Tutorial 5 Answers – Made By U19CS012

1. What are the uses of OPTAB (Mnemonic operation code) and SYMTAB (Symbol table) during assembling process? Specify the uses of each during pass 1 and pass 2 of a two pass assembler.

- **Defn** - OPTAB is used to look up mnemonic operation codes and translate them to their machine language equivalents.
- During **Pass 1**, OPTAB is used to look up and validate operation coded in the source program and to find the instruction length for incrementing LOCCTR.
- In **Pass 2** , it is used to translate the operation codes to machine language

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.

- **Defn** - The Symbol table (SYMTAB) includes the name and value (address) for each label in the source program, together with flags to indicate error conditions (e.g., a symbol defined in two different places).

- During **Pass 1** of the assembler, labels are entered into SYMTAB as they are encountered in the source program, along with their assigned addresses (form LOCCTR).
- During **Pass 2**, symbols used as operands are looked up in SYMTAB to obtain the addresses to be inserted in the assembled instructions.

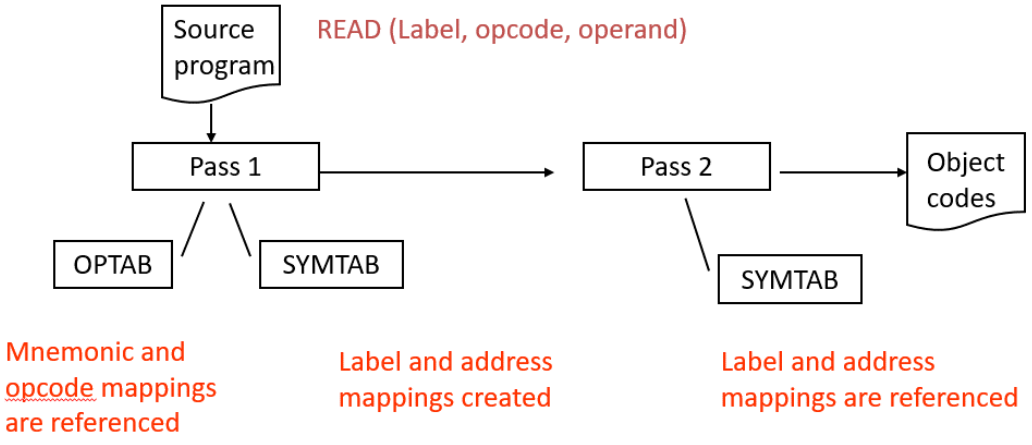
- **Characteristics** - Dynamic table (I.e., symbols may be inserted, deleted, or searched in the table)

- **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
-

Data Structures

- Operation Code Table (OPTAB)
  - Opcode, Instruction format, and length
  - Pass 1: Validate opcodes
  - Pass2: Assemble instructions
- Symbol Table (SYMTAB)
  - Label name and value, error flags
  - Pass 1: Created!
  - Lookup symbols to insert in assembled instr.
- Location Counter
  - Initialed to the Org or End

A Simple Two Pass Assembler Implementation



Hash Tables

- OPTAB is static (access)
  - Retrieval efficiency
  - Key : Mnemonic operation
- SYMTAB (add, access)
  - Insertion and Retrieval efficiency
  - Key: Label Name
    - LOOP1, LOOP2, LOOP3…, A, X, Y, Z…

2. What are assembler directives? List any three assembler directives.

Assembler directives are instructions that direct the assembler to do something

Directives do many things; some tell the assembler to set aside space for variables, others tell the assembler to include additional source files, and others establish the start address for your program.

Assemble directives can’t generate machine code.

The directives available are shown below:

1. **START < constant >** :- This directive indicates that the first word of the target program will start on ROM memory location with address < constant > . START < 200> ROM location will be 200 where first machine code will reside.
2. **END Directive :-** This directive indicates the end of the source program.

Advanced Assembler Directives :-

1) ORIGIN :- The syntax of this directive is

ORIGIN < address spec >

This directive indicates that LC (location counter) should be set to the address given by

< address spec >. The ORIGIN statement is useful when the target program does not consist of consecutive memory words.

e.g ORIGIN 200 :- Location counter will shift to ROM location 200.

2) EQU :- The syntax of this directive is

< symbol > EQU < address spec >

e.g A EQU 100 :- A is assigned to address spec 100.

Assembler Directive

- Assembler directive instruct the assembler to perform certain actions during assembly of a program.
- Some assembler directive are:
- START <address constant>

- END

Advanced Assembler Directives

- 1. ORIGIN
- 2. EQU
- 3. USING
- 4. DROP
- 5. LTORG

Advanced Assembler Directives

- ORIGIN
- EQU
- LTORG

Link - https://www.slideshare.net/ReshmaKapadi/system-programming-unit-1-introduction-86502311

3. Find out addresses of variable using LC.

Step 1: First identify all variables in your program.
Step2: Replace all symbolic address with numeric address.
Step3: Replace symbolic opcodes by machine operation codes.

