

Chapter 2: Assemblers

Mrs. Sunita M Dol (Aher),
Assistant Professor,
Computer Science and Engineering Department,
Walchand Institute of Technology, Solapur, Maharashtra

2. Assemblers

- Elements of Assembly Language Programming
- A simple Assembly Scheme
- Pass Structure of Assemblers
- Design of a two pass Assembler
- A Single Pass Assembler for IBM PC

2. Assemblers

- Elements of Assembly Language Programming
- A simple Assembly Scheme
- Pass Structure of Assemblers
- Design of a two pass Assembler
- A Single Pass Assembler for IBM PC

Assembly Language Programming

- An assembly language is a machine dependent, low level programming language which is specific to a certain computer system.
- Compared to machine language of computer, it provides three basic features:
 - Mnemonic operation codes
 - Eliminates the need to memorize numeric operation code.
 - Symbolic operands
 - Symbolic names can be used.
 - Data declaration
 - Data can be declared in any form e.g. -5, 10.5 etc.

Assembly Language Programming

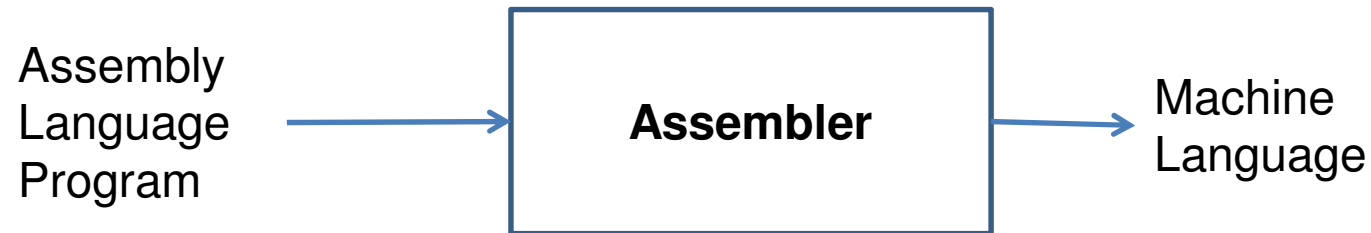


Figure: Assembler

Assembly Language Programming

- Statement Format

[Label] <Opcode> <operand spec> [<operand spec>....]

1. Label:- Is optional.
2. Opcode:- Symbolic opcode
3. Operand Spec:- *<operand spec>* has the following syntax
<symbolic name> [+<displacement>][(<index register>)]

e.g. AREA

AREA + 5

AREA (4)

AREA + 5(4)

Assembly Language Programming

- A Simple Assembly Language
 - Each statement has two operands
 - The first operand is always a register (AREG, BREG, CREG and DREG)
 - The second operand refers to a memory word.

Assembly Language Programming

Instruction Opcode	Assembly Mnemonic	Remarks
00	STOP	Stop Execution
01	ADD	$Op1 \leftarrow Op1 + Op2$
02	SUB	$Op1 \leftarrow Op1 - Op2$
03	MULT	$Op1 \leftarrow Op1 * Op2$
04	MOVER	CPU Reg \leftarrow Memory operand
05	MOVEM	Memory \leftarrow CPU Reg
06	COMP	Sets Condition Code
07	BC	Branch on Condition
08	DIV	$Op1 \leftarrow Op1 / Op2$
09	READ	Operand 2 \leftarrow input Value
10	PRINT	Output \leftarrow Operand2

Figure: Mnemonic Operation Codes

Assembly Language Programming

- Instruction Format



Figure: Instruction Format

Assembly Language Programming

- Assembly Language to Machine Language
 - Find address of variables and labels.
 - Replace Symbolic address by numeric address.
 - Replace Symbolic opcode by machine opcode.
 - Reserve storage for data.

Assembly Language Programming

	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
	END	

Figure: Sample program to find n!

Assembly Language 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	116
	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) + 00 0 001	
TERM	DS	1	—————→	116)	
	END				

Assembly Language Programming

Variable	Address
AGAIN	104
N	113
RESULT	114
ONE	115
TERM	116

Figure: After LC Processing

Assembly Language Programming

- Machine Code

LC	Opcode	Register	Address
101	09	0	113
102	04	2	115
103	05	2	116
104	03	2	116
105	04	3	116
106	01	3	115
107	05	3	116
108	06	3	113
109	07	2	104
110	05	2	114
111	10	0	114
112	00	0	000
113			
114			
115	00	0	001
116			

Assembly Language Programming

```
START 101
READ      X
READ      Y
MOVER     AREG, X
ADD       AREG, Y
MOVEM     AREG, RESULT
PRINT RESULT
STOP
X DS      1
Y DS      1
RESULT DS  1
END
```

Figure: Sample program to find X+Y

Assembly Language Programming

Program				LC	Opcode	Register	Memory operand
START	101						
READ	X	→		101	+	09 0	108
READ	Y	→		102	+	09 0	109
MOVER	AREG, X	→		103	+	04 1	108
ADD	AREG, Y	→		104	+	01 1	109
MOVEM	AREG, RESULT	→		105	+	05 0	110
PRINT	RESULT	→		106	+	10 0	110
STOP		→		107	+	00 0	000
X	DS		1	→			
Y	DS		1	→			
RESULT	DS		1	→			
END							

Assembly Language Programming

Variable	Address
X	108
Y	109
RESULT	110

Figure: After LC Processing

Assembly Language Programming

- Machine Code

LC	Opcode	Register	Address
101	09	0	108
102	09	0	109
103	04	1	108
104	01	1	109
105	05	0	110
106	10	0	110
107	00	0	000
108			
109			
110			
111			

Assembly Language Programming

- Assembly Language Statement
 - Imperative Statement.
 - Indicates an action to be taken during execution of a program.
 - Eg: MOV, ADD, MULT, etc.
 - Declaration Statement.
 - To reserve memory for variable.
 - [Label] DS <constant> eg: X DS 5
 - [Label] DC '<value>' eg: X DC '3'
 - Assembler Directives
 - Instructs the assembler to perform certain action during assembly of a program.
 - START <constant>
 - END

Assembly Language Programming

- Literals and Constants
 - Literal cannot be changed during program execution
 - Literal is more safe and protected than a constant.
 - Literals appear as a part of the instruction.
 - e.g.

ADD	AREG,	= '5'	=>	ADD	AREG,	FIVE

				FIVE	DC	'5'

Assembly Language Programming

- Assembler Directives
 - START <constant> : The first word of the target program by the assembler should be placed in the memory with the address <constant>
e.g. START 100
 - END [<operand spec>] : It indicates the end of source program. The optional <operand spec> indicates the address of instruction where the execution of the program should begin.

Assembly Language Programming

- Advantages of Assembly Language

Program				LC	Opcode	Register	Memory operand
START	101						
READ	X	→		101	+	09 0	108
READ	Y	→		102	+	09 0	109
MOVER	AREG, X	→		103	+	04 1	108
ADD	AREG, Y	→		104	+	01 1	109
DIV	AREG, TWO	→		105	+	08 2	112
MOVEM	AREG, RESULT	→		106	+	05 0	110
PRINT	RESULT	→		107	+	10 0	110
STOP		→		108	+	00 0	000
X	DS	1	→	109			
Y	DS	1	→	110			
RESULT	DS	1	→	111			
TWO	DC	'2'	→	112	+	00 0	002
END							

2. Assemblers

- Elements of Assembly Language Programming
- A Simple Assembly Scheme
- Pass Structure of Assemblers
- Design of a two pass Assembler
- A Single Pass Assembler for IBM PC

A Simple Assembly Scheme

- Design Specification for an Assembler
 - Four step approach
 - Identify the information necessary to perform a task.
 - Design a suitable data structure to record the information.
 - Determine the processing necessary to obtain and maintain the information.
 - Determine the processing necessary to perform the task.

A Simple Assembly Scheme

- Synthesis Phase
 - Consider instruction MOVER AREG, X

	START	101		LC				
	READ	X	————→	101	+	09	0	108
	READ	Y	————→	102	+	09	0	109
	MOVER	AREG, X	————→	103	+	04	1	108
	ADD	AREG, Y	————→	104	+	01	1	109
	MOVEM	AREG, RESULT	————→	105	+	05	0	110
	PRINT	RESULT	————→	106	+	10	0	110
	STOP		————→	107	+	00	0	000
X	DS	1	————→	108				
Y	DS	1	————→	109				
RESULT	DS	1	————→	110				
	END							

A Simple Assembly Scheme

- Synthesis Phase
 - we must have the following information to synthesize the machine instruction corresponding to this statement:
 - Address of the memory word with which name X is associated.
 - Machine operation code corresponding to the mnemonic MOVER.
 - The first item of information depends on the source program. Hence it must be available by the analysis phase. The second item of information does not depend on the source program

A Simple Assembly Scheme

- Synthesis Phase
 - we consider the use of two data structures during the synthesis phase:
 - Symbol table
 - Mnemonics table

A Simple Assembly Scheme

- Analysis Phase
 - The primary function performed by the analysis phase is the building of the symbol table.
 - For this purpose it must determine the address with which the symbolic names in a program are associated.
 - To determine the address of any instruction, we must fix the address of all program elements preceding it. This function is called memory allocation.
 - To implement memory allocation a data structure called location counter is introduced.

A Simple Assembly Scheme

- Analysis Phase
 - It is initialized to the constant specified in the START statement.
 - The location counter is always made to contain the address of the next memory word in the target program.

A Simple Assembly Scheme

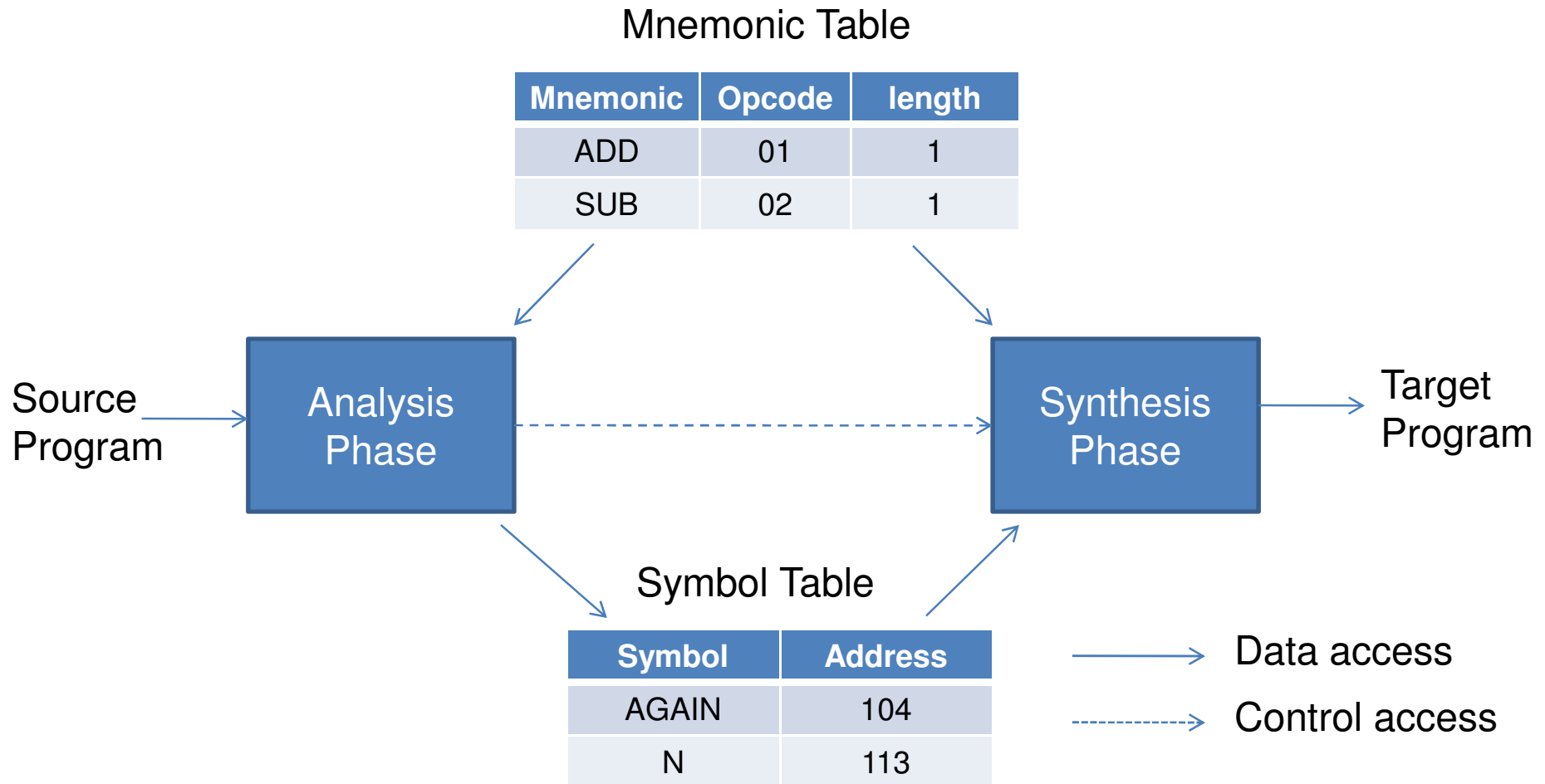


Figure: Overview of Two Pass Assembler

A Simple Assembly Scheme

- Analysis Phase
 - Isolate the label, mnemonic opcode and operand fields of a statement.
 - If a label is present, enter the pair(symbol, <LC counter>) in a new entry of the symbol table
 - Check the validity of the mnemonic opcode through a look-up in the mnemonic table.
 - Perform LC processing i.e. update the value contained in LC by considering the opcode and operands of the statement.

A Simple Assembly Scheme

- Synthesis phase
 - Obtain the machine opcode corresponding to the mnemonic from mnemonics table.
 - Obtain address of a memory operand from the symbol table.
 - Synthesize a machine instruction or the machine form of a constant, as the case may be.

2. Assemblers

- Elements of Assembly Language Programming
- A Simple Assembly Scheme
- **Pass Structure of Assemblers**
- Design of a two pass Assembler
- A Single Pass Assembler for IBM PC

Pass Structure of Assembler

- Two pass translation
 - Two pass translation of an assembly language program can handle forward references easily.
 - LC processing is performed in the first pass and symbols defined in the program are entered into the symbol table.
 - The second pass synthesizes the target form using the address information found in the symbol table.

Pass Structure of Assembler

- Two pass translation
 - The first pass constructs an intermediate representation of the source program for use by the second pass .
 - This representation consists of two main components—
 - data structures, e.g. the symbol table, and
 - a processed form of the source program

Pass Structure of Assembler

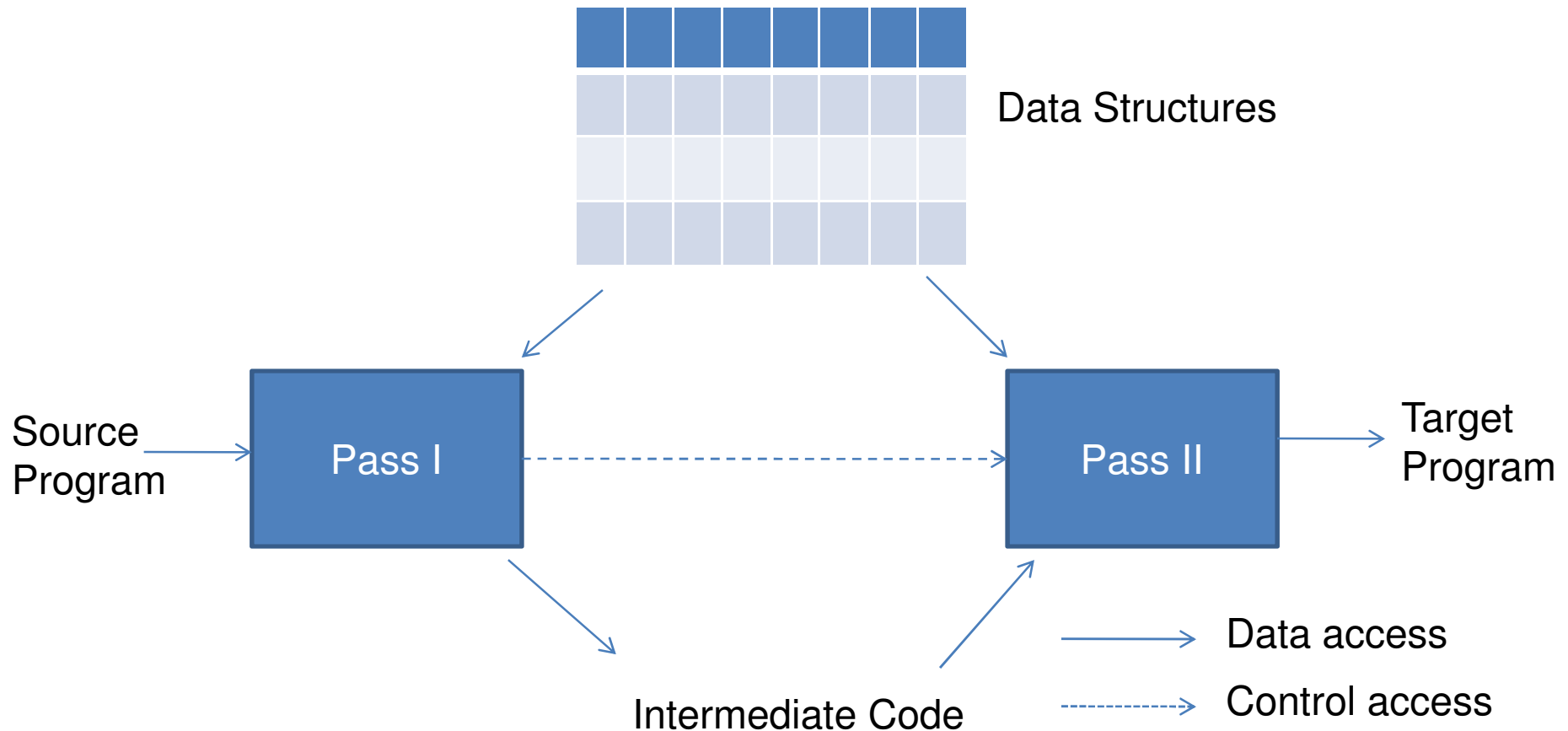


Figure: Overview of Two Pass Assembler

Pass Structure of Assembler

- Single pass translation
 - LC processing and construction of the symbol table proceed as in two pass translation.
 - The problem of forward references is tackled using a process called backpatching.
 - The operand field of an instruction containing a forward reference is left blank initially.
 - The address of the forward referenced symbol is put into this field when its definition is encountered.

Pass Structure of Assembler

- Single pass translation
 - The need for inserting the operand's address at a later stage can be indicated by adding an entry to the table of incomplete instructions (TII).
 - In TII, each entry is a pair (<instruction address>,<symbol>)

Pass Structure of Assembler

- Single pass translation
 - By the time the END statement is processed, the symbol table would contain the addresses of all symbols defined in the source program and TII would contain information describing all forward references.
 - The assembler can now process each entry in TII to complete the concerned instruction.

Pass Structure of Assembler

- Single Pass Translation
 - The problem of forward reference can be handled using a technique called as back patching.
 - The need for inserting the second operand's address at a later stage can be indicated by adding an entry to the Table of Incomplete Instruction (TII)
 - The entry in TII is a pair
(<instruction address>, <symbol>)

Pass Structure of Assembler

- Single Pass Translation Example

	START	100
	MOVER	AREG, X
	ADD	BREG, ONE
	ADD	CREG, TEN
	STOP	
X	DC	'5'
ONE	DC	'1'
TEN	DC	'10'
	END	

Pass Structure of Assembler

- Single Pass Translation Example

```

START 100
MOVER AREG, X      100    04 1  _ _ _
ADD    BREG, ONE   101    01 2  _ _ _
ADD    CREG, TEN   102    06 3  _ _ _
STOP                   103    00 0  000
X      DC    '5'     104
ONE    DC    '1'     105
TEN    DC    '10'    106
END
    
```

Instruction Address	Symbol Making a forward reference
100	X
101	ONE
102	TEN

Figure : TII (Table of Incomplete Instruction)

Symbol	Address
X	104
ONE	105
TEN	105

Figure : Symbol Table

Pass Structure of Assembler

- Single Pass Translation Example

	START	100			
	MOVER	AREG, X	100	04 1	<u>104</u>
	ADD	BREG, ONE	101	01 2	<u>105</u>
	ADD	CREG, TEN	102	06 3	<u>106</u>
	STOP		103	00 0	000
X	DC	'5'	104		
ONE	DC	'1'	105		
TEN	DC	'10'	106		
	END				

2. Assemblers

- Elements of Assembly Language Programming
- A Simple Assembly Scheme
- Pass Structure of Assemblers
- Design of a Two Pass Assembler
- A Single Pass Assembler for IBM PC

Design of a Two Pass Assembler

- Pass I:-
 1. Separate the symbol, mnemonic, opcode and operand.
 2. Build Symbol Table.
 3. Perform LC Processing.
 4. Construct Intermediate Representation.
- Pass II:-
 1. Process IR to synthesize the target program.

Design of a Two Pass Assembler

- Advanced Assembler Directives
 - ORIGIN:
 - The syntax is
 ORIGIN <address specification>
 where address specification is an <operand spec> or <constant>
 - This directives indicates that LC should be set to the address given by <address spec>.
 - The ORIGIN is useful when the target program does not consist of consecutive memory word.

Design of a Two Pass Assembler

1		START	200	
2		MOVER	AREG, ='5'	200) + 04 1 211
3		MOVEM	AREG, A	201) + 05 1 217
4	LOOP	MOVER	AREG, A	202) + 04 1 217
5		MOVER	CREG, B	203) + 05 3 218
6		ADD	CREG, ='1'	204) + 01 2 212
7			
12		BC	ANY, NEXT	210) + 07 6 214
13		LTORG		
			= '5'	211) + 00 0 005
			= '1'	212) + 00 0 001
			
14		SUB	AREG, ='1'	214) + 02 1 219
15	NEXT	BC	LT, BACK	215) + 07 1 202
16		STOP		216) + 00 0 000
17	LAST			
18		ORIGIN	LOOP+2	
19		MULT	CREG, B	204) + 03 3 218
20		ORIGIN	LAST+1	
21	A	DS	1	217)
22	BACK	EQU	LOOP	
23	B	DS	1	218)
24		END		
25			= '1'	219)

Statement number 18 sets LC to the value 204 since the symbol LOOP is associated with the address 202

Figure : An assembly language illustrating ORIGIN

Design of a Two Pass Assembler

- Advanced Assembler Directives
 - EQU:
 - The syntax is
 <symbol> ORIGIN <address specification>
 where address specification is an <operand spec> or
 <constant>
 - The EQU statement defines the symbol to represent <address spec>.
 - No LC processing is implied.

