**Assignment – Assembler Ready Reference**

An Assembler is a software processor that takes as input an assembly language program and translates it

into equivalent machine code if it is error free, otherwise provides a list of errors.

Format of assembly language statement is as follows

[Label] <Mnemonic Opcode> <Operand spec> [<Operand spec…..]

Assembler consists of three types of statements: Imperative, Declarative and Assembler Directives. The assembler should able to handle programs containing:-

Imperative statements with their opcode values as STOP=0, ADD=1, SUB=2, MULT=3, MOVER=4, MOVEM=5, COMP=6, BC=7, DIV=8, READ=9 and PRINT=10

Declarative statements with their values as DC=1 and DS=2 Assembler directives with their values as START=1 and END=2.

Assembler consists of four types of registers: AREG=1, BREG=2, CREG=3, DREG=4 Assembler consists of six types of condition code: LT=1, LE=2, EQ=3, GT=4, GE=5 and ANY=6

Use of Constants

Contrary to the name ‘declare constant’, the DC statement does not really implement constants, it merely initializes memory words to given values. These values are not protected by the assembler; they may be changed by moving a new value into the memory word.

An assembly program can use constants in two ways – immediate operands, and as literals. Immediate operands can be used in an assembly statement only if the architecture of the target machine includes the necessary features.

A literal is an operand with the syntax =’<value>’. It differs from a constant because its location cannot be specified in the assembly program. This helps to ensure that its value is not changed during execution of a program. It differs from an immediate operand because no architectural provision is needed to support its use. The value of the literal is protected by the fact that the name and address of this word is known to the assembly language programmer.

Design Specification of an assembler

1. Identify the information necessary to perform a task.
2. Design a suitable data structure to record the information
3. Determine the processing necessary to obtain and maintain the information
4. Determine the processing necessary to perform the task.

The Assembler has two phases: Analysis phase and Synthesis Phase.

Tasks performed by Analysis Phase

1. Isolate the label, mnemonic opcode and operand fields of a statement.
2. If a label is present, enter the pair (symbol, LC contents) in a new entry of symbol table.
3. Check validity of the mnemonic opcode through a look-up in the mnemonics table.
4. Perform LC processing, i.e. update the value contained in LC by considering the opcode and operands of the statements.

Tasks performed by Synthesis Phase

1. Obtain the machine opcode corresponding to the mnemonic from the Mnemonic table.
2. Obtain address of a memory operand from the symbol table.
3. Synthesize a machine instruction or the machine form of a constant, as the case may be.

The Assembler will have two passes. Tasks performed by the passes of a two pass assembler are as follows:

Pass I

1. Separate the symbol, mnemonic opcode and operand fields
2. Build the symbol table
3. Perform LC processing
4. Construct Intermediate code

Pass II

Synthesize the target program.

**Data Structure Design** – A design of assembler requires several tables such symbol table, mnemonic opcode table, intermediate code table and error table. Apart from these tables, the assembler uses a variable called ‘LC’ (location counter) pointing to the current line being processed.

There are three static tables - mnemonic opcode, general purpose registers and condition codes. The mnemonic opcode table stores a table of mnemonic opcodes and related information.

The intermediate code table stores intermediate code for each source line. Each table entry stores address, opcode, register operand, operand class. Operand class can be ‘S’ for symbol or ‘C’ for constant.

The error table stores line number and error description.

Symbol Table (SYMTAB)

Each symbol encountered in source program either in label field or memory operand field is added to symbol table. Each symbol table entry stores symbol name, address and two flags indicating whether symbol has been used and whether it is defined. When Symbol appears as label, it gets defined and corresponding address (i.e. current location counter value) gets added to the symbol table. It is used when a symbol appears as a memory operand. Symbol table contains unique entries of symbol.