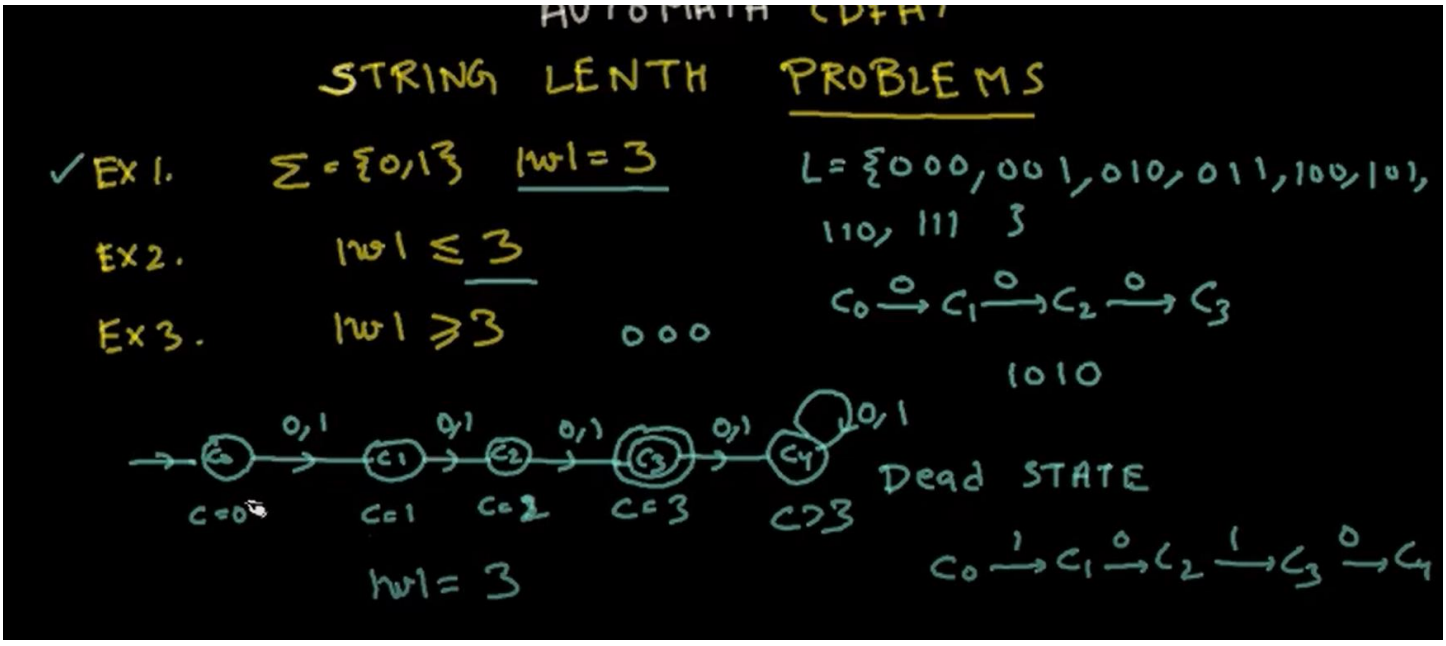
```
START 101
READ N
MOVER BREG, ONE
MOVEM BREG, TERM
AGAIN MULT BREG, TERM
MOVER CREG, TERM
ADD CREG, ONE
MOVEM CREG, TERM
COMP CREG, N
BC LE, AGAIN
MOVEM BREG, RESULT
PRINT RESULT
STOP
N DS 1
RESULT DS 1
ONE DC '1'
TERM DS 1
```

https://www.slideshare.net/BhattBalkrishna/ch-3-assembler-in-system-programming

	START	101	
	READ	N	101) + 09 0 113
	MOVER	BREG, ONE	102) + 04 2 115
	MOVEM	BREG, TERM	103) + 05 2 116
AGAIN	MULT	BREG, TERM	104) + 03 2 116
	MOVER	CREG, TERM	105) + 04 3 116
	ADD	CREG, ONE	106) + 01 3 115
	MOVEM	CREG, TERM	107) + 05 3 116
	COMP	CREG, N	108) + 06 3 113
	BC	LE, AGAIN	109) + 07 2 104
	MOVEM	BREG, RESULT	110) + 05 2 114
	PRINT	RESULT	111) + 10 0 114
	STOP		112) + 00 0 000
N	DS	1	113)
RESULT	DS	1	114)
ONE	DC	'1'	115)
TERM	DS	1	116)
	END		

https://www.slideshare.net/ReshmaKapadi/system-programming-unit-1-introduction-86502311

4. Design an automata for set of all strings of length 5.

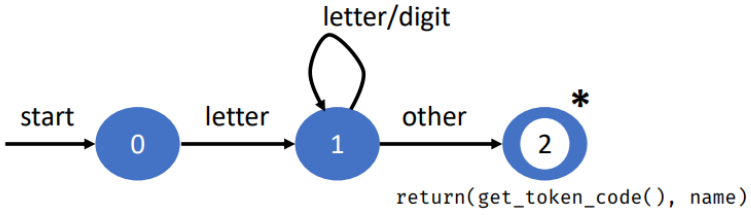


5. Design an automata for identifying constants and keywords.

Examples of Transition Diagrams

Identifiers and reserved words

$letter = [a-zA-Z]$
$digit = [0-9]$
$identifier = letter(letter|digit)^*$



- * indicates a retraction state
- `get_token_code()` searches a table to check if the name is a reserved word and returns its integer code if so
- Otherwise, it returns the integer code of the IDENTIFIER token, with `name` containing the string of characters forming the token
  - Name is not relevant for reserved words