Design of a Two Pass Assembler

1		START	200	
2		MOVER	AREG, ='5'	200) + 04 1 211
3		MOVEM	AREG, A	201) + 05 1 217
4	LOOP	MOVER	AREG, A	202) + 04 1 217
5		MOVER	CREG, B	203) + 05 3 218
6		ADD	CREG, ='1'	204) + 01 2 212
7			
12		BC	ANY, NEXT	210) + 07 6 214
13		LTORG		
			= '5'	211) + 00 0 005
			= '1'	212) + 00 0 001
14			
15	NEXT	SUB	AREG, ='1'	214) + 02 1 219
16		BC	LT, BACK	215) + 07 1 202
17	LAST	STOP		216) + 00 0 000
18		ORIGIN	LOOP+2	
19		MULT	CREG, B	204) + 03 3 218
20		ORIGIN	LAST+1	
21	A	DS	1	217)
22	BACK	EQU	LOOP	
23	B	DS	1	218)
24		END		
25			= '1'	219)

Statement
22
introduces
the symbol
BACK to
represent
the operand
LOOP

Figure : An assembly language illustrating ORIGIN

Design of a Two Pass Assembler

- Advanced Assembler Directives
 - LTORG:
 - The LTORG statement permits a programmer to specify where literals should be placed .
 - By default, assembler places the literals after the END statement.
 - At every LTORG statement, as also at the END statement, the assembler allocates memory to the literals of a literal pool.
 - The pool contains all literals used in the program since start of the program or since the last LTORG statement.

Design of a Two Pass Assembler

1		START	200	
2		MOVER	AREG, ='5'	200) + 04 1 211
3		MOVEM	AREG, A	201) + 05 1 217
4	LOOP	MOVER	AREG, A	202) + 04 1 217
5		MOVER	CREG, B	203) + 05 3 218
6		ADD	CREG, ='1'	204) + 01 2 212
7			
12		BC	ANY, NEXT	210) + 07 6 214
13		LTORG		
			= '5'	211) + 00 0 005
			= '1'	212) + 00 0 001
14			
15	NEXT	SUB	AREG, ='1'	214) + 02 1 219
16		BC	LT, BACK	215) + 07 1 202
17	LAST	STOP		216) + 00 0 000
18		ORIGIN	LOOP+2	
19		MULT	CREG, B	204) + 03 3 218
20		ORIGIN	LAST+1	
21	A	DS	1	217)
22	BACK	EQU	LOOP	
23	B	DS	1	218)
24		END		
25			= '1'	219)

The literals ='5' and ='1' are added to the literal pool in the statement 2 and 6 respectively. The first LTORG statement allocates the addresses 211 and 212 to the values '5' and '1'. A new literal pool is now started. The value '1' is put into this pool in statement 15.

Figure : An assembly language illustrating ORIGIN

Design of a Two Pass Assembler

	Program	LC	
	START 200		
	MOVER AREG, ='5'	200	+04 1 205
	MOVEM AREG, X	201	+05 1 214
L1	MOVER BREG, ='2'	202	+04 2 206
	ORIGIN L1+3		
	LTORG		
		205	+00 0 005
		206	+00 0 002
NEXT	ADD AREG,='1'	207	+01 1 210
	SUB BREG,='2'	208	+02 2 211
	BC LT, BACK	209	+07 1 202
	LTORG		
		210	+00 0 001
		211	+00 0 002
BACK	EQU L1		
	ORIGIN NEXT+5		
	MULT CREG,='4'	212	+03 3 215
	STOP	213	+00 0 000
X	DS 1	214	
	END	215	+00 0 004

Design of a Two Pass Assembler

- Pass I uses the following data structures
 1. Machine Opcode table (OPTAB)
 2. Symbol Table (SYMTAB)
 3. Literal Table (LITTAB)
 4. Pool Table (POOLTAB)

Design of a Two Pass Assembler

- Pass-I Data Structures
 - OPTAB
 - OPTAB contains the fields
 - **Mnemonic opcode**
 - **Class** : the class indicate whether the opcode corresponding to an imperative statement (IS), a declaration statement (DL) or an assembler directives (AD).
 - **Mnemonic info**: If an imperative statement, the mnemonic info field contains the pair (machine opcode, instruction length) else it contains the id of a routine to handle the declaration or directive statement.
 - A SYMTAB entry contains the fields address and length.
 - A LITTAB entry contains the fields literal and address.

Design of a Two Pass Assembler

	Program	LC	
	START 200		
	MOVER AREG, ='5'	200	+04 1 205
	MOVEM AREG, X	201	+05 1 214
L1	MOVER BREG, ='2'	202	+04 2 206
	ORIGIN L1+3		
	LTORG		
		205	+00 0 005
		206	+00 0 002
NEXT	ADD AREG,='1'	207	+01 1 210
	SUB BREG,='2'	208	+02 2 211
	BC LT, BACK	209	+07 1 202
	LTORG		
		210	+00 0 001
		211	+00 0 002
BACK	EQU L1		
	ORIGIN NEXT+5		
	MULT CREG,='4'	212	+03 3 215
	STOP	213	+00 0 000
X	DS 1	214	
	END	215	+00 0 004

2. Assemblers

- Elements of Assembly Language Programming
- A Simple Assembly Scheme
- Pass Structure of Assemblers
- Design of a Two Pass Assembler
- A Single Pass Assembler for IBM PC

Algorithm for PASS-I

1) Loc_cntr:=0;(default value)

Pooltab_ptr:=;POOLTAB[1]:=1;

Littab_ptr:=1;

2) While next statement is not an END statement

a) If label is present

This_label :=symbol in label field:

Enter(this_label,loc_cntr)in SYMTAB

b) If an LORG statement then

- i) Process literals LITTAB[POOLTAB[pooltab_ptr]]
...LITTAB[littab_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 the 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

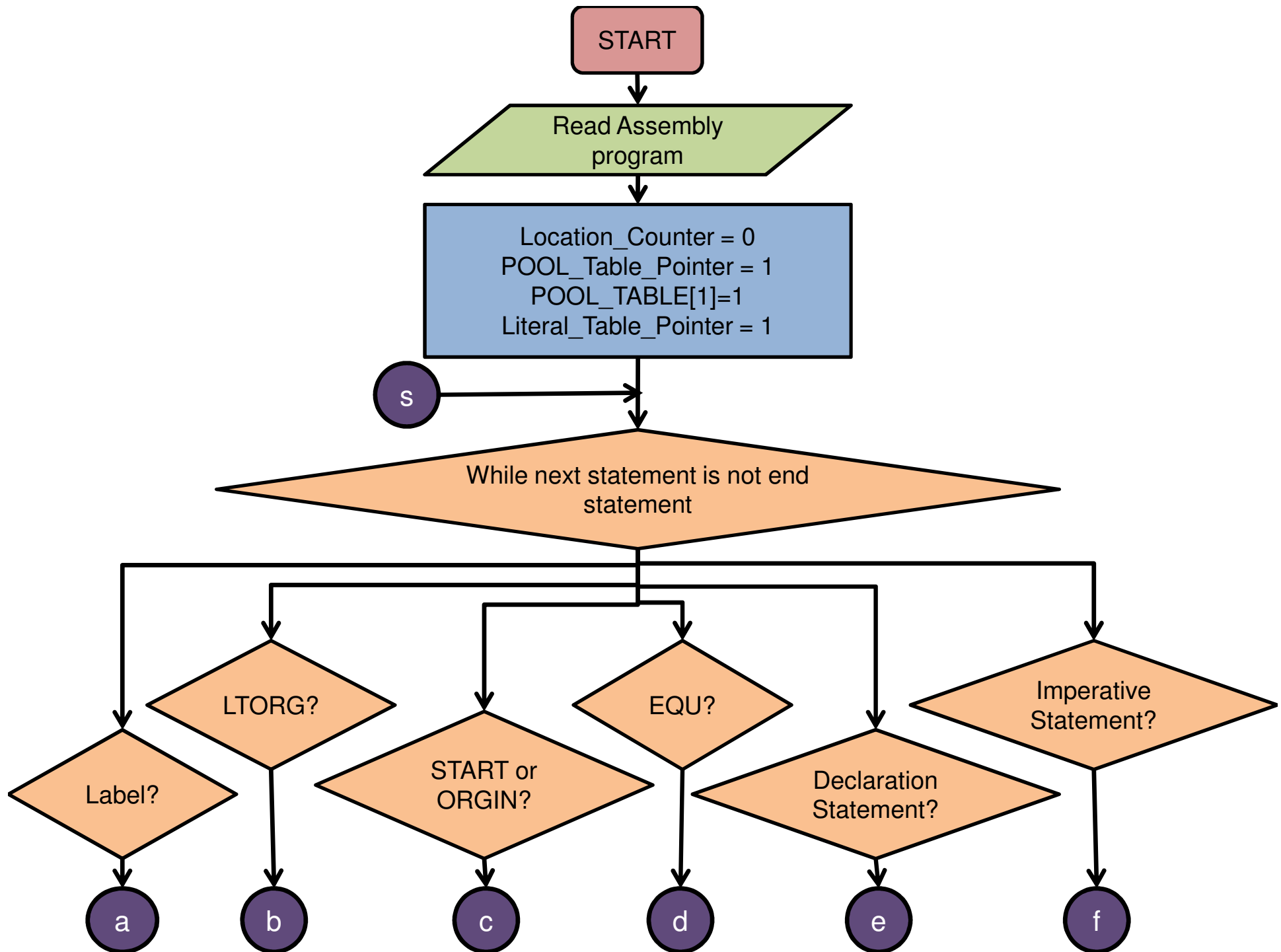
- i) Code:=machine opcode from OPTAB;
- ii) Loc_cntr := loc_cntr + instruction length from OPTAB;
- iii) If operand is literal then
 - this_literal:=literal in operand field;
 - LITTAB[littab_ptr]:=this_literal;
 - littab_ptr:=littab_ptr+1;
- Else (i.e. operand is a symbol)
 - this_entry := SYMTAB entry number of operand;
- Generate IC '(IS,code)(S,this_entry)';

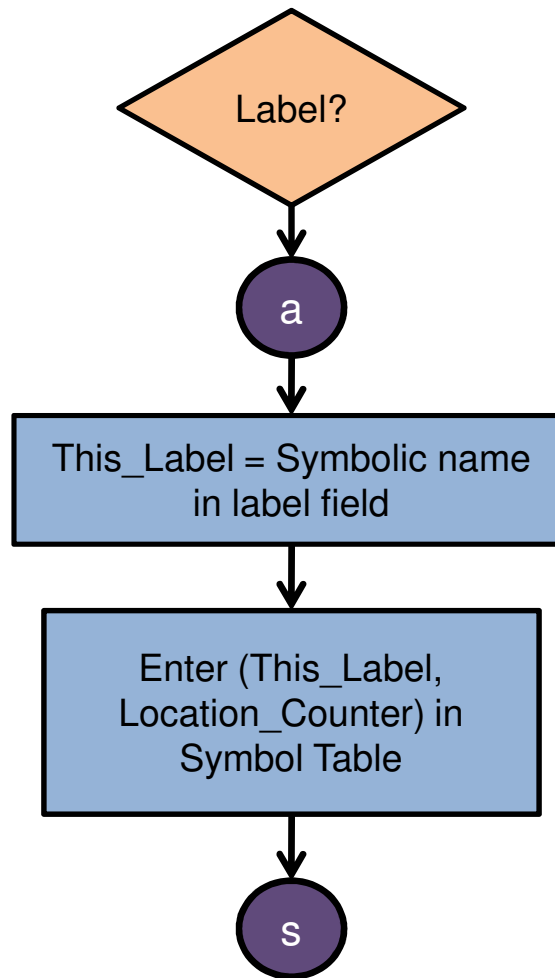
3) (Processing of END statement)

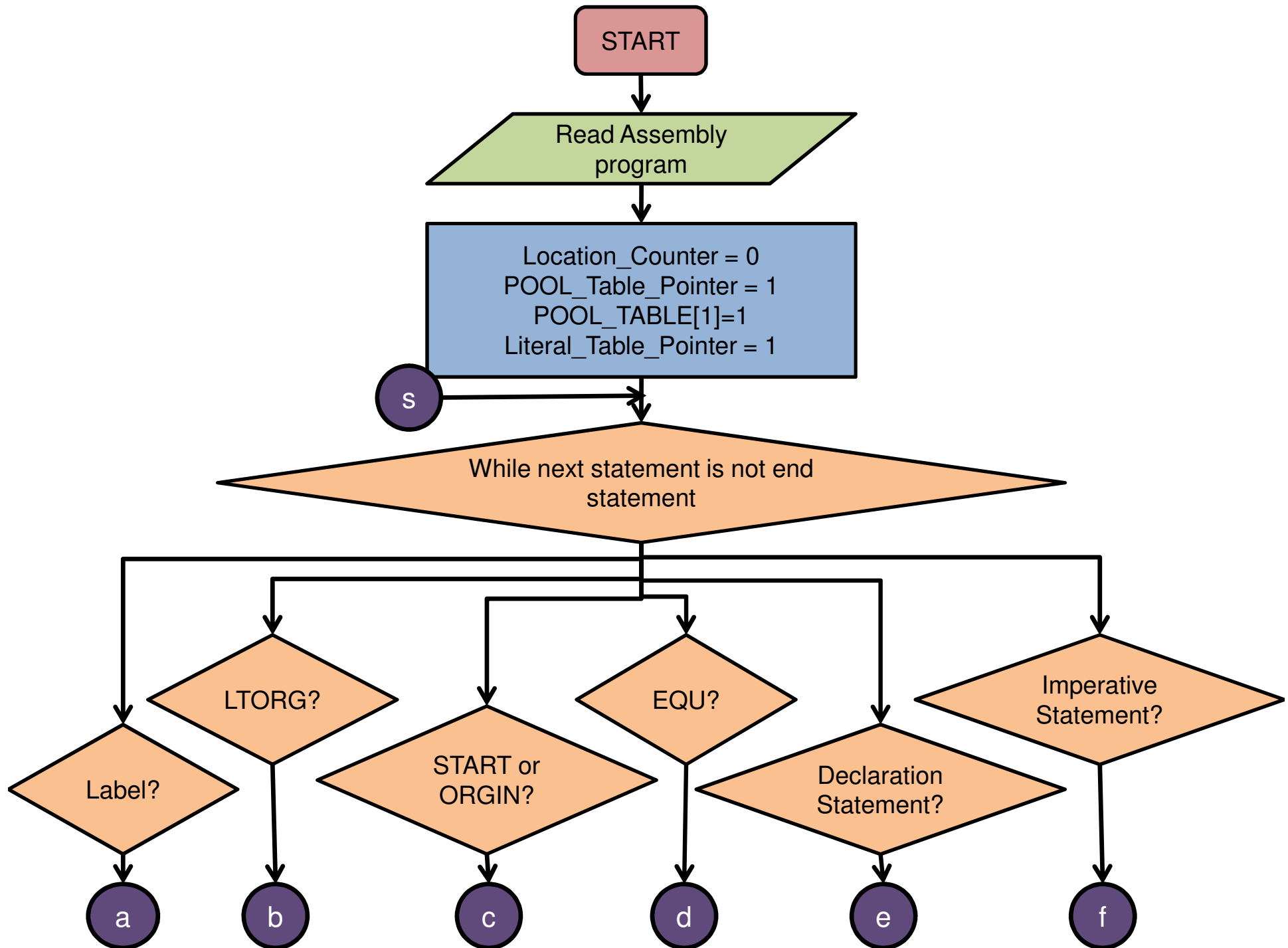
a)Perform step 2(b).

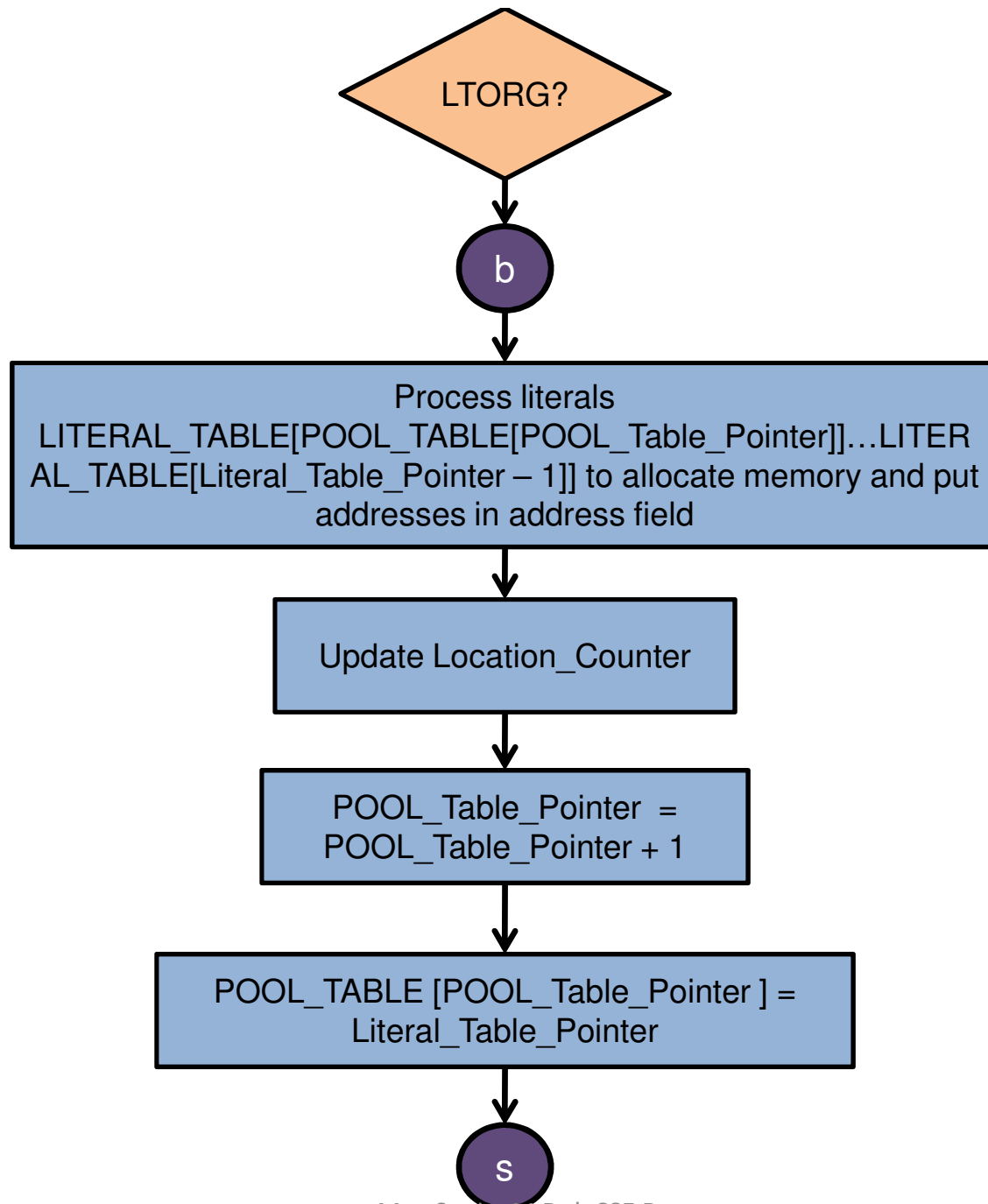
b)Generate IC '(AD,02)'.

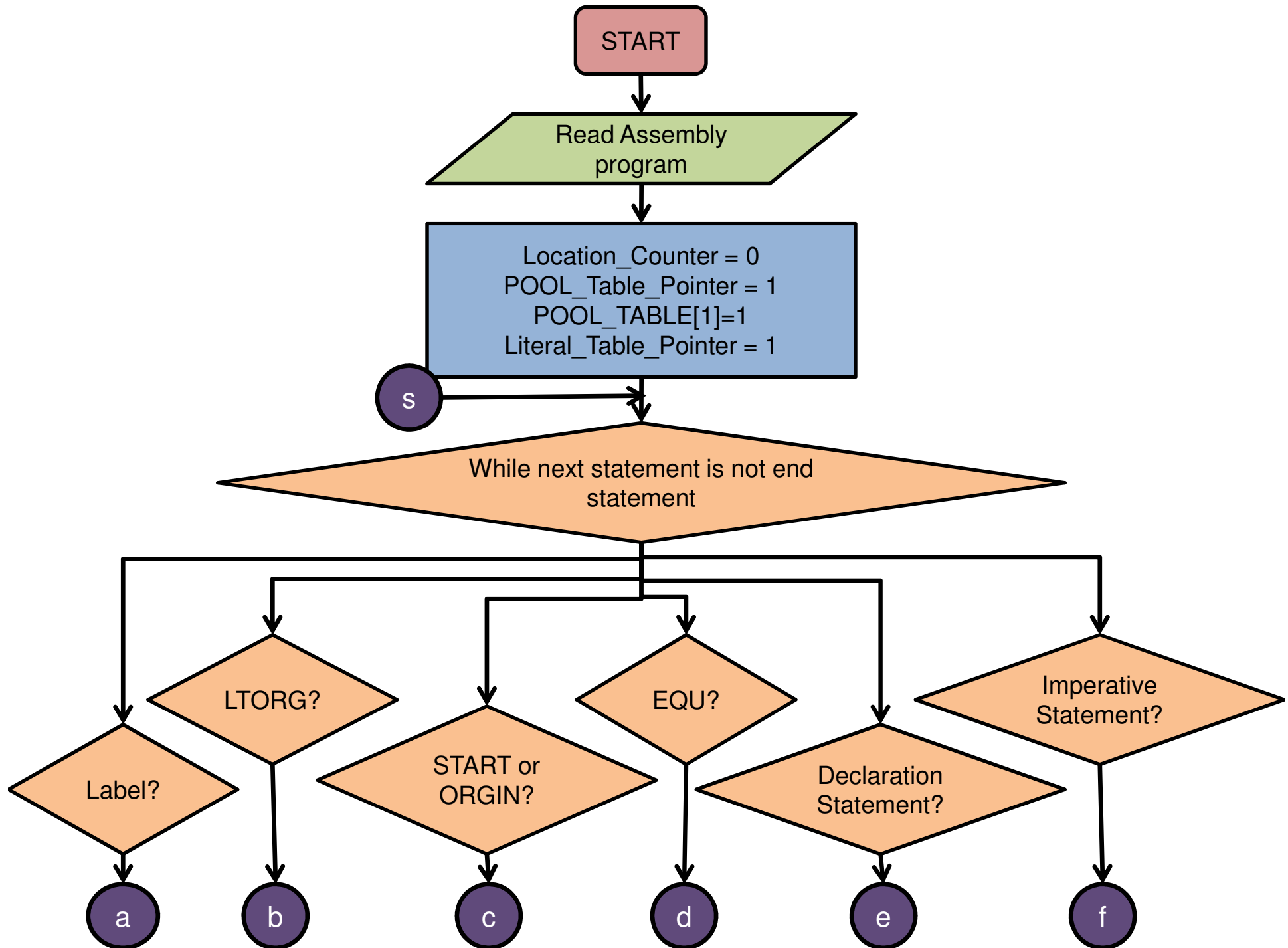
c)Go to Pass 2.