Data Structures

|  |  |
| --- | --- |
| **Component** | **Description** |
| Mnemonic  table | Table that stores valid mnemonics and also the index matches the opcode or imperative  statements |
| Register  table | Table stores register names and index indicates the register number |
| Condition  Code table | Table stores condition names and index indicates the condition code |
| Symbol  table | Each entry in symbol table contains the name, address and flags indicating whether  the symbol is used and defined |
| Intermediate  code table | Each table entry stores address, opcode, register operand and memory operand |
| Error table | Each entry contains line number and the error description |

**Procedural Design** – The following table explains the input , algorithm and implementation hints for some of the procedures

|  |  |
| --- | --- |
| Procedure | Description |
| Assembler(main) | Input – source file as command line argument |
| Apply Pass one to source program. If no errors are found then apply Pass two |
| Pass one | Open the file  In a loop, Read a source program line Separate tokens  Process tokens  Generate Intermediate Code – IC |
| Pass two | In a loop, Get an entry from IC to generate target code. Refer symbol table |
| Separate tokens | Split the input string into strings Check if number of tokens are 4 If mnemonic is valid  If opcode requires two operands check validity of condition code check validity of register operand copy label, opcode and operands if invalid add error to error table Check if number of tokens are 3  Check if mnemonic requires two operands, validate and copy Check if number of tokens are 2 handle different possibilities Check if number of tokens are 1  It can be END or STOP |

|  |  |
| --- | --- |
| Process tokens | if label is present  add label to symbol table as defined along with lc if mnemonic is start  change lc  if mnemonic is DS  modify symbol table and change lc if mnemonic is DC  add appropriate entry IC table if imperative opcode  add operand2 to symbol table as used, if not present add to IC table. |
| Check opcode | Check if the mnemonic is present and return the index in the table as opcode else  return -1 |
| Add Symbol | Check if symbol is present and update or add the symbol to symbol table |
| Display Error  table | Display logged errors and corresponding error messages |

**Specifications – Assignment No. 1, 2, 3 and 4**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Input

1. Type all the input files with asm extension.
2. Accept the input file name to assembler as command line argument.

Output (With Reference to input program given in the assignment)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  | 105 |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| Assignment No.1 | Do as per Lab book |
| Assignment No. 2 | **Symbol Table**  **Symbol Address Used Defined**  X 103 1 1  LOOP 101 0 1  Y 106 1 1  **Errors**   1. Re-declaration of Symbol X 2. Symbol Z is used but not defined   **Warning**  1. Symbol LOOP is defined but not used  **Symbol table for Try it 1**  **Symbol Address Used Defined**  BEGIN 300 0 1  NUM 307 1 1  LOOP 301 1 1  HUNDRED 309 1 1  Warning: Symbol BEGIN defined but not used  **Symbol table for Try it 2**  **Symbol Address Used Defined**  A 107 1 1  B 108 1 1  LOOP 105 0 1  C - 1 0  HALT 106 0 1  **Errors**   1. Invalid Symbolic name ORIGIN 2. Invalid Symbolic name STOP 3. Invalid Symbolic name AREG 4. Symbol C is used but not defined 5. Re-declaration of Symbol A   **Warning**   1. Symbol LOOP is defined but not used 2. Symbol HALT is defined but not used |

|  |  |
| --- | --- |
| Assignment No.3 | **Intermediate Code Variant I**  <AD,1> <C,100>  <IS,4> 2 <S,1>  <IS,4> 1 <S,2>  <IS,4> 1 <S,3>  <IS,4> 4 <S,4>  <IS,4> 3 <S,2>  <IS,4> 2 <S,5>  <IS,10> <S,3>  <IS,0>  <DL,2> <C,1>  <DL,2> <C,2>  <DL,1> <C,7>  <DL,1> <C,1>  <DL,1> <C,4>  <AD,2> |
|  | **Intermediate Code Variant II**  <AD,1> <C,100>  <IS,4> BREG, SEVEN 0  <IS,4> AREG, ONE 1  <IS,4> AREG, A 2  <IS,4> DREG, FOUR 3  <IS,4> CREG, ONE 4  <IS,4> BREG, B 5  <IS,10> A 6  <IS,0> 7  <DL,2> <C,1> 8  <DL,2> <C,2> 9  <DL,1> <C,7> 10  <DL,1> <C,1>  <DL,1> <C,4>  <AD,2> |
| Assignment No.4 | **Target code should be stored in the file. Take File name from user.**  100 042111  101 041112  102 041108  103 044113  104 043112  105 042109  106 100108  107 000000 |

