD 01 C' 200.
IS 04 1 L 1
IS 05 1 S 1
IS 04 2 L 2
IS 04 3 S 3
AD 05 IS 00.
IS 01 3 L 3
DL 02 C 1.
DL 02 C 1.
AD 02.
```

Conclusions -

Thus, we have Implemented Pass-I Assembler using object oriented features.