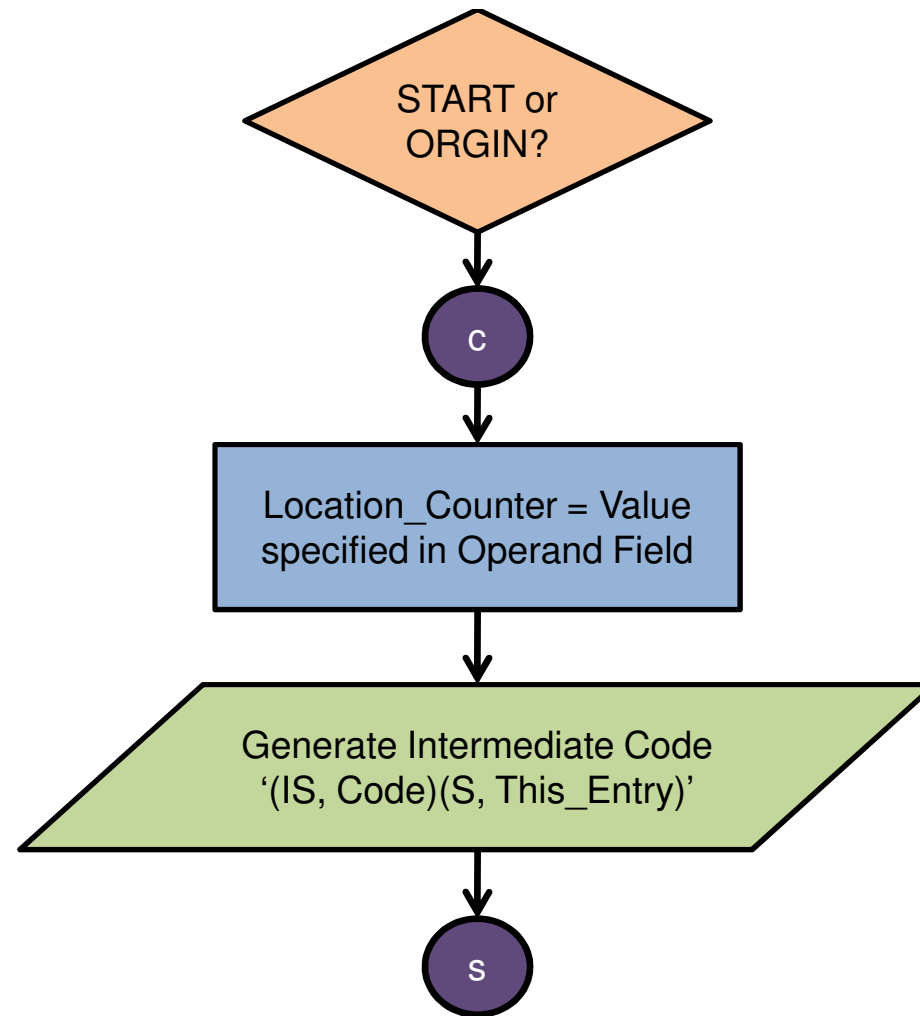


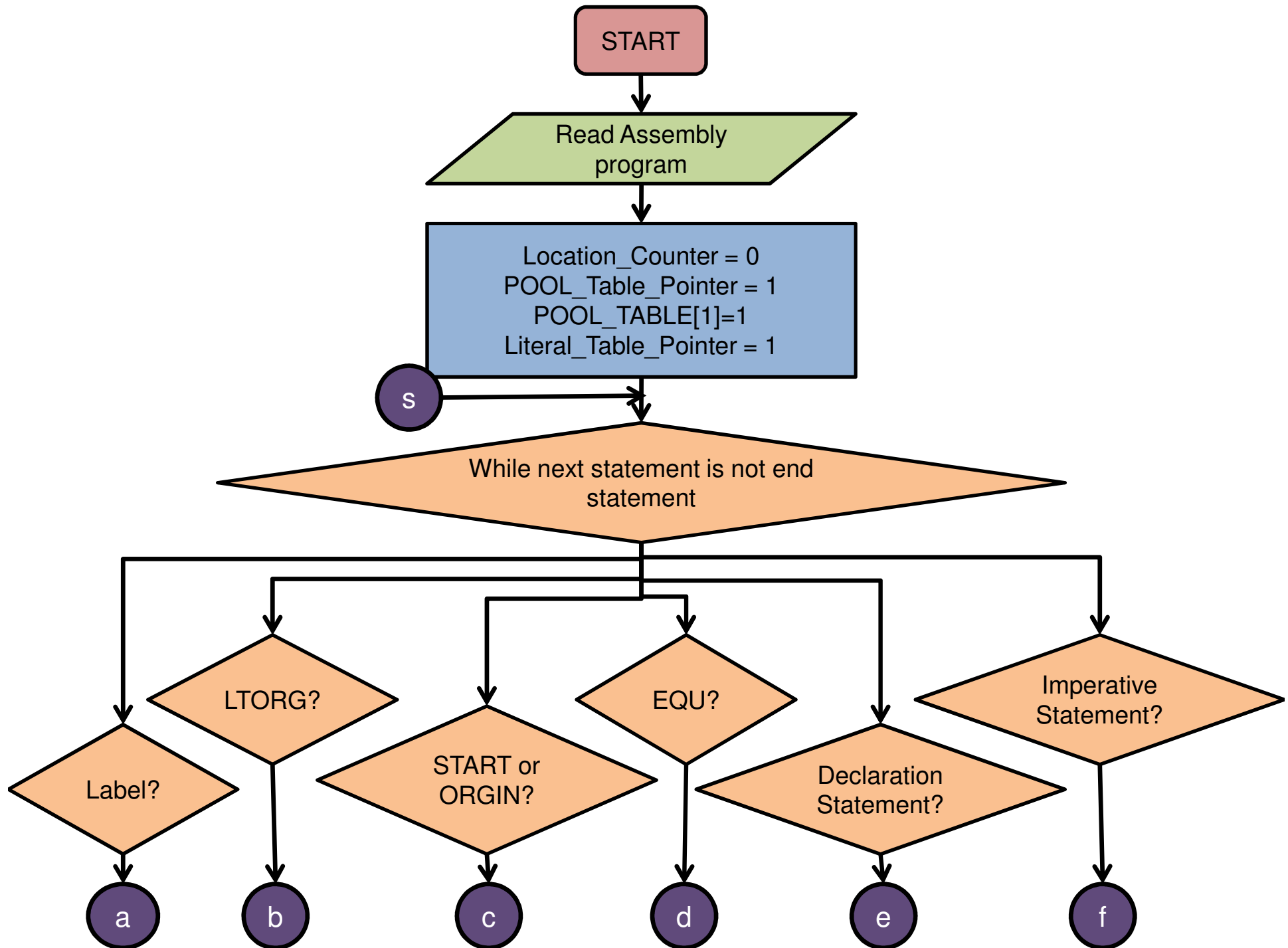


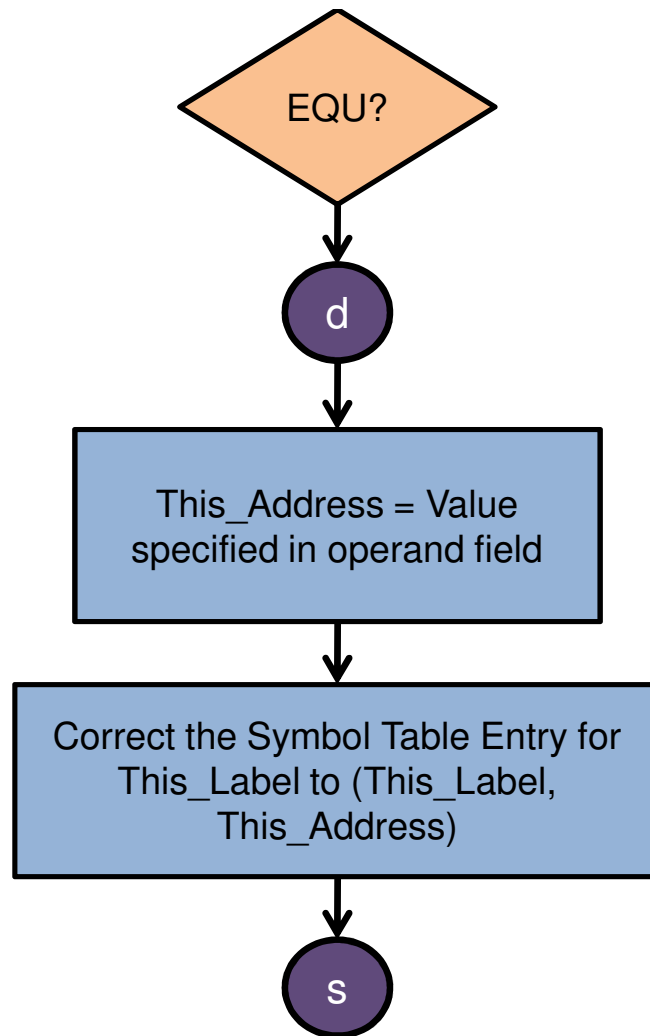


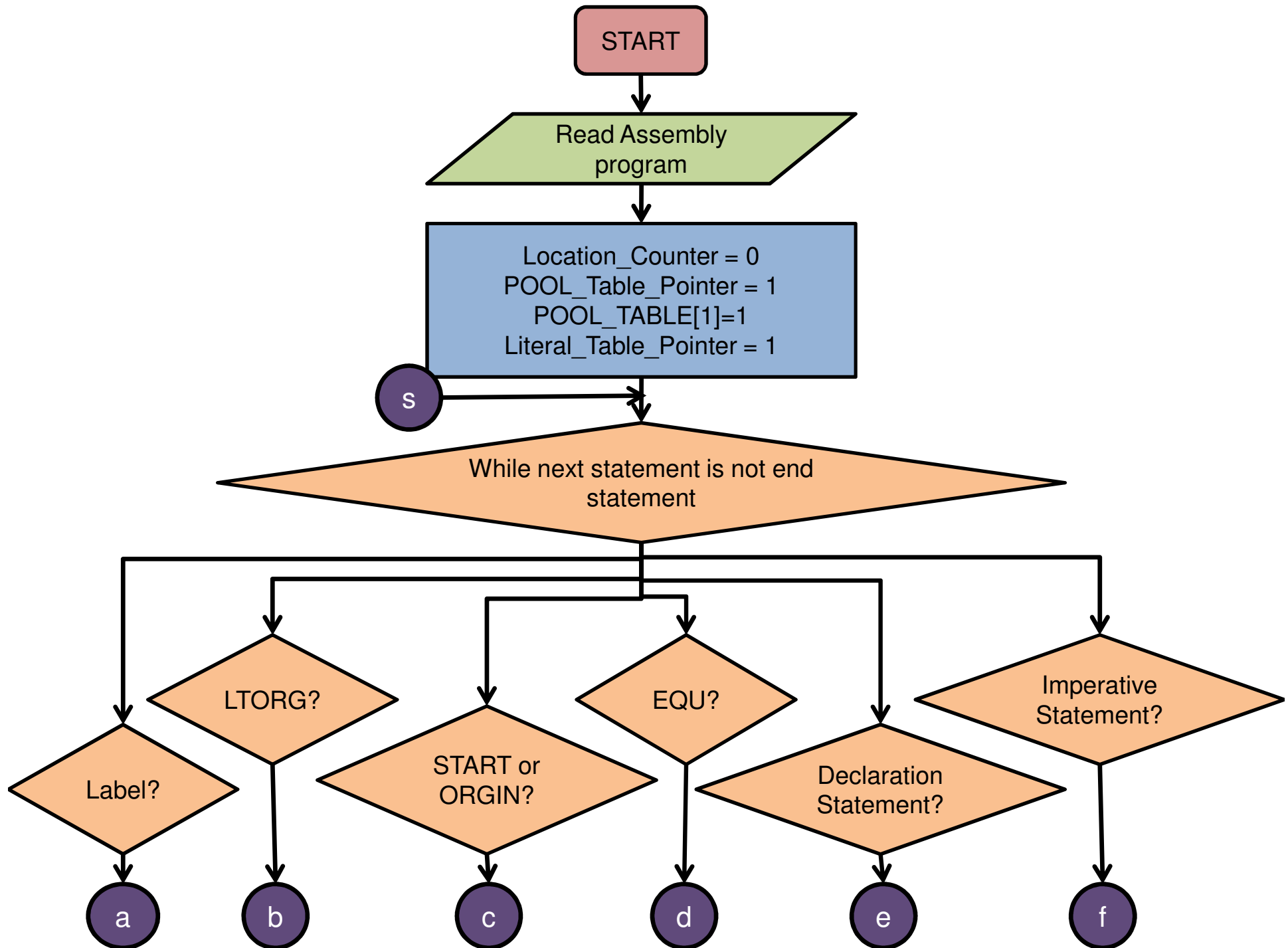


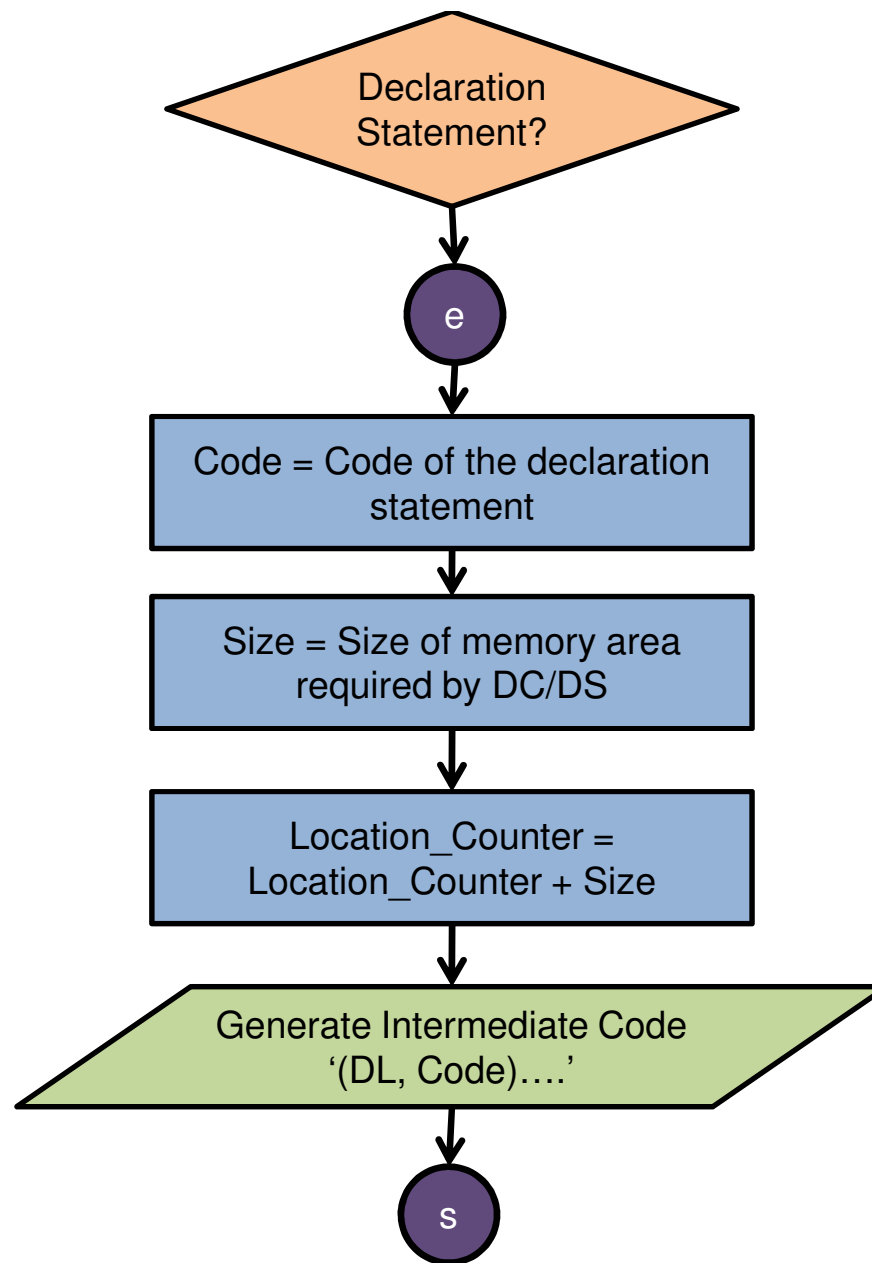


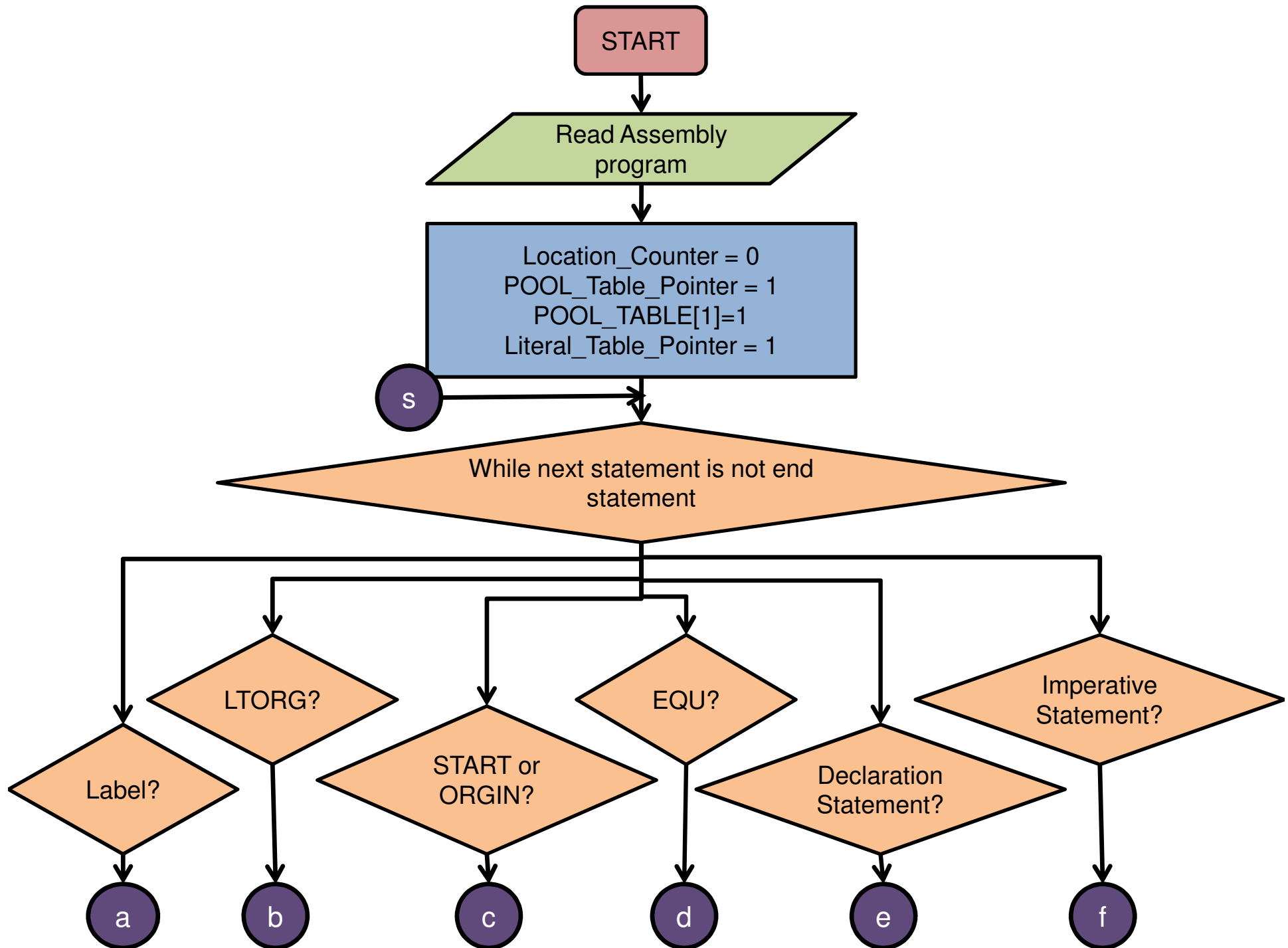


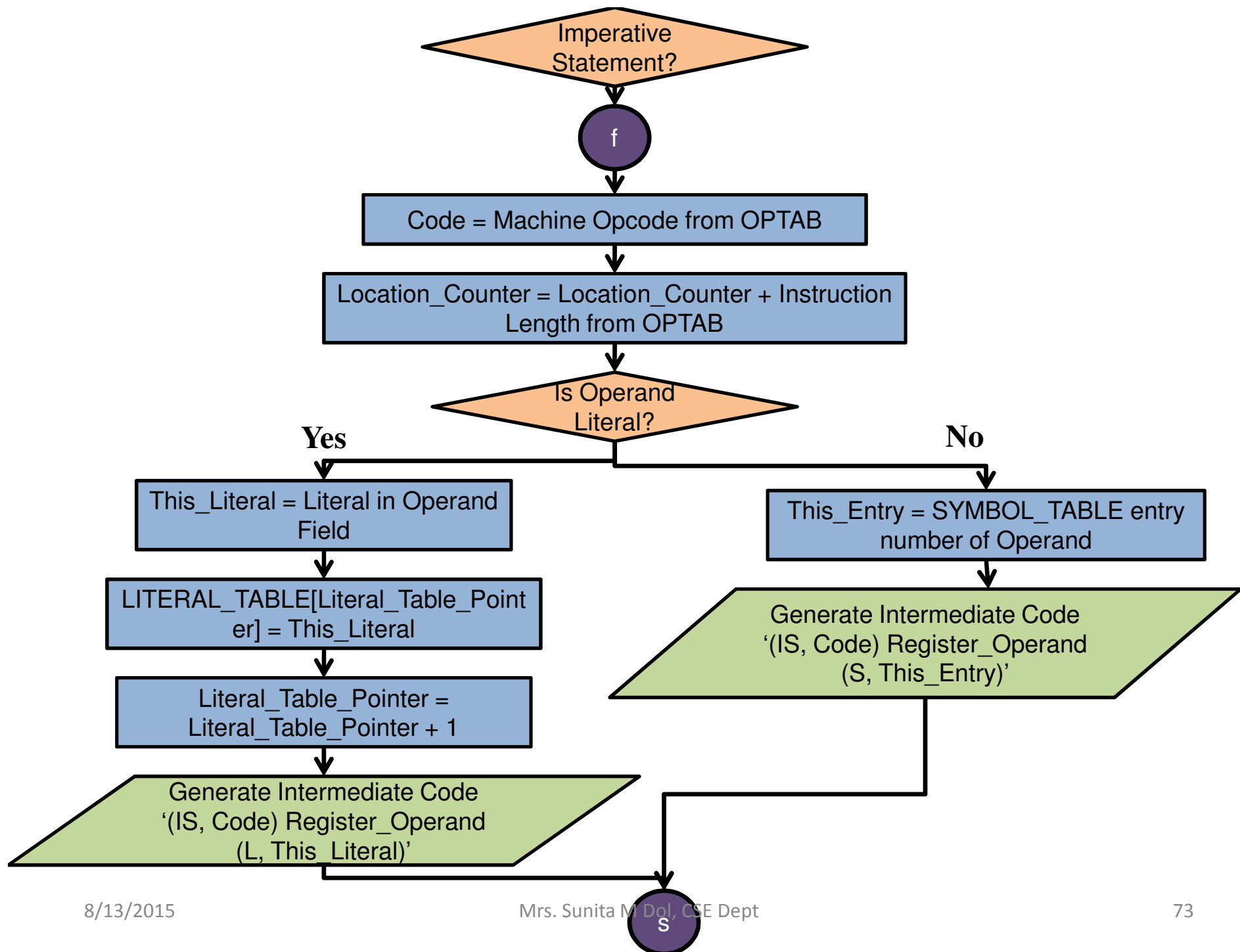


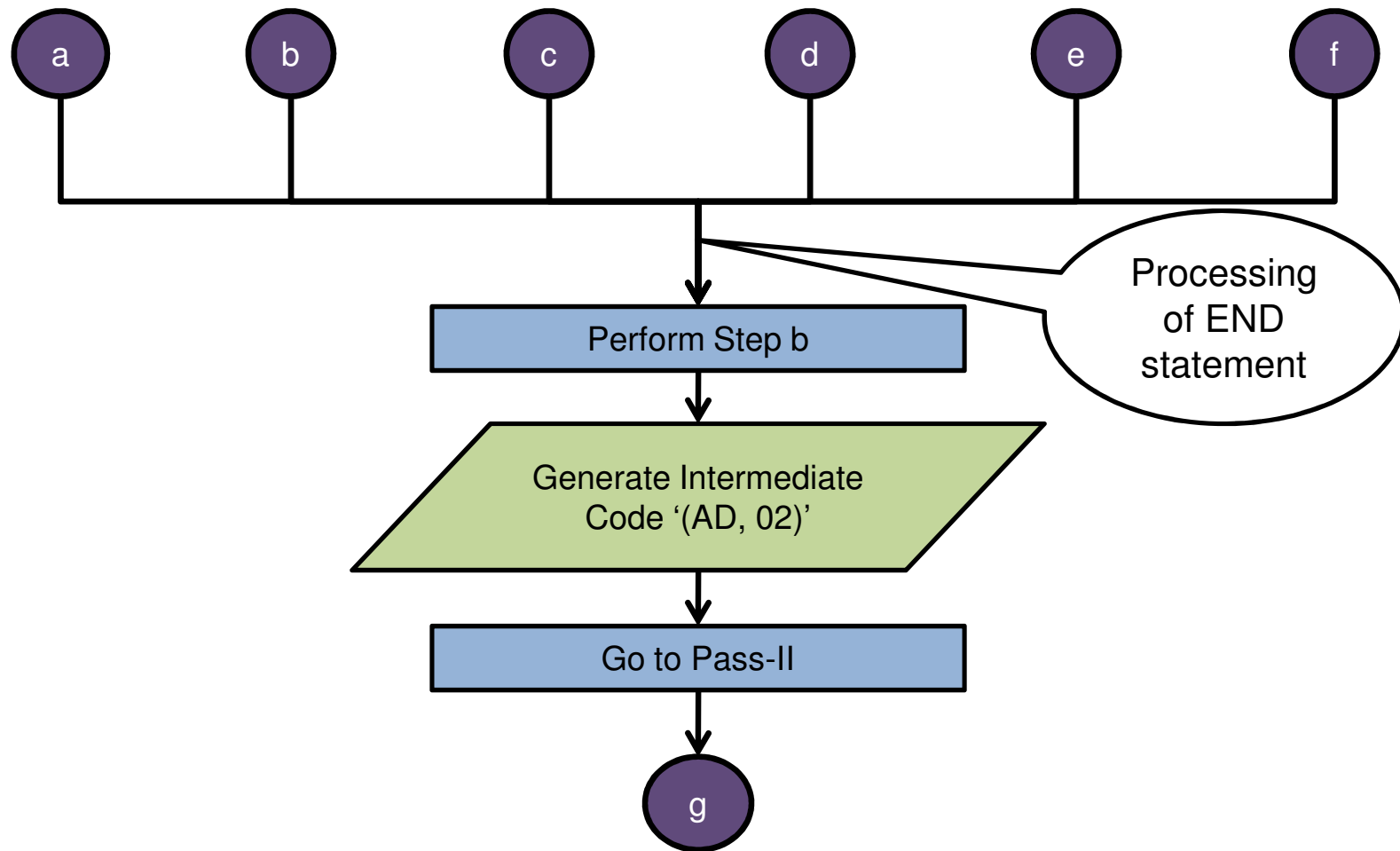












Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

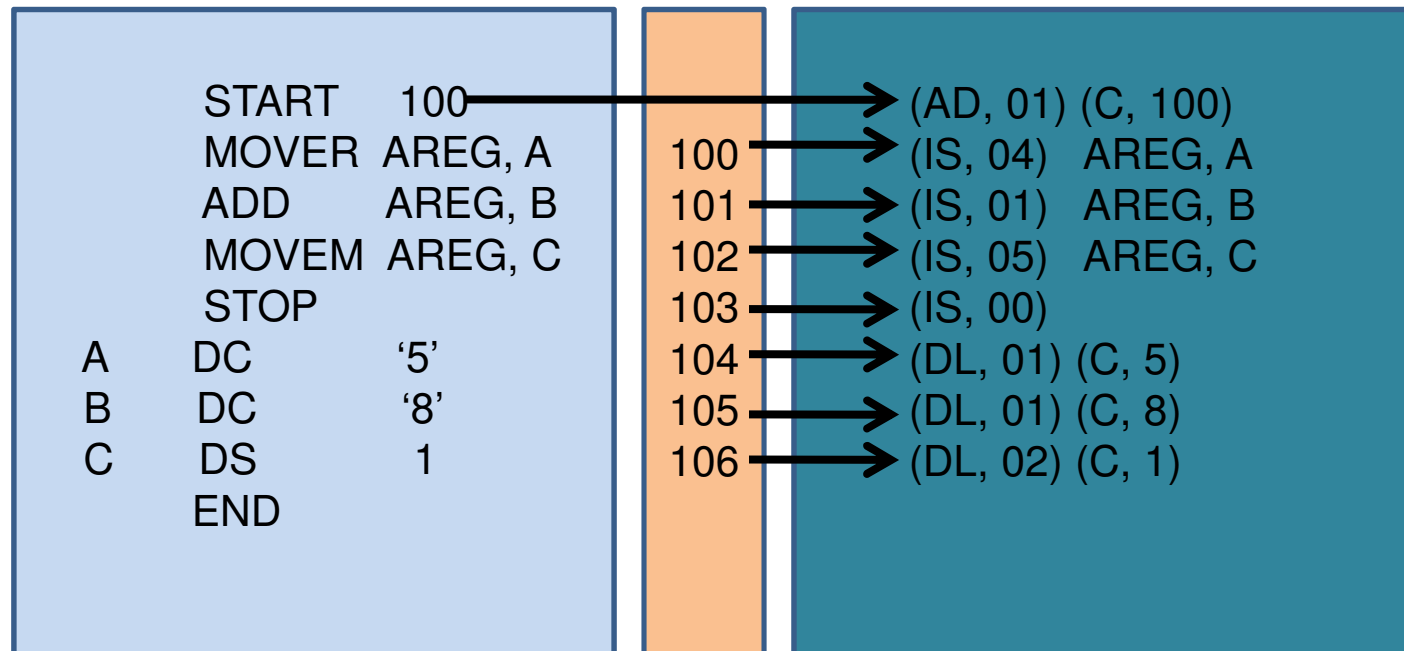
OP Table

```
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END
```

Source Program

LC

Intermediate Representation

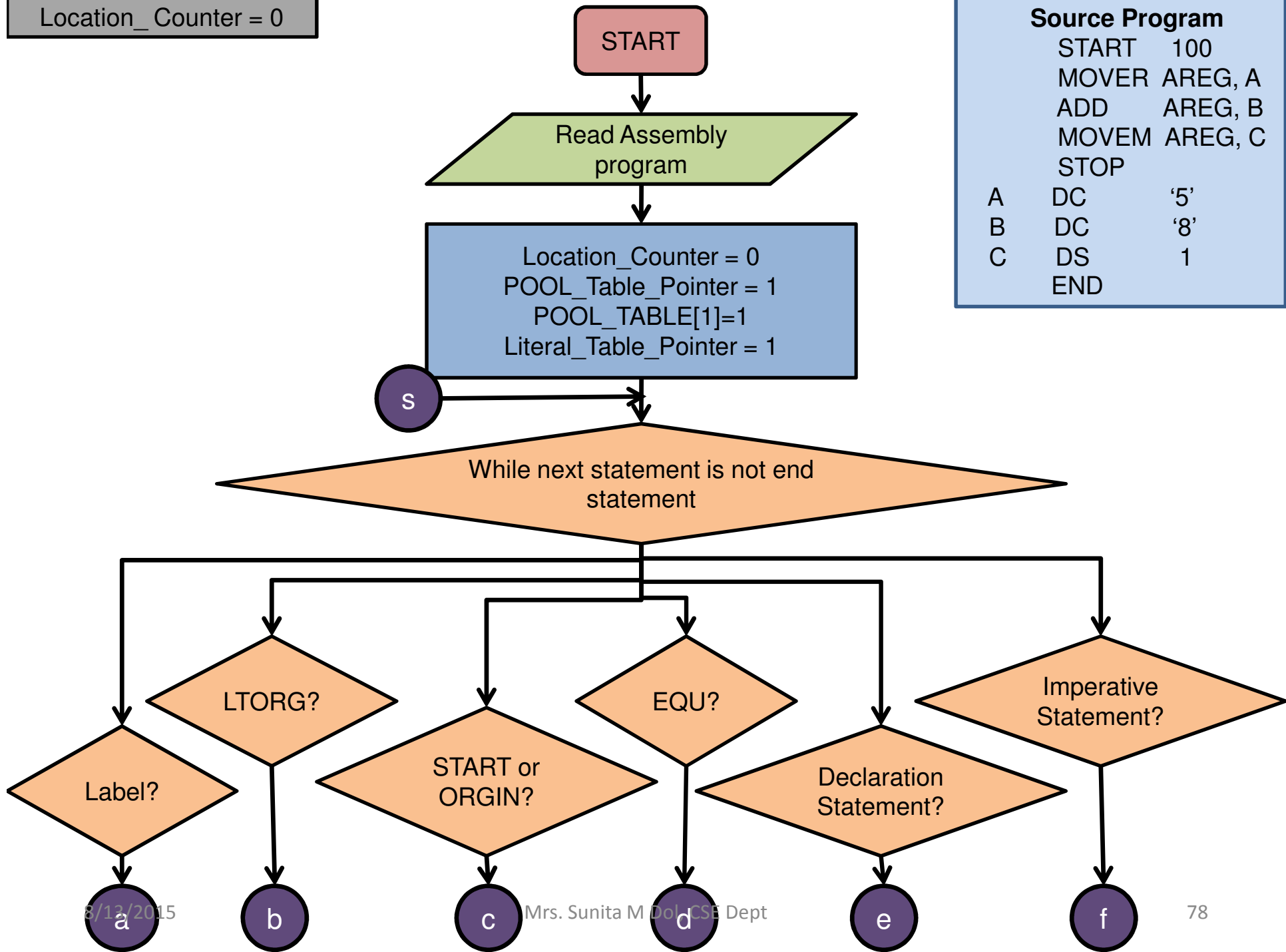


Symbol	Address	Length
A	104	1
B	105	1
C	106	1

Symbol Table

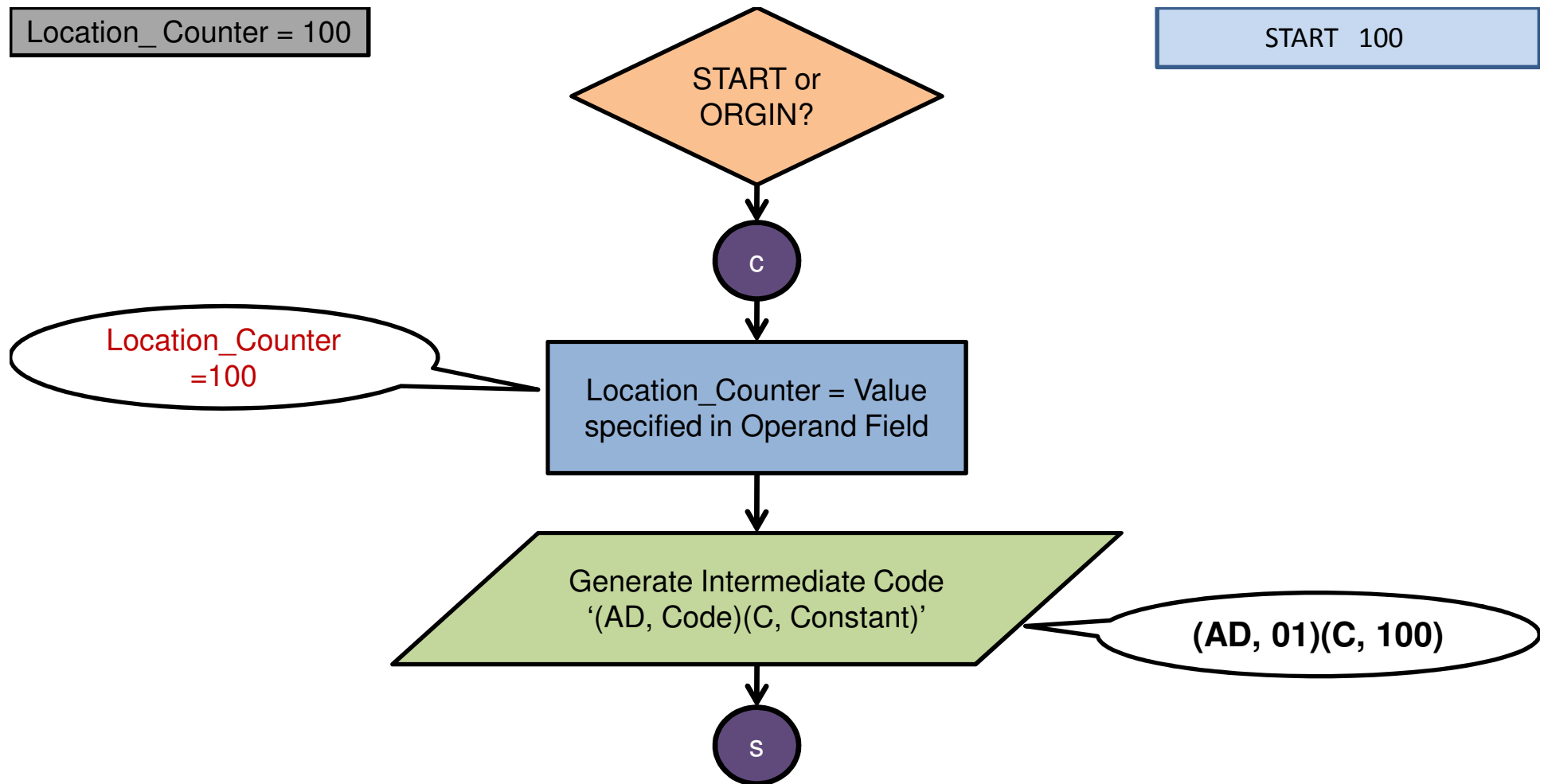
Location_Counter = 0

```
Source Program
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END
```



Location_Counter = 100

START 100



Location_Counter = 100

Intermediate Code
(AD, 01)(C, 100)

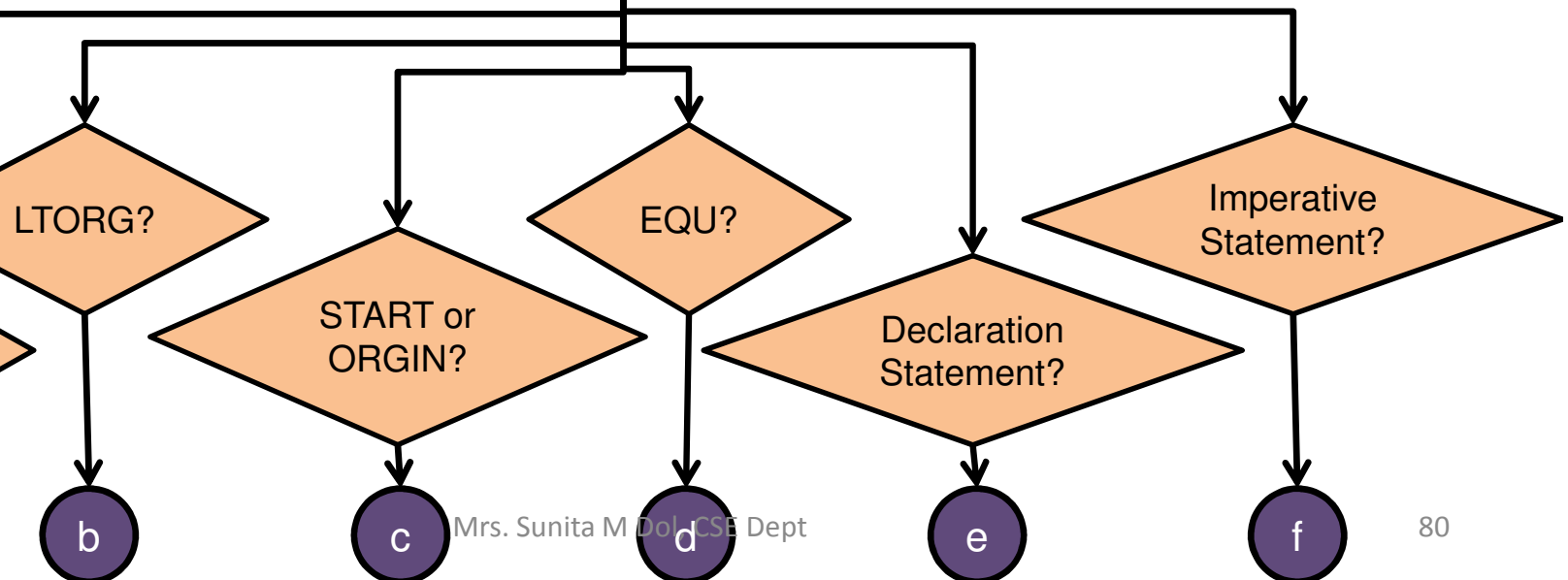
START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement



Source Program