***Assignment no. 4 will be checked only when assignment no.3 is completed. Implementing intermediate code Variant I or II is student’s choice.***

Assignment No. 1 - Assembler 1

**Error Handling related to statements in the source program.**

Practical Assignment

1. Write an assembler to handle the following cases.
   1. If numbers of tokens are 4 i.e. <label> <mnemonic instruction> <register operand>, <memory operand> then
2. Print Error “Invalid Symbolic Name” if Label belongs to imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands.

E.g.

STOP/ADD/…./PRINT STOP/ADD/…./PRINT AREG, B DS/DC STOP/ADD/…./PRINT AREG, C

START/END/ORIGIN/EQU/LTORG STOP/ADD/…./PRINT ANY, HALT LT/LE/…./ANY STOP/ADD/…./PRINT AREG, D AREG/BREG/CREG/DREG STOP/ADD/…./PRINT AREG, ONE

1. Print Error “Invalid Mnemonic Instruction” if Mnemonic instruction belongs to Declarative statement, Assembler directive statement, Condition Codes or Register operands.

E.g.

LOOP DS/DC AREG, B

LOOP START/END/ORIGIN/EQU/LTORG BREG, B

LOOP LT/LE/…./ANY AREG, D

LOOP AREG/BREG/CREG/DREG AREG, ONE

1. Print Error “Invalid Register Operand” if Register operand belongs to imperative statement, Declarative statement, Assembler directive statement or Condition Codes.

E.g.

|  |  |  |
| --- | --- | --- |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | STOP/ADD/…./PRINT, B |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | DS/DC, C |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | START/END/ORIGIN/EQU/LTORG, HALT |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | LT/LE/…./ANY AREG, D |

1. Print Error “Invalid Symbolic Name” if memory operand belongs to imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands. E.g.

LOOP STOP/ADD/…/PRINT AREG, STOP/ADD/…/PRINT

LOOP STOP/ADD/…/PRINT BREG, DS/DC

LOOP STOP/ADD/…/PRINT CREG, START/END/ORIGIN/EQU/LTORG LOOP STOP/ADD/…/PRINT DREG, LT/LE/…/ANY

LOOP STOP/ADD/…/PRINT DREG, AREG/BREG/…/DREG

LOOP BC LT, imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands

* 1. If numbers of tokens are 3 i.e.
     + <label> <mnemonic instruction> <memory operand>
     + <label> DS/DC <constant> // constant here does not mean numerical constant
     + <mnemonic instruction> <register operand>, <memory operand> // BC LT, LOOP
     + <label> EQU <address specifiers> then

1. Print Error “Invalid Symbolic Name” if Label belongs to imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands. Also applicable to constant in case of DS / DC statement and address specifiers in case of EQU. E.g.

STOP/ADD/…./PRINT STOP/ADD/…./PRINT AREG, B DS/DC STOP/ADD/…./PRINT AREG, C

START/END/ORIGIN/EQU/LTORG STOP/ADD/…./PRINT ANY, HALT LT/LE/…./ANY STOP/ADD/…./PRINT AREG, D AREG/BREG/CREG/DREG STOP/ADD/…./PRINT AREG, ONE

1. Print Error “Invalid Mnemonic Instruction” if Mnemonic instruction belongs to Declarative statement, Assembler directive statement, Condition Codes or Register operands.

E.g.

LOOP DS/DC AREG, B

LOOP START/END/ORIGIN/EQU/LTORG BREG, B

LOOP LT/LE/…./ANY AREG, D

LOOP AREG/BREG/CREG/DREG AREG, ONE

1. Print Error “Invalid Register Operand” if Register operand belongs to imperative statement, Declarative statement, Assembler directive statement or Condition Codes.