```
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END
```


Location_Counter = 100

START 100
MOVER AREG, A

Location_Counter
= 100+1
=101

Imperative
Statement?

f

Code = Machine Opcode from
OPTAB

Code = 04

Location_Counter = Location_Counter + Instruction
Length from OPTAB

Is Operand
Literal?

No

This_Entry = SYMBOL_TABLE entry
number of Operand

Generate Intermediate Code
'(IS, Code) Register_Operand,
(S, This_Entry)'

(IS, 04) AREG, A

S

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

OP Table

Location_Counter = 101

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, A

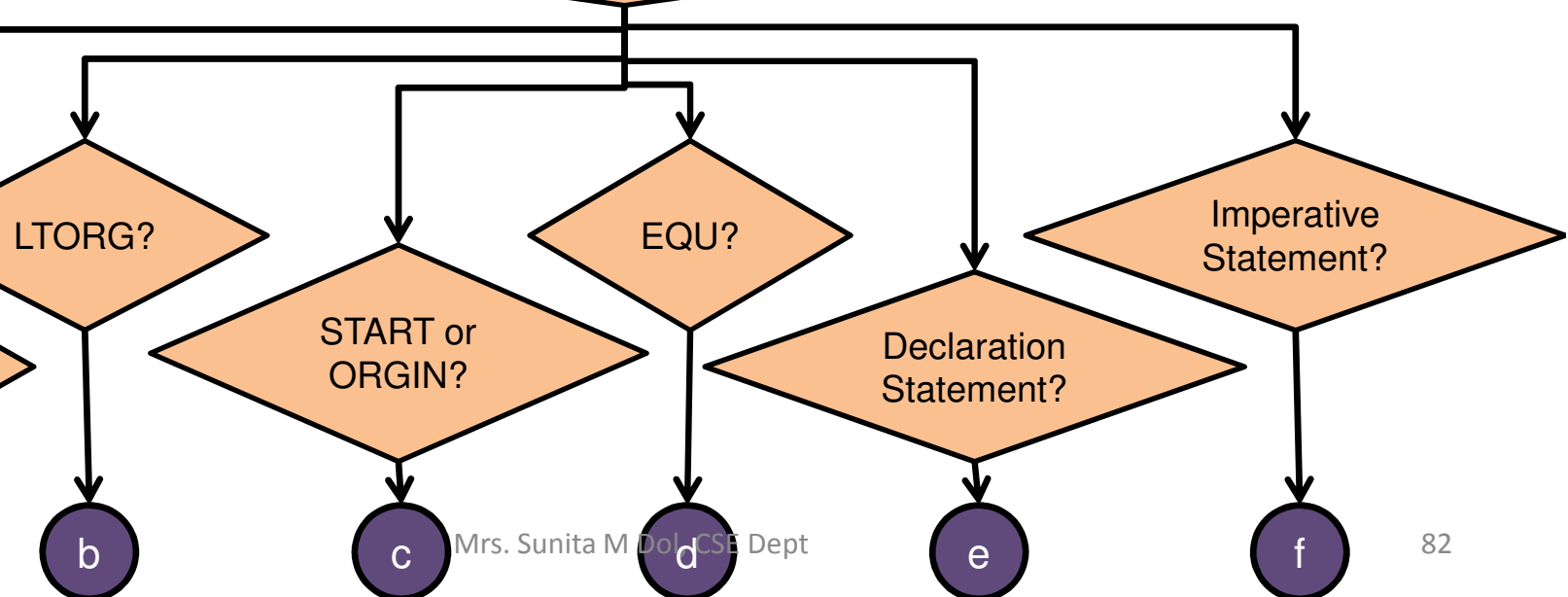
START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement



Source Program

```
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END
```

Location_Counter = 101

START 100
MOVER AREG, A
ADD AREG, B

Imperative
Statement?

f

Code = Machine Opcode from
OPTAB

Code = 01

Location_Counter
= 101+1
= 102

Location_Counter = Location_Counter + Instruction
Length from OPTAB

Is Operand
Literal?