E.g.

|  |  |  |
| --- | --- | --- |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | STOP/ADD/…./PRINT, B |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | DS/DC, C |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | START/END/ORIGIN/EQU/LTORG, HALT |
| LOOP | ADD/SUB/MULT/MOVER/MOVEM/COMP/DIV | LT/LE/…./ANY AREG, D |

1. Print Error “Invalid Symbolic Name” if memory operand belongs to imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands. E.g.

LOOP STOP/ADD/…/PRINT AREG, STOP/ADD/…/PRINT

LOOP STOP/ADD/…/PRINT BREG, DS/DC

LOOP STOP/ADD/…/PRINT CREG, START/END/ORIGIN/EQU/LTORG LOOP STOP/ADD/…/PRINT DREG, LT/LE/…/ANY

LOOP STOP/ADD/…/PRINT DREG, AREG/BREG/…/DREG

LOOP BC LT, imperative statement, Declarative statement, Assembler directive statement,

Condition Codes or Register operands

* 1. If numbers of tokens are 2 i.e.
     + <label> <mnemonic instruction>
     + <mnemonic Instruction> <constant>
     + <mnemonic instruction> <memory operand>
     + ORIGIN <address specifiers> then

1. Print Error “Invalid Symbolic Name” if Label belongs to imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands. Also applicable to constant in case of DS / DC statement and address specifiers in case of ORIGIN. E.g.

STOP/ADD/…./PRINT STOP/ADD/…./PRINT AREG, B DS/DC STOP/ADD/…./PRINT AREG, C

START/END/ORIGIN/EQU/LTORG STOP/ADD/…./PRINT ANY, HALT LT/LE/…./ANY STOP/ADD/…./PRINT AREG, D AREG/BREG/CREG/DREG STOP/ADD/…./PRINT AREG, ONE

1. Print Error “Invalid Mnemonic Instruction” if Mnemonic instruction belongs to Declarative statement, Assembler directive statement, Condition Codes or Register operands.

E.g.

LOOP DS/DC AREG, B

LOOP START/END/ORIGIN/EQU/LTORG BREG, B LOOP LT/LE/…./ANY AREG, D

LOOP AREG/BREG/CREG/DREG AREG, ONE

1. Print Error “Invalid Symbolic Name” if memory operand belongs to imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands. E.g.

LOOP STOP/ADD/…/PRINT AREG, STOP/ADD/…/PRINT LOOP STOP/ADD/…/PRINT BREG, DS/DC

LOOP STOP/ADD/…/PRINT CREG, START/END/ORIGIN/EQU/LTORG LOOP STOP/ADD/…/PRINT DREG, LT/LE/…/ANY

LOOP STOP/ADD/…/PRINT DREG, AREG/BREG/…/DREG

LOOP BC LT, imperative statement, Declarative statement, Assembler directive statement, Condition Codes or Register operands

* 1. If number of tokens are 1 i.e. <mnemonic instruction> then ensure that it is only STOP and LTORG.

Do you Know it ?

1. How mnemonic table will be implemented in C?
2. How you will declare register code table in C?
3. How you will declare condition code table in C?

Assignment Evaluation

0: Not Done

3: Needs Improvement

**Viva –Voce (0 to 2)**

1: Incomplete

4: Complete

2: Late Complete

5: Well Done

**Signature of the Teacher**

**Signature of Student**

**Date** / /

Try it!

1) Extend above program to discover any errors missing in the above set and implement it.

Assignment No. 2 - Assembler 2

**Symbol Table Handling: Errors or Warnings related to the Symbol Table.**

Practical Assignment

1. Write an assembler that will display the errors and warnings in the given program for example:

|  |  |  |  |
| --- | --- | --- | --- |
|  | START | 100 |  |
| READ | X |
| LOOP | MOVER | BREG, | X |
|  | ADD | BREG, | Y |
| X | MOVEM | BREG, | Z |
|  | STOP |  |  |
| X | DS 1 |  |  |
| Y | DS 1 |  |  |
|  | END |  |  |

Errors:

* 1. Symbol used but not defined
  2. Re-declaration of the Symbol Warnings

a. Symbol defined but not used

Do you Know it ?