No

This_Entry = SYMBOL_TABLE entry
number of Operand

Generate Intermediate Code
'(IS, Code) Register_Operand,
(S, This_Entry)'

(IS, 01) AREG, B

S

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

OP Table

Mrs. Sunita M Dol, CSE Dept

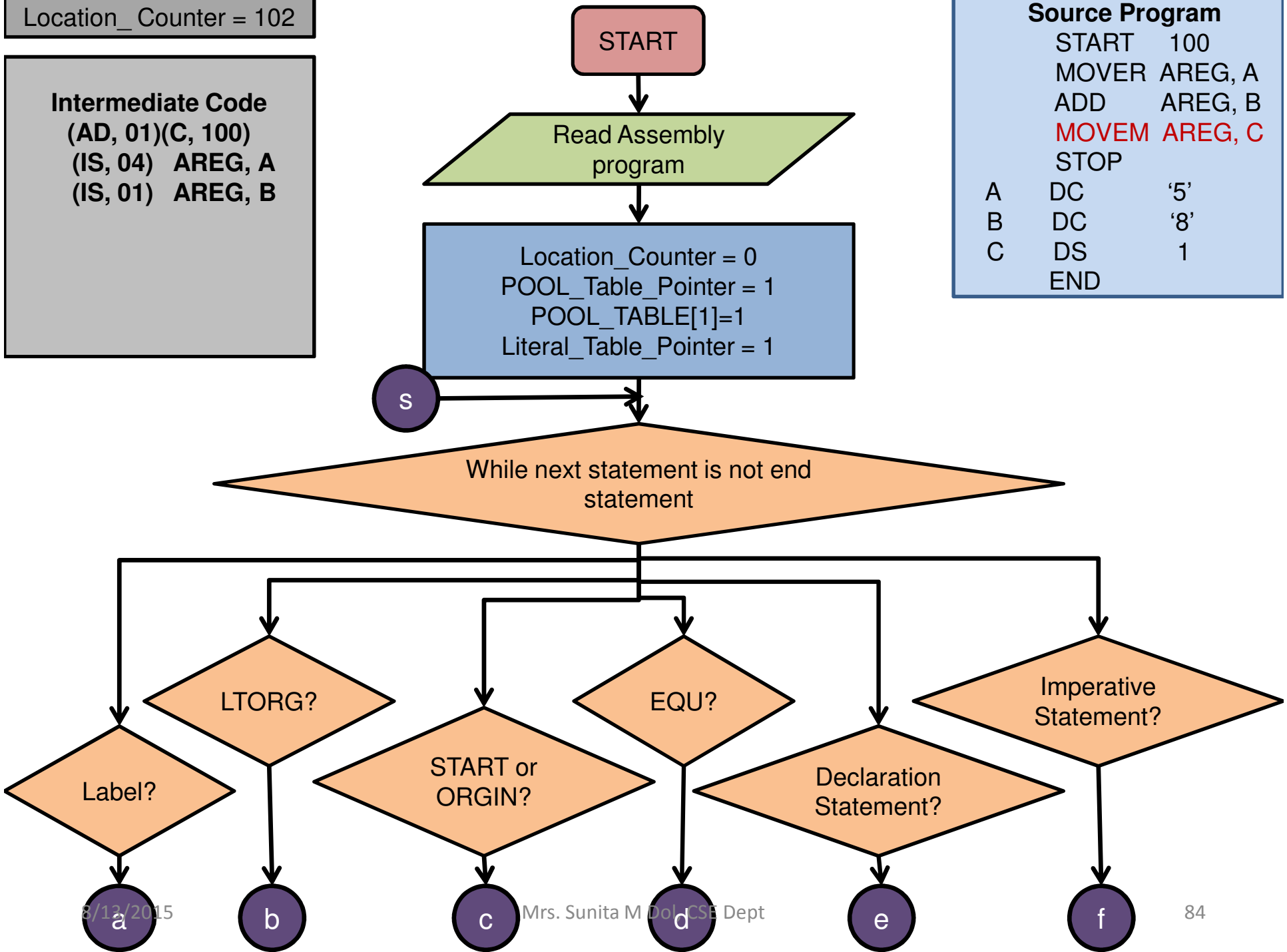
83

Location_Counter = 102

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B

Source Program
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP

A DC '5'
B DC '8'
C DS 1
END



Location_Counter = 102

```
START 100
MOVER AREG, A
ADD   AREG, B
MOVEM AREG, C
```

Location_Counter
= 102+1
=103

Imperative
Statement?

f

Code = Machine Opcode from
OPTAB

Code = 05

Location_Counter = Location_Counter + Instruction
Length from OPTAB

Is Operand
Literal?

No

This_Entry = SYMBOL_TABLE entry
number of Operand

Generate Intermediate Code
'(IS, Code) Register_Operand,
(S, This_Entry)'

(IS, 05) AREG, C

S

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

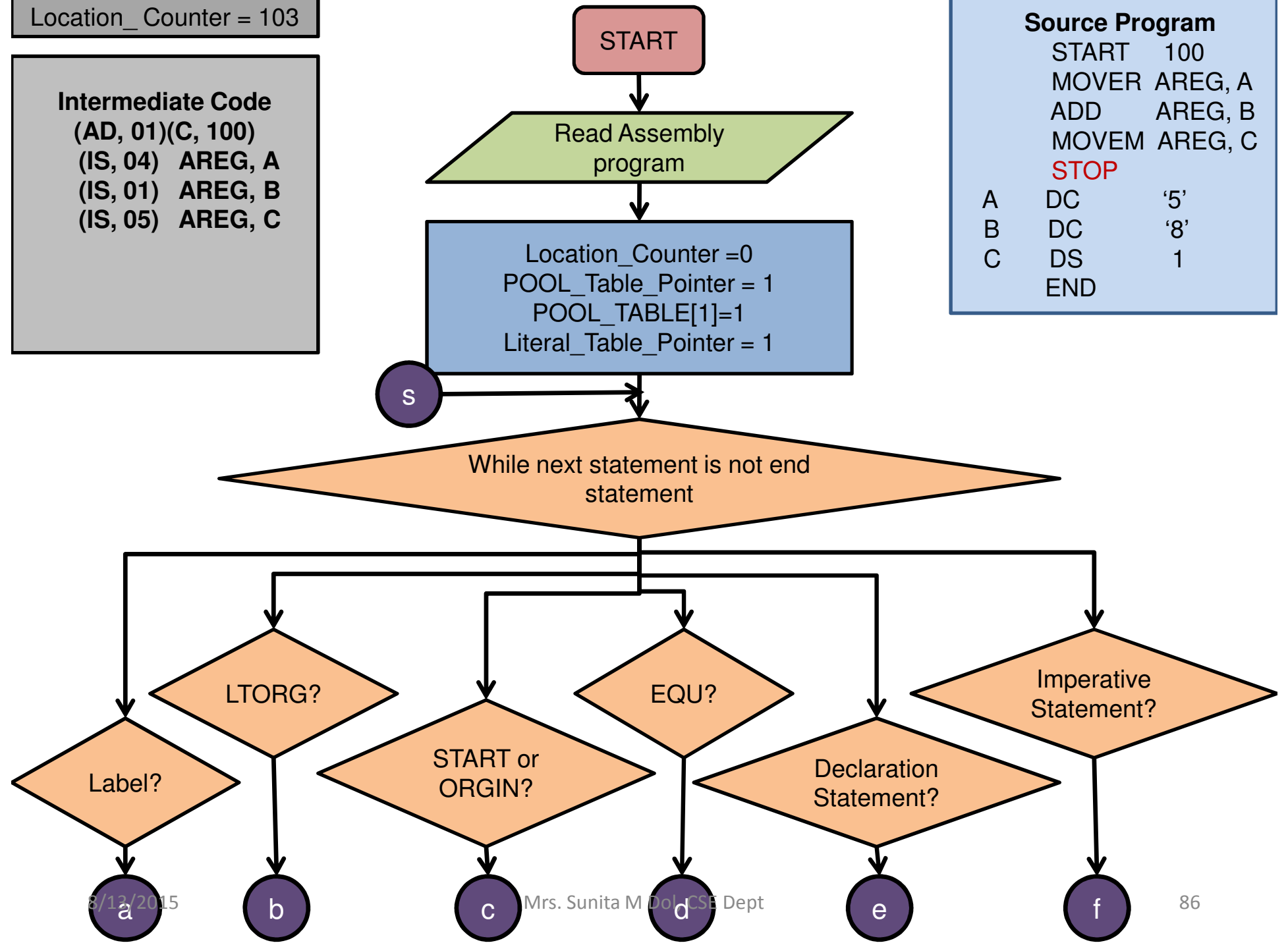
8/13/2015

OP Table

Location_Counter = 103

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C

Source Program
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END



Location_Counter = 103

```
START 100
MOVER AREG, A
ADD   AREG, B
MOVEM AREG, C
STOP
```

Location_Counter
= 103+1
=104

Imperative
Statement?

f

Code = Machine Opcode from
OPTAB

Location_Counter = Location_Counter + Instruction
Length from OPTAB

Code = 01

Is Operand
Literal?

No

This_Entry = SYMBOL_TABLE entry
number of Operand

Generate Intermediate Code
'(IS, Code) Register_Operand,
(S, This_Entry)'

(IS, 00)

S

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

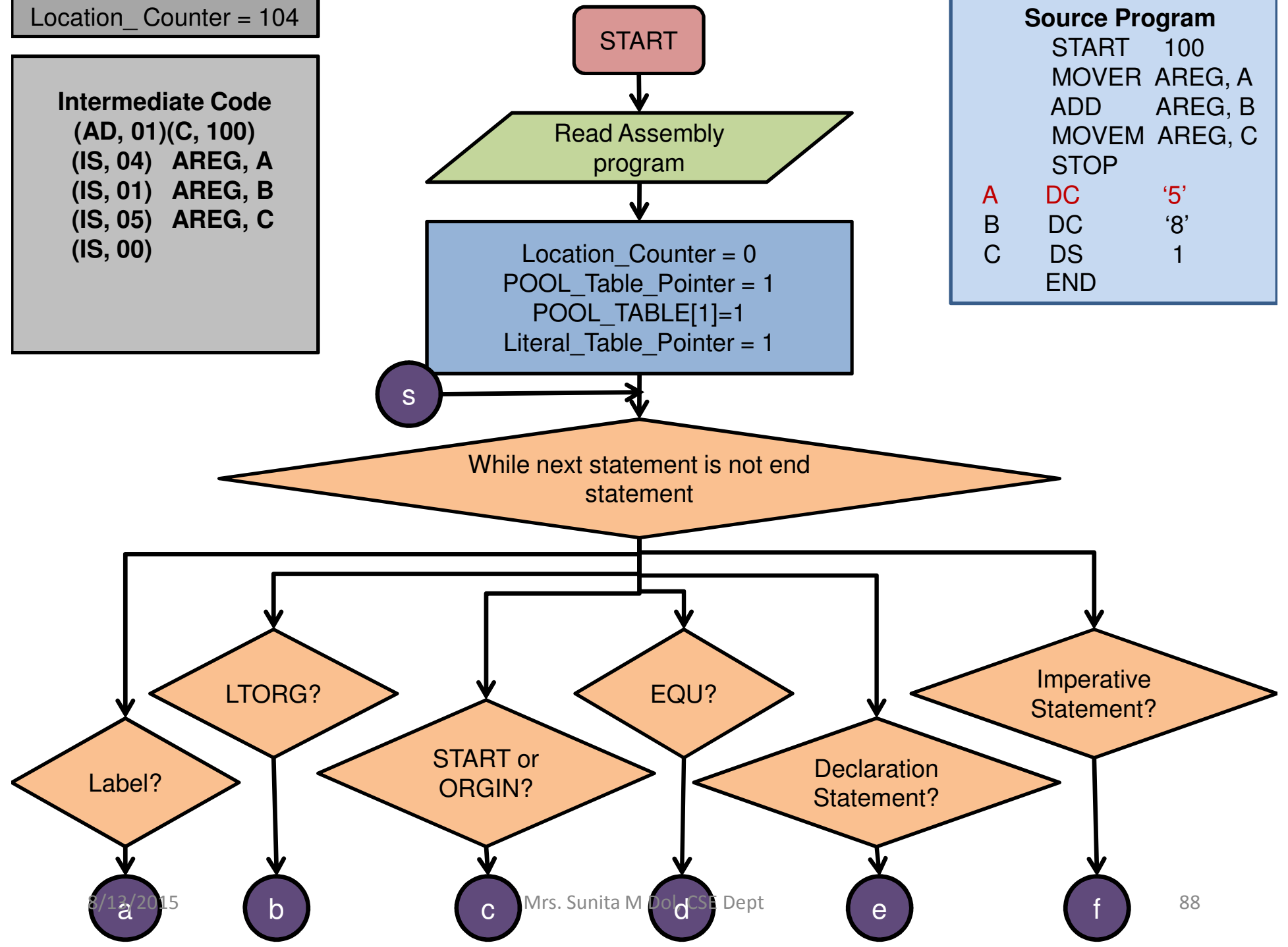
OP Table

Location_Counter = 104

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)

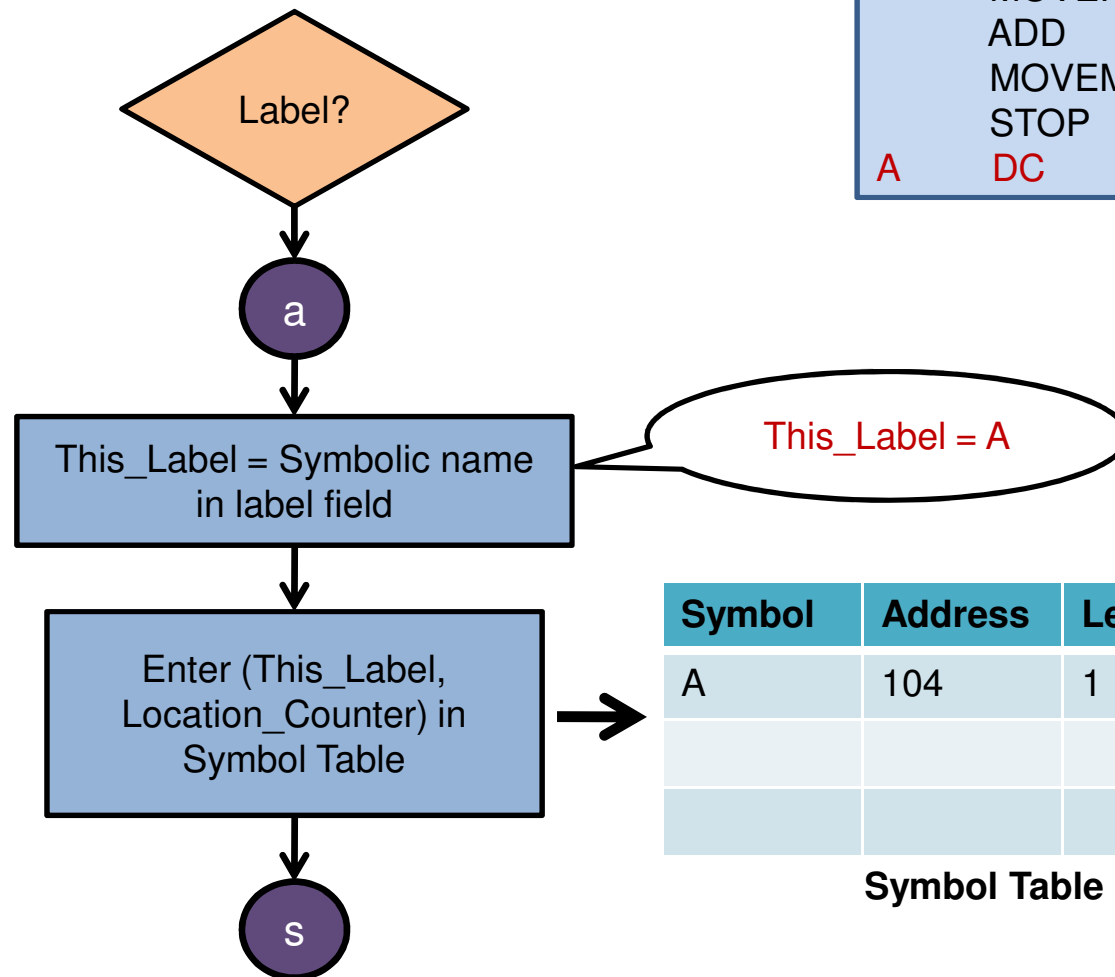
Source Program
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP

A DC '5'
B DC '8'
C DS 1
END



Location_Counter = 104

```
START 100  
MOVER AREG, A  
ADD    AREG, B  
MOVEM AREG, C  
STOP  
A      DC    '5'
```



Location_Counter = 104

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)

START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement

LTORG?

Label?

a

b

START or
ORIGIN?

c

EQU?

Declaration
Statement?

d

e

Imperative
Statement?

f

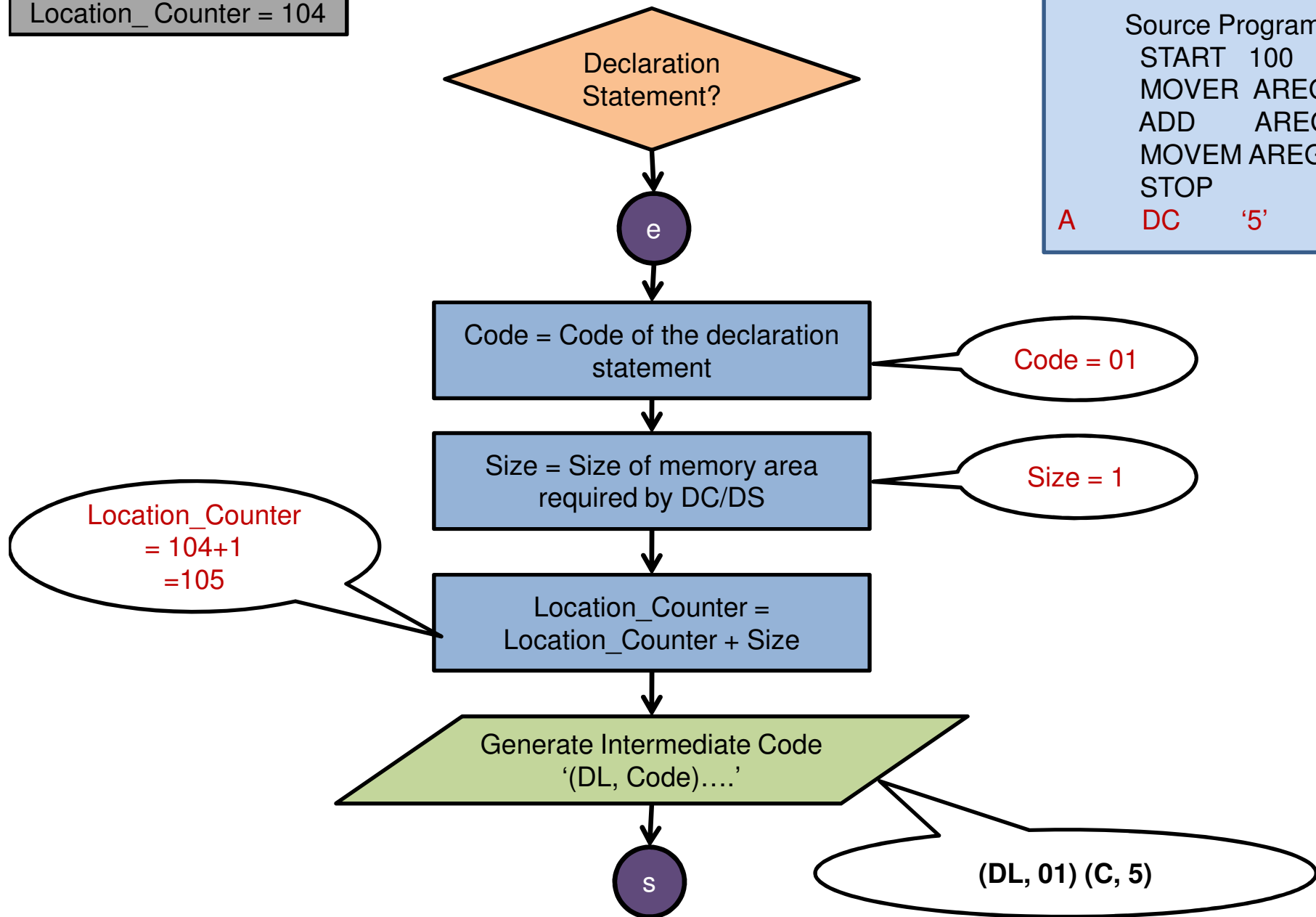
Source Program

```
START 100  
MOVER AREG, A  
ADD AREG, B  
MOVEM AREG, C  
STOP
```

A	DC	'5'
B	DC	'8'
C	DS	1
END		

Location_Counter = 104

```
Source Program
START 100
MOVER AREG, A
ADD    AREG, B
MOVEM AREG, C
STOP
A      DC      '5'
```



Location_Counter = 105

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)

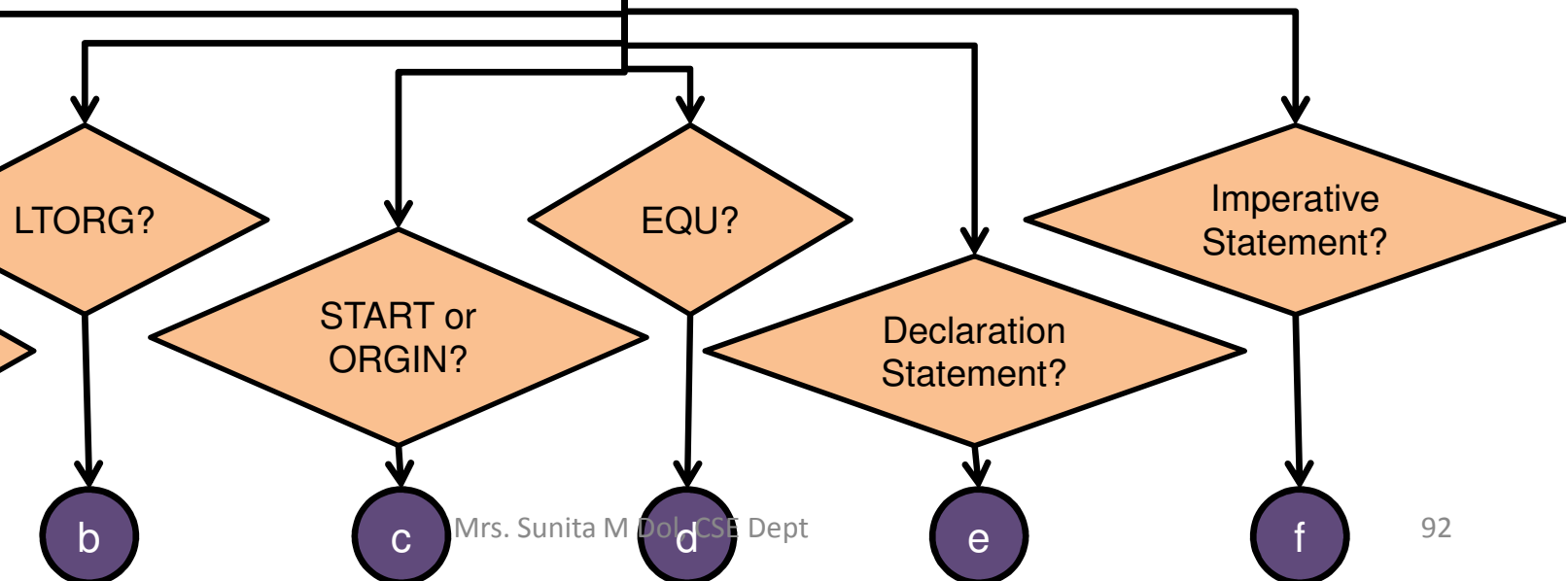
START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement

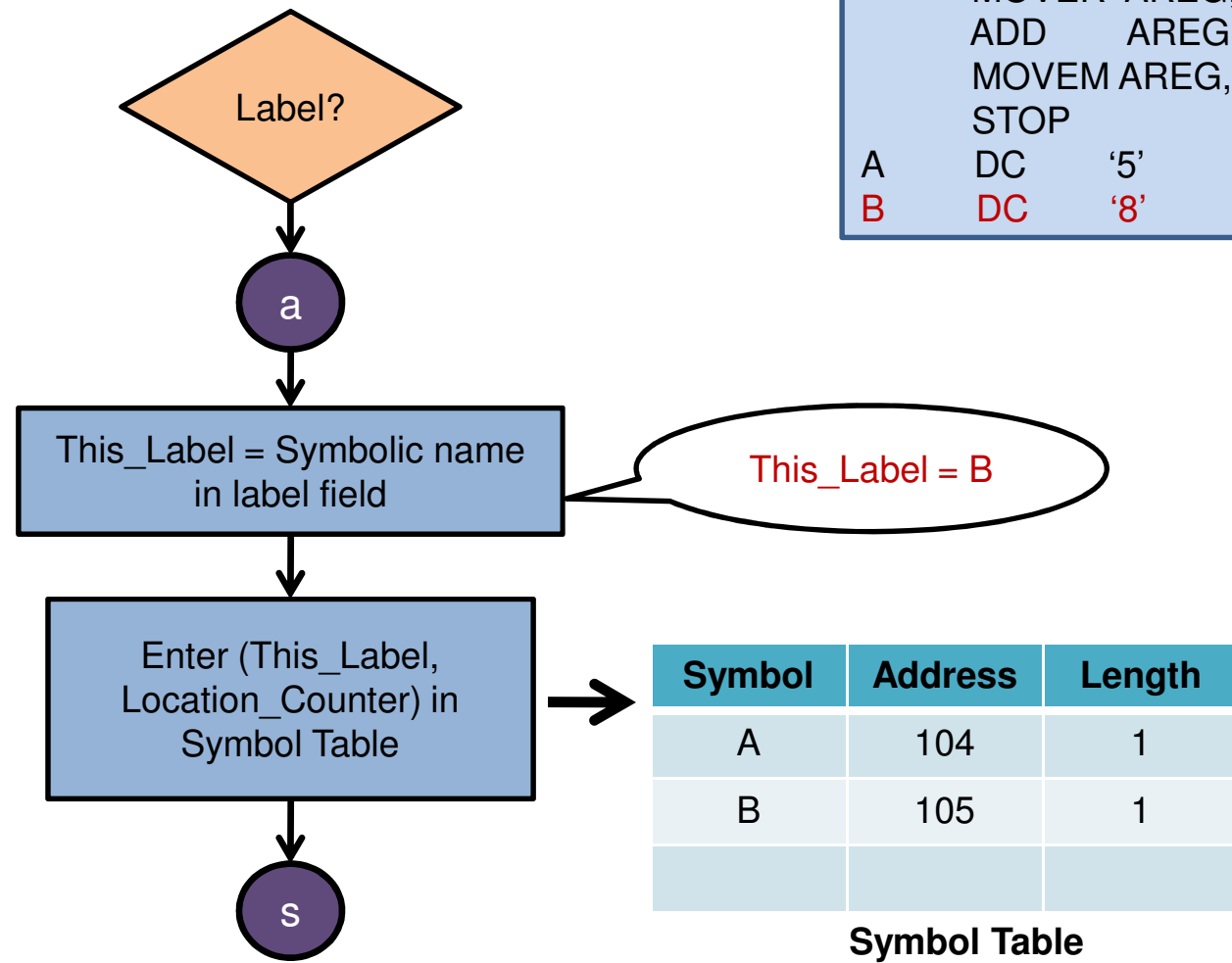


Source Program

```
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
```

A	DC	'5'
B	DC	'8'
C	DS	1
	END	

Location_Counter = 105



Location_Counter = 105

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)

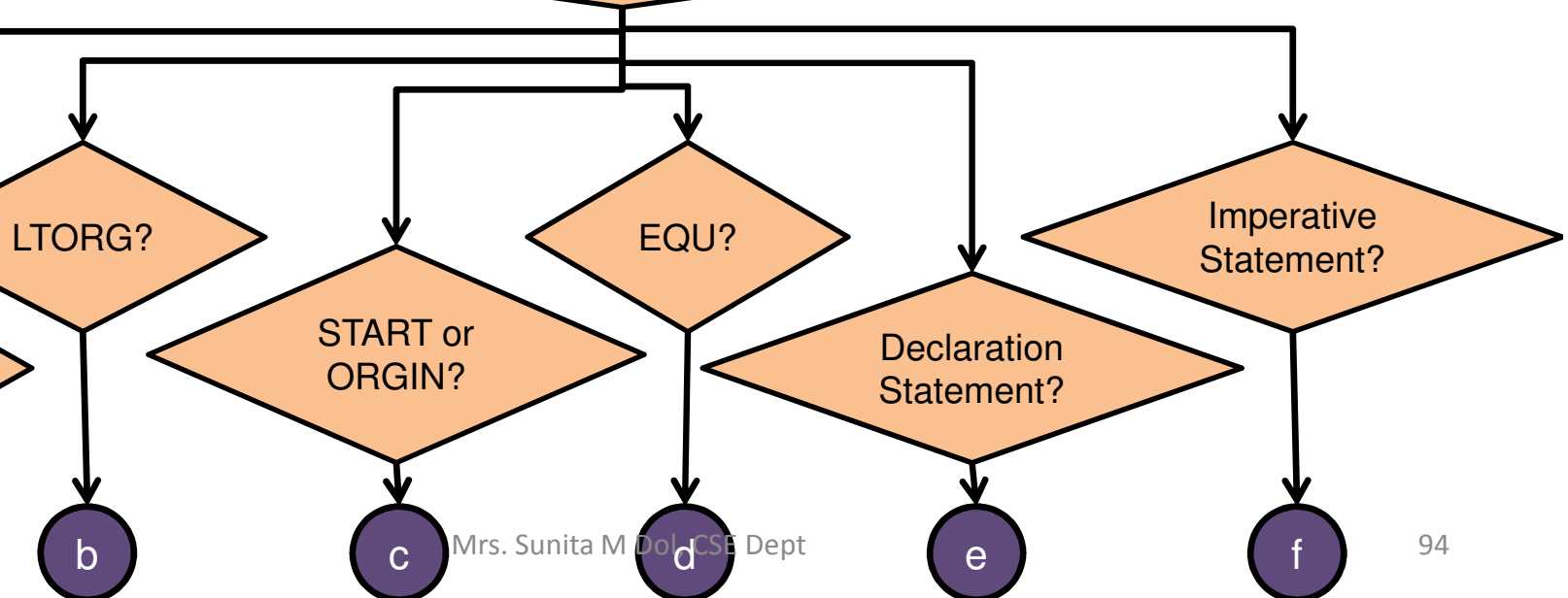
START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement

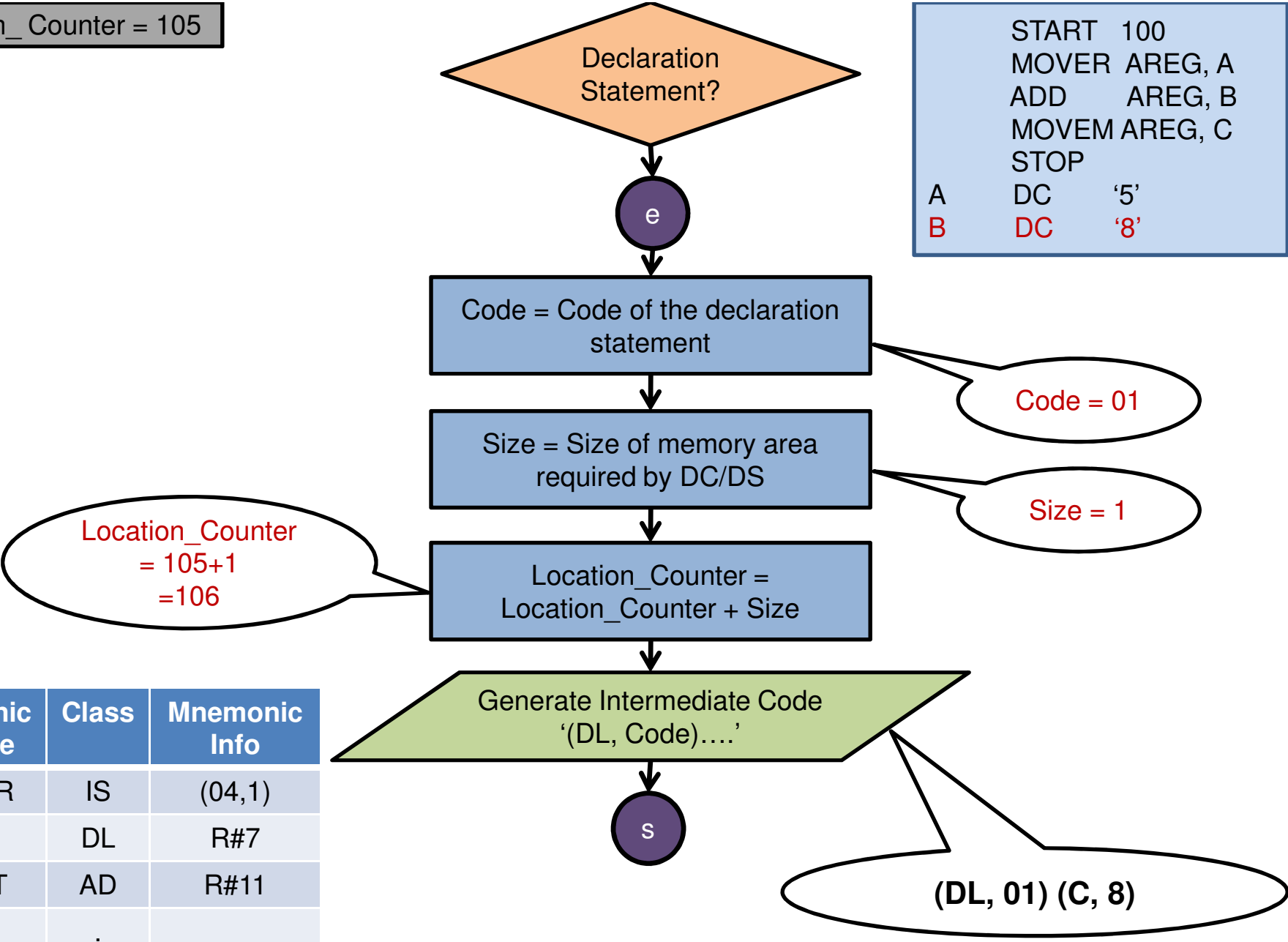


Source Program

```
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END
```

Location_Counter = 105

```
START 100
MOVER AREG, A
ADD   AREG, B
MOVEM AREG, C
STOP
A    DC '5'
B    DC '8'
```



Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015
OP Table

Location_Counter = 106

Intermediate Code

(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)

START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement

LTORG?

Label?

a

b

START or
ORIGIN?

c

EQU?

Declaration
Statement?

d

e

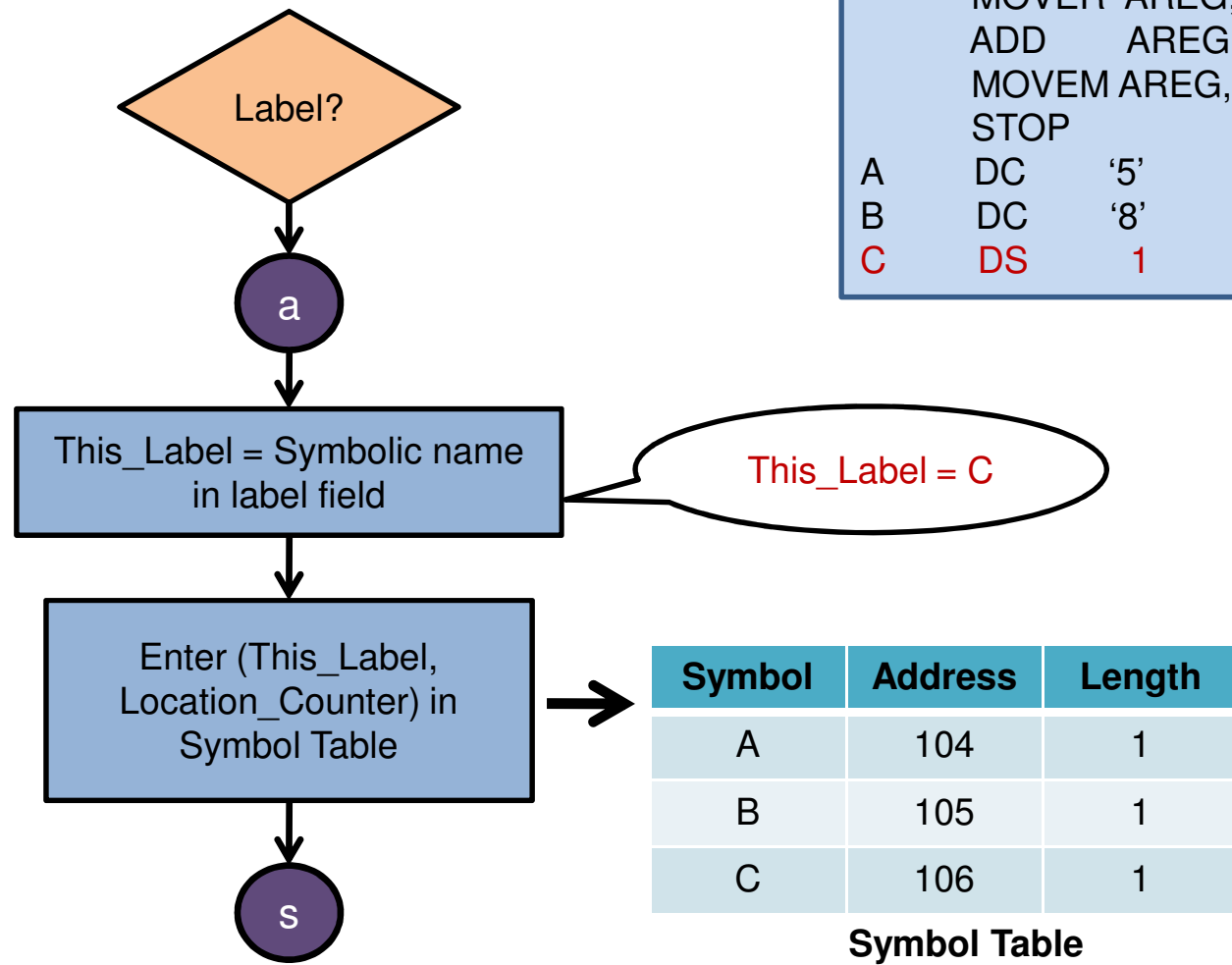
Imperative
Statement?

f

Source Program

```
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END
```


Location_Counter = 106



Location_Counter = 106

Intermediate Code

(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)

START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement

LTORG?

Label?

a

b

START or
ORIGIN?

c

EQU?

Declaration
Statement?

d

e

Imperative
Statement?

f

Source Program

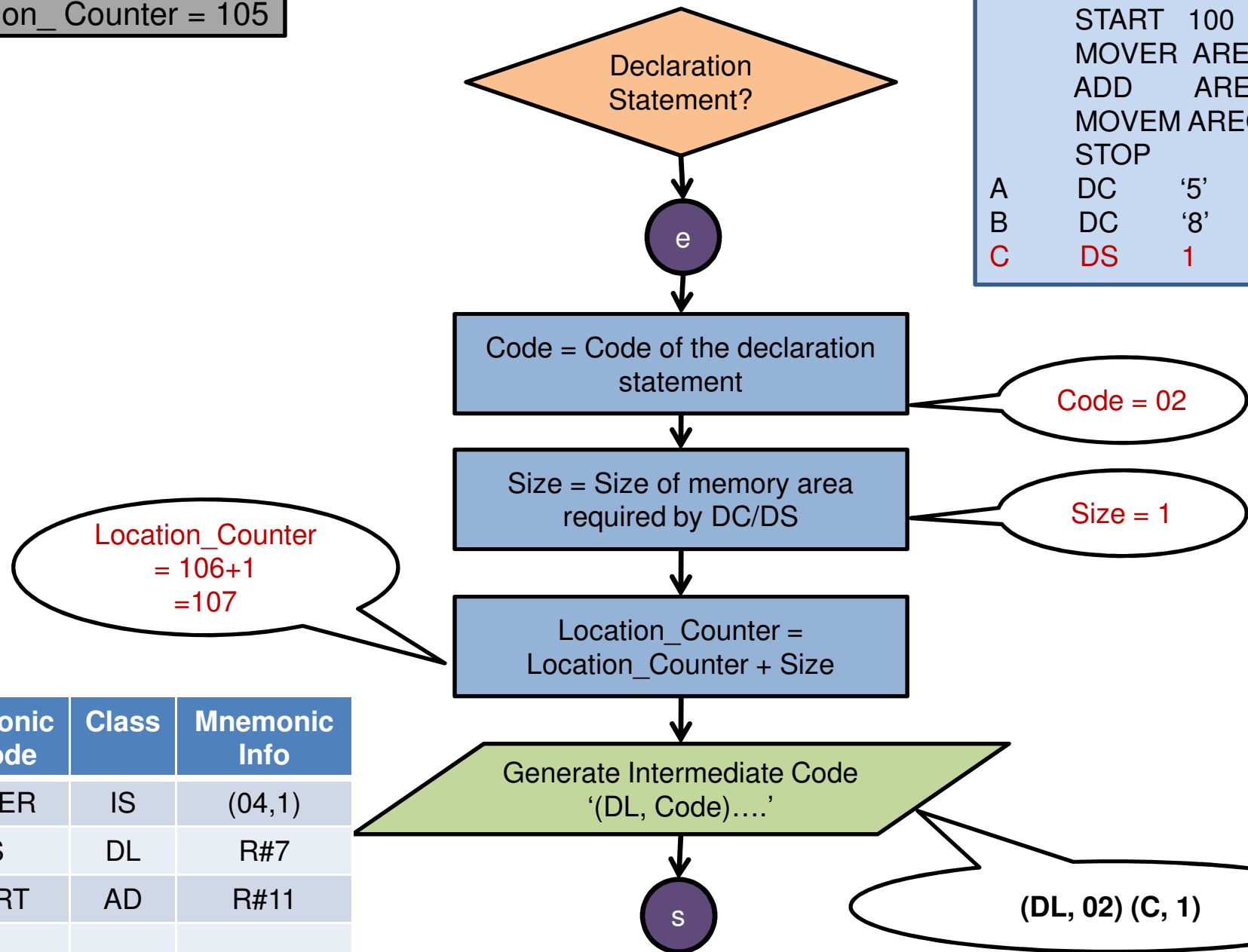
```
START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP
A DC '5'
B DC '8'
C DS 1
END
```

Location_Counter = 105

```

START 100
MOVER AREG, A
ADD   AREG, B
MOVEM AREG, C
STOP
A    DC    '5'
B    DC    '8'
C    DS    1

```



Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

OP Table

Mrs. Sunita M Dol, CSE Dept

99

Location_Counter = 107

Intermediate Code

(AD, 01)(C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement

LTORG?

Label?

a

b

START or
ORIGIN?

c

EQU?

Mrs. Sunita M Dolu CSE Dept

d

Declaration
Statement?

e

Imperative
Statement?

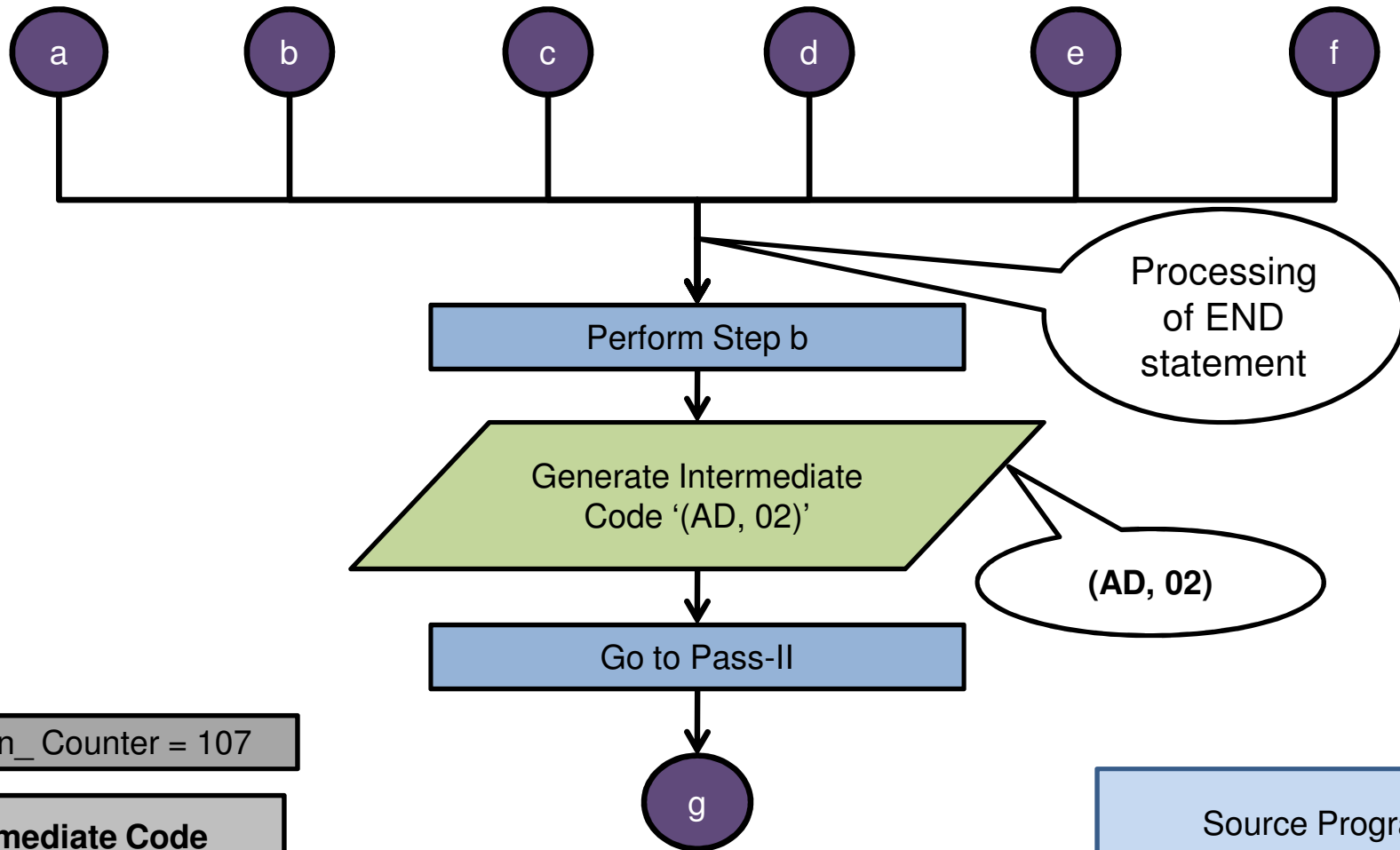
f

Source Program

START 100
MOVER AREG, A
ADD AREG, B
MOVEM AREG, C
STOP

A DC '5'
B DC '8'
C DS 1

END



Location_Counter = 107

Intermediate Code
 (AD, 01)(C, 100)
 (IS, 04) AREG, A
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 5)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