1. Which data structures are used by the first pass of assembler?
2. Give the declaration for all above with respect to your program.
3. List the static and dynamic data structures of the assembler.

Assignment Evaluation

0: Not Done

3: Needs Improvement

**Viva –Voce (0 to 2)**

1: Incomplete

4: Complete

2: Late Complete

5: Well Done

**Signature of the Teacher**

**Signature of Student**

**Date** / /

Try it!

1. Show the contents of Symbol table and errors related to symbol table (if any) for the following assembly program.

START 300

BEGIN READ NUM

LOOP MOVEM AREG, NUM PRINT NUM

MULT AREG, NUM COMP AREG, HUNDRED BC LT, LOOP STOP

NUM DS 2

HUNDRED DC ‘100’ END

1. Show the contents of Symbol table and errors related to symbol table (if any) for the following assembly program.

|  |  |  |  |
| --- | --- | --- | --- |
|  | START | 100 |  |
| READ | A |
| READ | B |
| ORIGIN | MOVER | AREG, | A |
|  | ADD | AREG, | STOP |
|  | SUB | BREG, | AREG |
| LOOP | MOVEM | AREG, | C |
| HALT | STOP |  |  |
| A | DS | 1 |  |
| B | DS | 1 |  |
| A | DC | ‘1’ |  |
|  | END |  |  |

Assignment No. 3 - Assembler 3 Intermediate Code Generation

Practical Assignment

1) Write an assembler that will display the contents of intermediate code variant I for the following error- free assembly program:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| START | | | 100 |  |
| MOVER | | | BREG, | SEVEN |
| MOVER | | | AREG, | ONE |
| MOVER | | | AREG, | A |
| MOVER | | | DREG, | FOUR |
| MOVER | | | CREG, | ONE |
| MOVER | | | BREG, | B |
| PRINT | | | A |  |
| STOP | | |  |  |
| A | DS | 1 | | |
| B | DS | 2 | | |
| SEVEN | DC | ‘7’ | | |
| ONE | DC | ‘1’ | | |
| FOUR | DC | ‘4’ | | |
|  | END |  | | |

Do you Know it ?

1. Write the format of Intermediate Code.
2. Write a note on statement class used in intermediate code.
3. Explain the various operand classes used in intermediate code variant I.
4. How registers and condition codes are treated in variant I and II of intermediate code?
5. Compare intermediate code representation in variant I and II.
6. Give the intermediate code representation in variant I and II for the above program.

Assignment Evaluation

0: Not Done

3: Needs Improvement

**Viva –Voce (0 to 2)**

1: Incomplete

4: Complete

2: Late Complete

5: Well Done

**Signature of the Teacher**

**Signature of Student**

**Date** / /

Try it!

1) Write an assembler that will display the contents of intermediate code variant II for the following error-free assembly program:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| START | | | 100 |  |
| MOVER | | | BREG, | SEVEN |
| MOVER | | | AREG, | ONE |
| MOVER | | | AREG, | A |
| MOVER | | | DREG, | FOUR |
| MOVER | | | CREG, | ONE |
| MOVER | | | BREG, | B |
| PRINT | | | A |  |
| STOP | | |  |  |
| A | DS | 1 | | |
| B | DS | 2 | | |
| SEVEN | DC | ‘7’ | | |
| ONE | DC | ‘1’ | | |
| FOUR | DC | ‘4’ | | |
|  | END |  | | |

Assignment No. 4 - Assembler 4 Target Code Construction

Practical Assignment

1) Extend the assignment 3 to implement target code of the given program and store the target code in the file specified by the user.

Do you Know it ?

1. Explain: object code and executable code.
2. Write target code of the given program.