8/13/2015

Mrs. Sunita M Dol, CSE Dept

Source Program
 START 100
 MOVER AREG, A
 ADD AREG, B
 MOVEM AREG, C
 STOP
 A DC '5'
 B DC '8'
 C DS 1
END 101

Source Program

```
START 100  
MOVER AREG, A  
ADD AREG, B  
MOVEM AREG, C  
STOP
```

```
A DC '5'  
B DC '8'  
C DS 1  
END
```

Intermediate Code

```
(AD, 01) (C, 100)  
(IS, 04) AREG, A  
(IS, 01) AREG, B  
(IS, 05) AREG, C  
(IS, 00)  
(DL, 01) (C, 5)  
(DL, 01) (C, 8)  
(DL, 02) (C, 1)  
(AD, 02)
```

Symbol	Address	Length
A	104	1
B	105	1
C	106	

Symbol Table

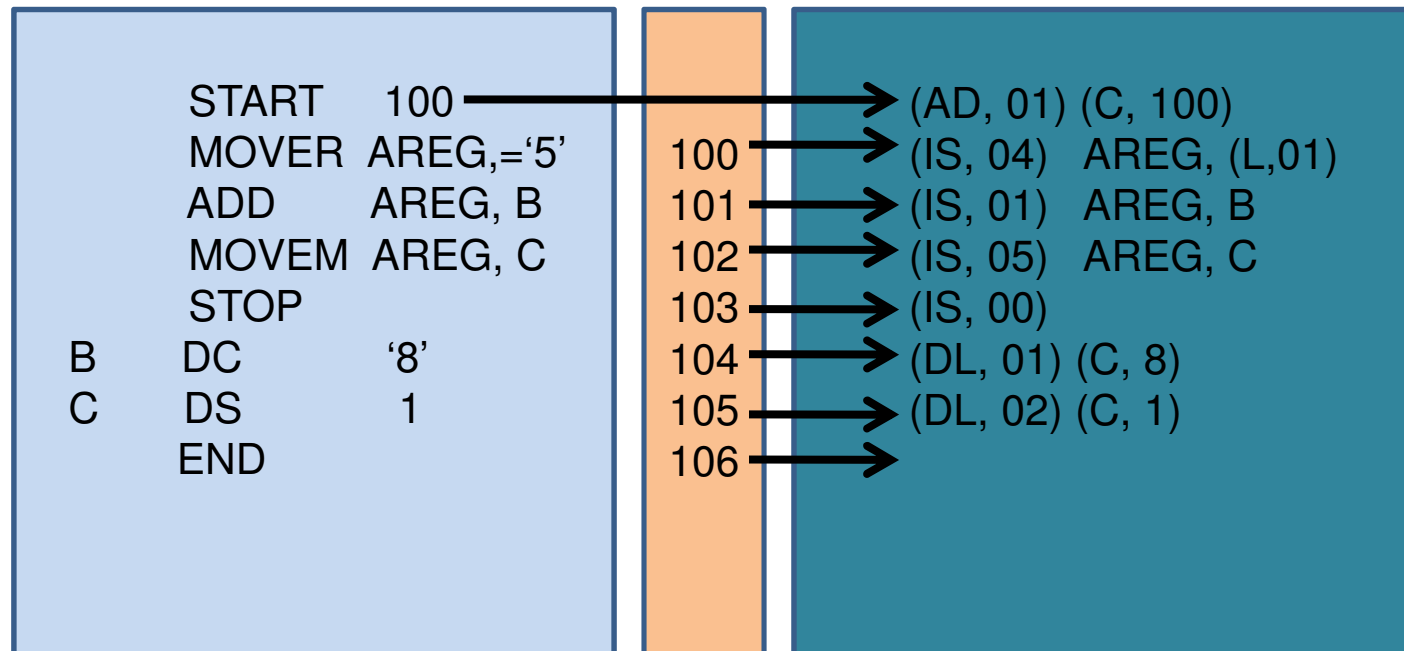
Mnemonic Opcode	Class	.Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

```
START 100
MOVER AREG, ='5'
ADD    AREG, B
MOVEM  AREG, C
STOP
B      DC      '8'
C      DS      1
END
```


Source Program

LC

Intermediate Representation



Literal_Table_P ointer	Literal	Address
1	= '5'	106
2		

Literal Table

POOL_Table_ Pointer	Literal_Table_ Pointer
1	1
2	2

POOL Table

Symbol	Address	Length
B	104	1
C	105	1

Symbol Table

Location_Counter = 0

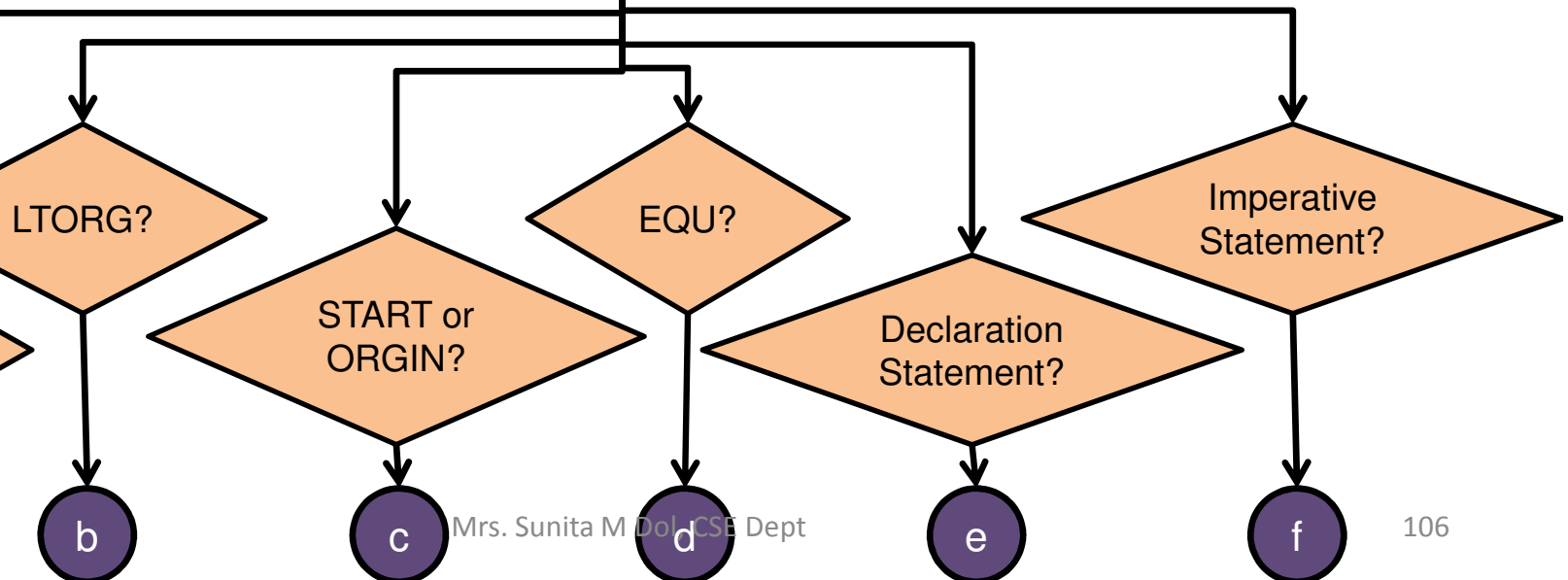
START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement



Source Program

START 100

MOVER AREG, ='5'

ADD AREG, B

MOVEM AREG, C

STOP

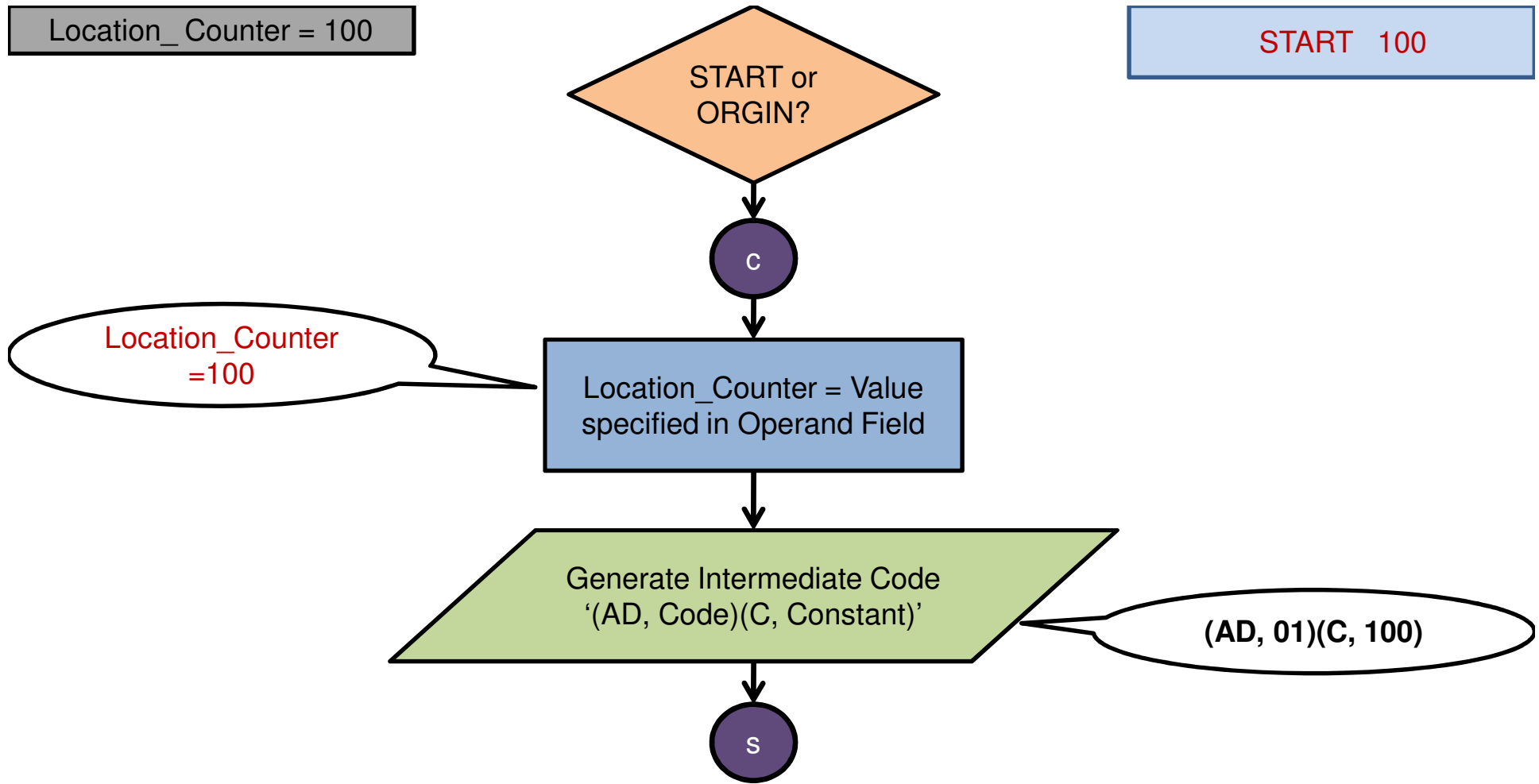
B DC '8'

C DS 1

END

Location_Counter = 100

START 100



Location_Counter = 100

Intermediate Code
(AD, 01)(C, 100)

START

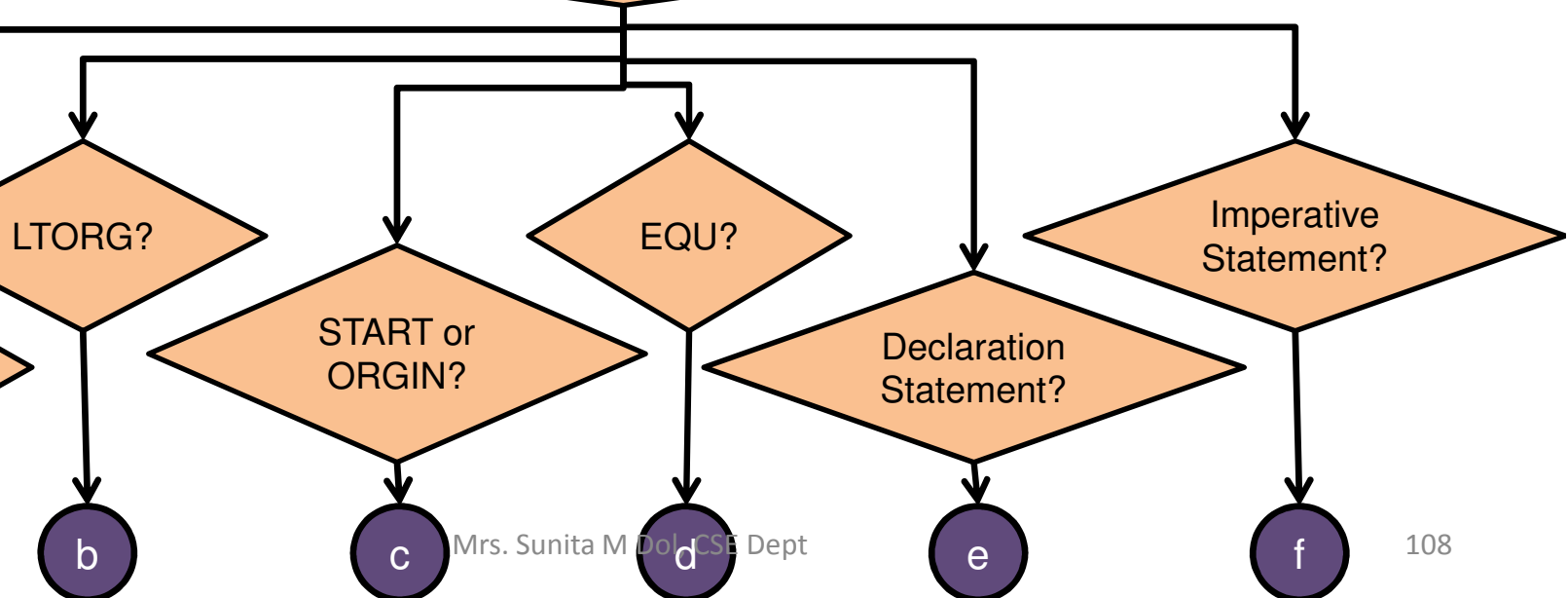
Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

Source Program
START 100
MOVER AREG, =5'
ADD AREG, B
MOVEM AREG, C
STOP
B DC '8'
C DS 1
END

S

While next statement is not end
statement



Location_Counter = 100
 Literal_Table_Pointer = 1
 POOL_Table_Pointer = 1

START 100
 MOVER AREG, ='5'

Imperative Statement?

f

Code = Machine Opcode from OPTAB

Code = 04

Location_Counter
 = 100+1
 =101

Location_Counter = Location_Counter + Instruction Length from OPTAB

Is Operand Literal?

Yes

This_Literal = Literal in Operand Field

Code = "'5'"

LITERAL_TABLE[Literal_Table_Pointer] = This_Literal

LITERAL_TABLE[1] = "'5'"

Literal_Table_Pointer = Literal_Table_Pointer + 1

Literal_Table_Pointer = 1+1 = 2

Literal_Table_Pointer	Literal	Address
1	'5'	
2		

Literal Table

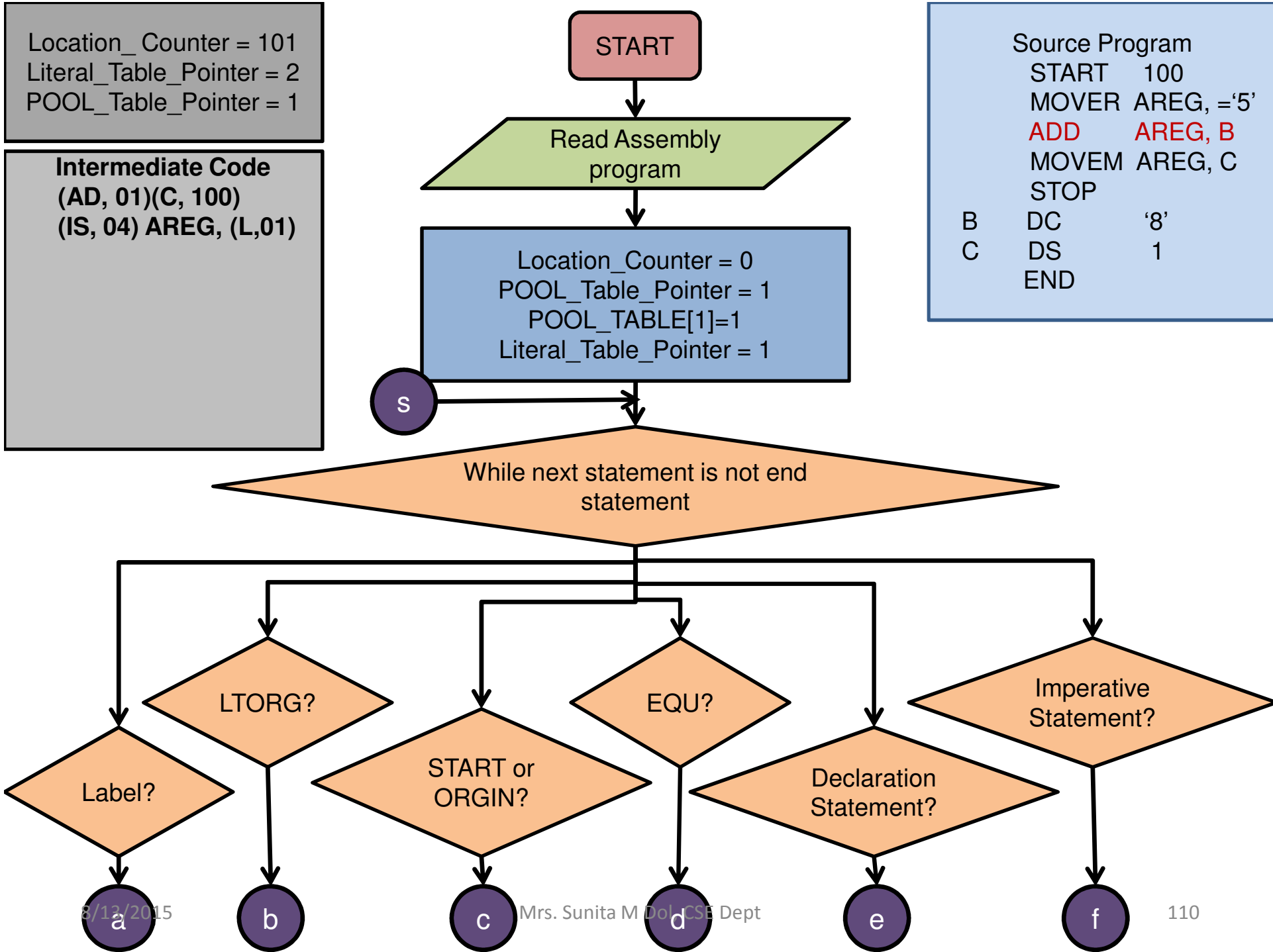
Generate Intermediate Code
 '(IS, Code) Register_Operand (L, This_Literal)'

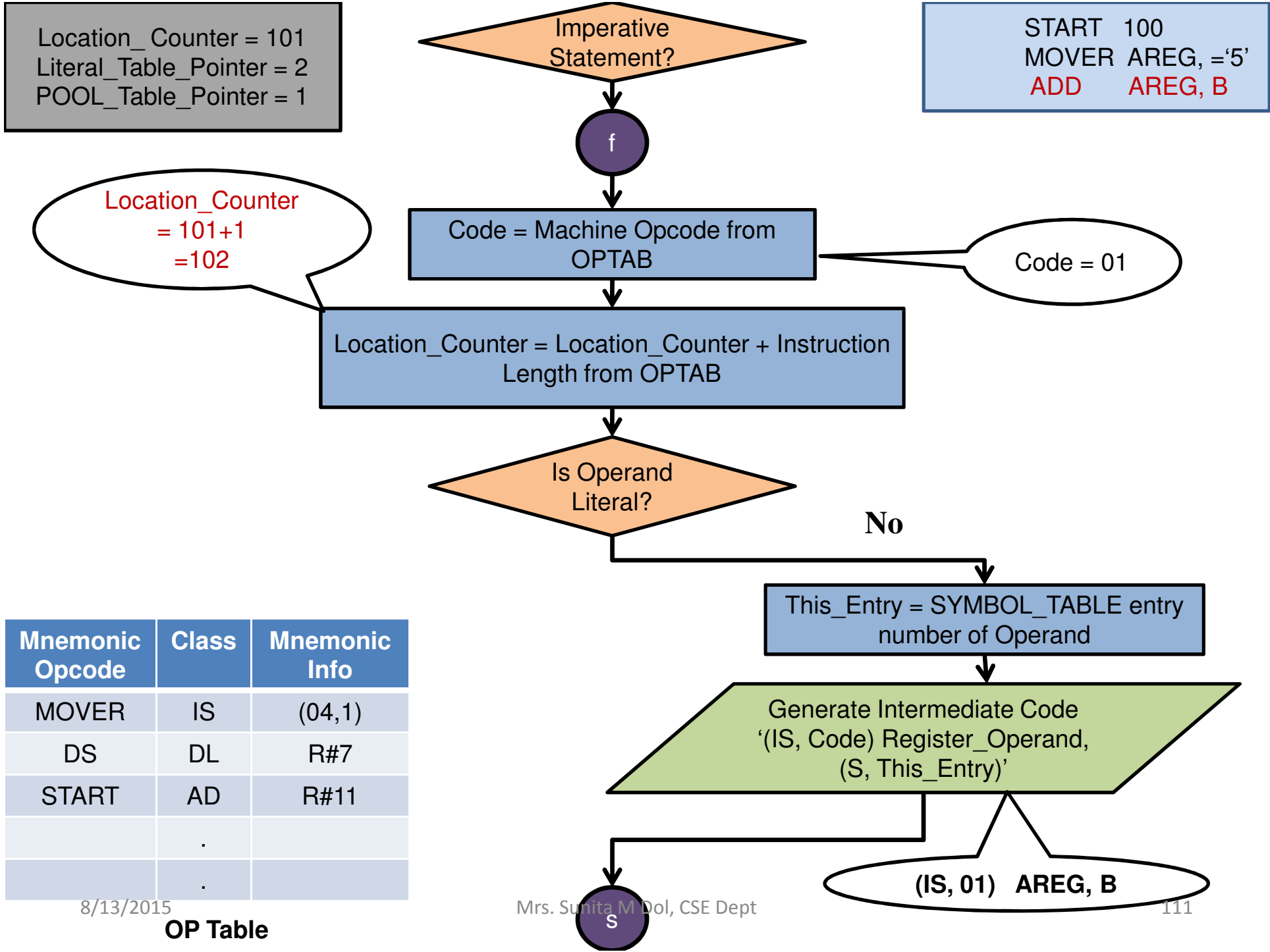
(IS, 04) AREG, (L,01)

POOL_Table_Pointer	Literal_Table_Pointer
1	1

POOL Table

S





OP Table

Location_Counter = 102
Literal_Table_Pointer = 2
POOL_Table_Pointer = 1

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B

START

Read Assembly
program

Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

S

While next statement is not end
statement

LTORG?

Label?

a

b

START or
ORIGIN?

c

EQU?

Declaration
Statement?

d

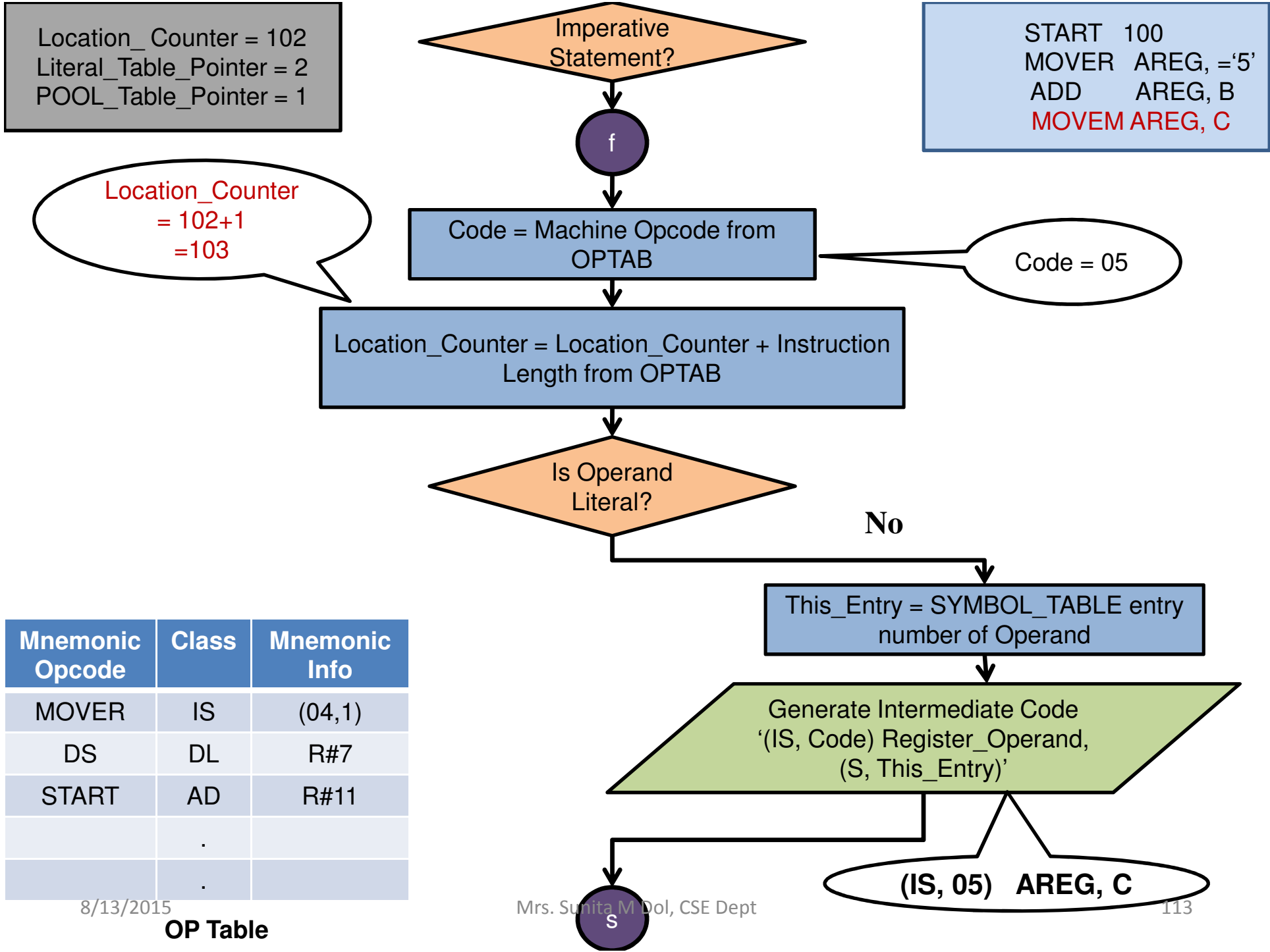
e

Imperative
Statement?

f

Source Program

```
START 100
MOVER AREG, ='5'
ADD AREG, B
MOVEM AREG, C
STOP
B DC '8'
C DS 1
END
```

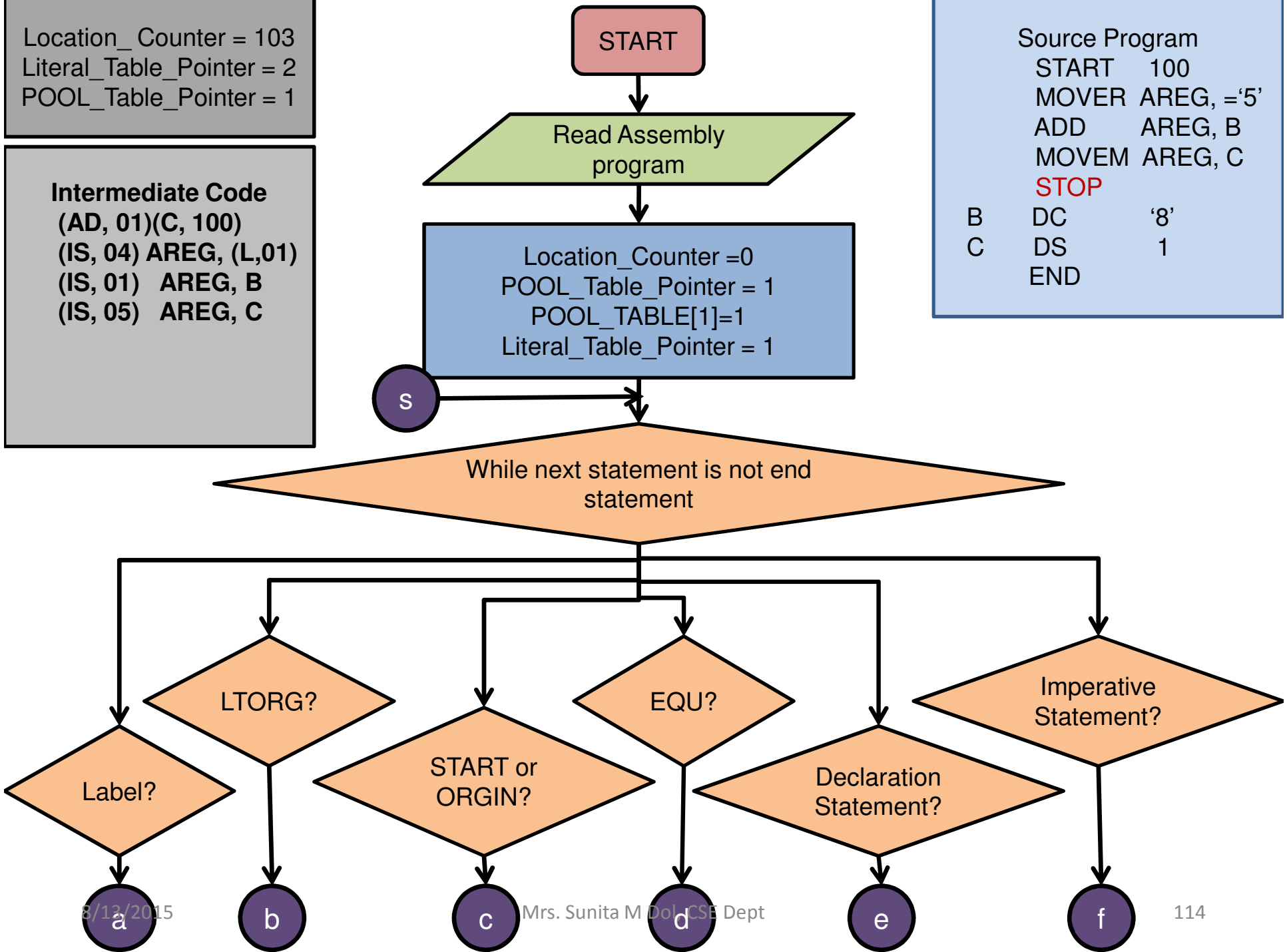



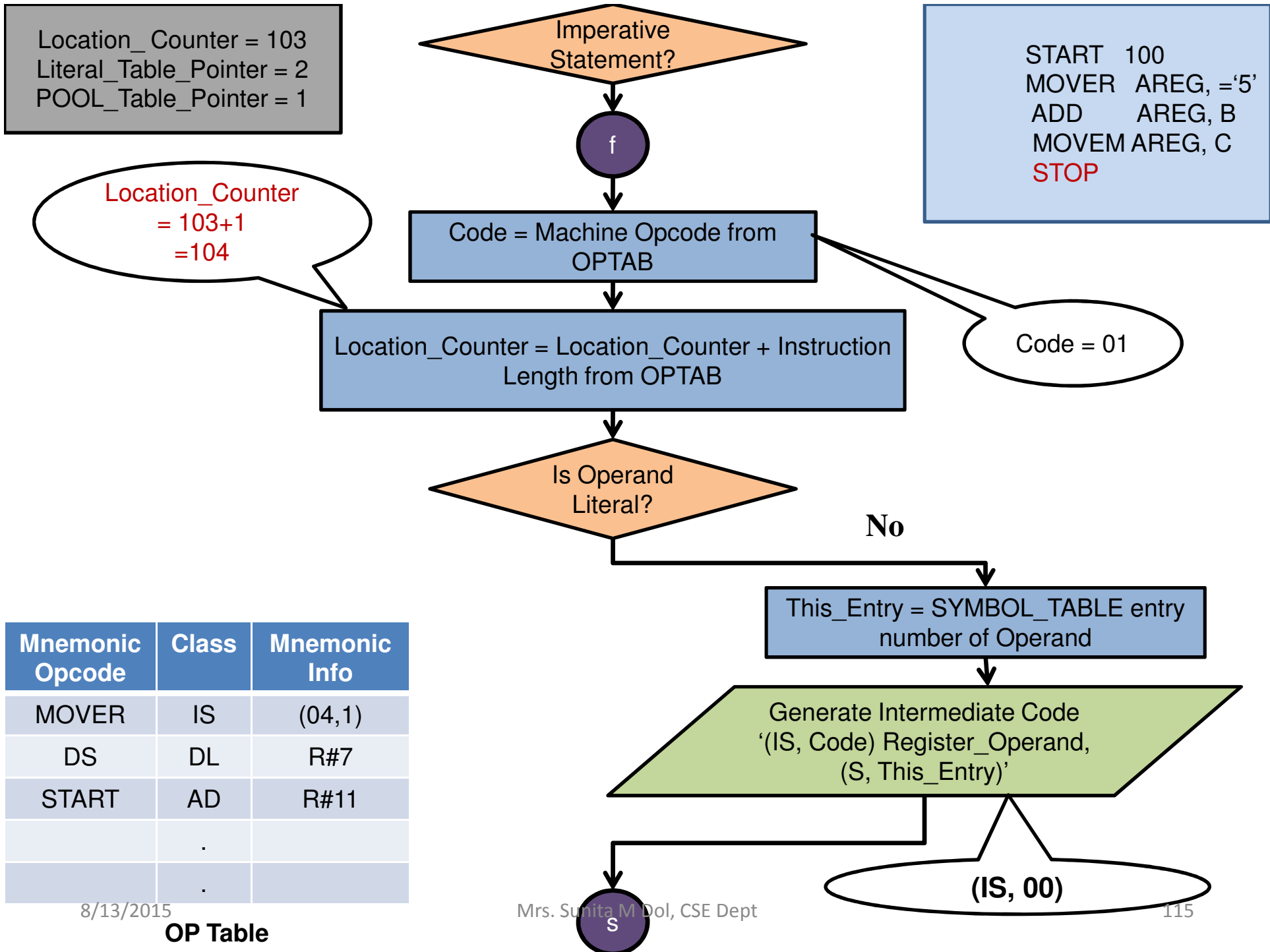
Location_Counter = 103
Literal_Table_Pointer = 2
POOL_Table_Pointer = 1

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C

Source Program

```
START 100
MOVER AREG, ='5'
ADD AREG, B
MOVEM AREG, C
STOP
B DC '8'
C DS 1
END
```





Location_Counter = 104
Literal_Table_Pointer = 2
POOL_Table_Pointer = 1

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, (L, 01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)

START

Read Assembly
program

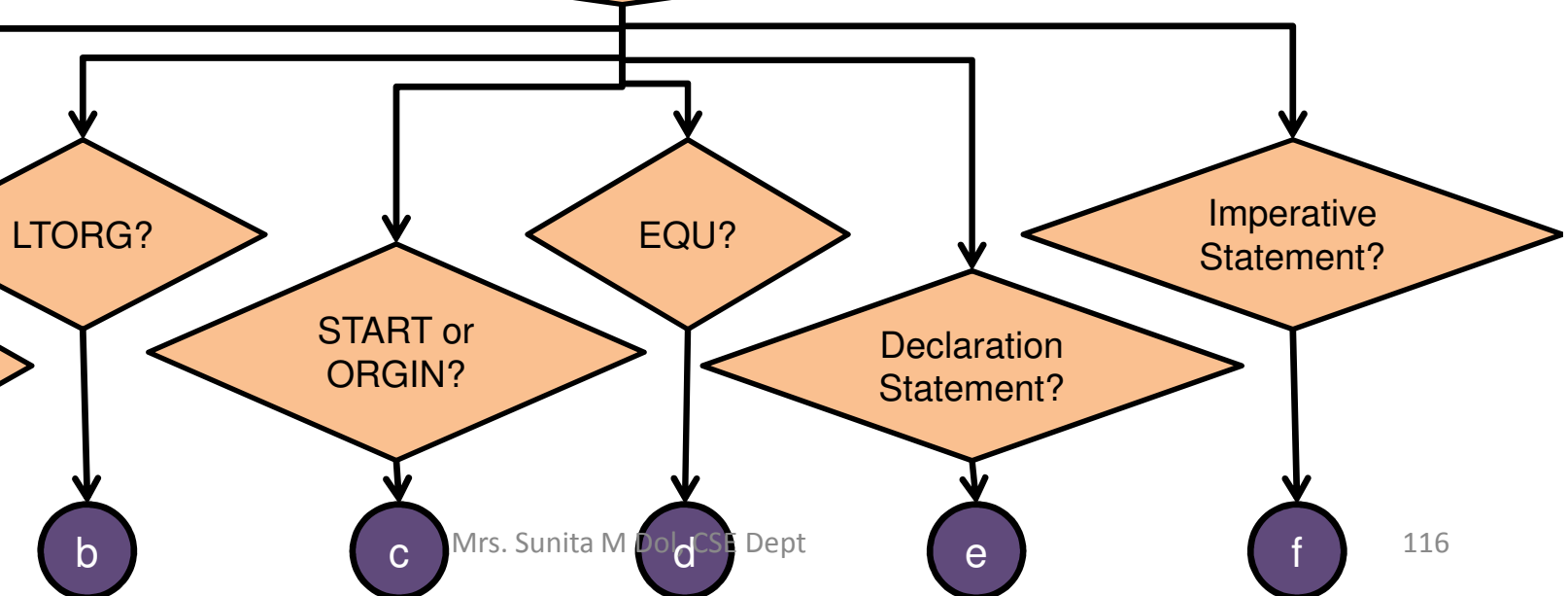
Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

Source Program
START 100
MOVER AREG, ='5'
ADD AREG, B
MOVEM AREG, C
STOP

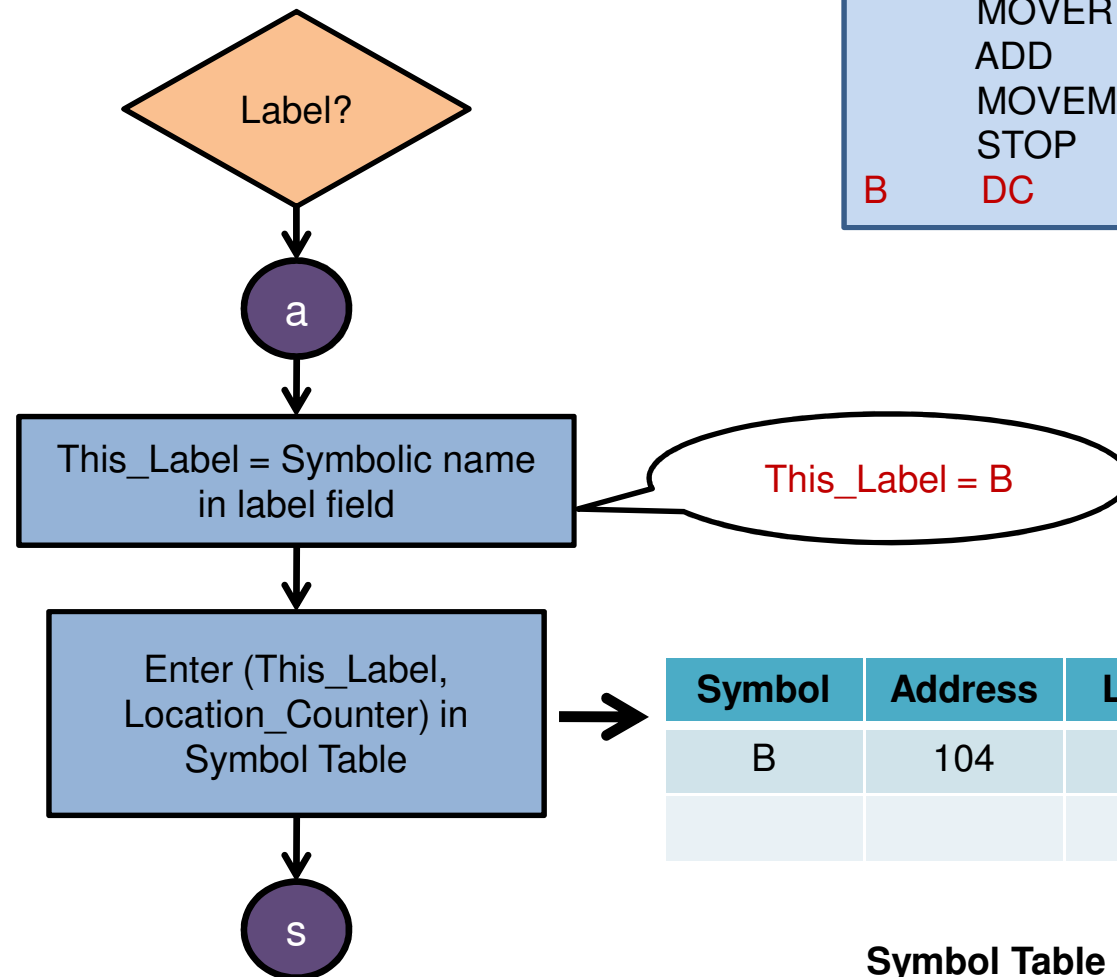
B DC '8'
C DS 1
END

S

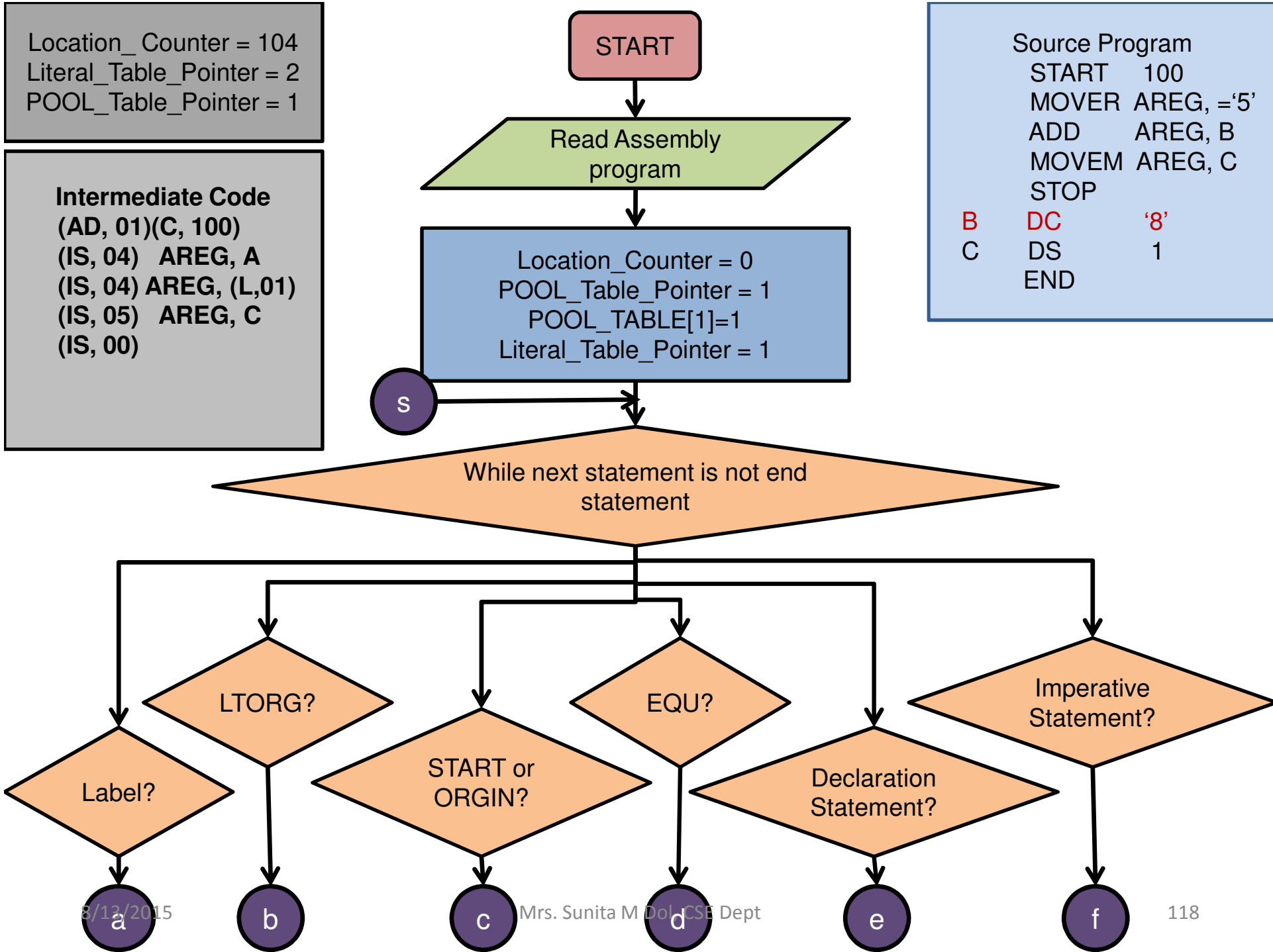
While next statement is not end
statement



Location_Counter = 104
Literal_Table_Pointer = 2
POOL_Table_Pointer = 1

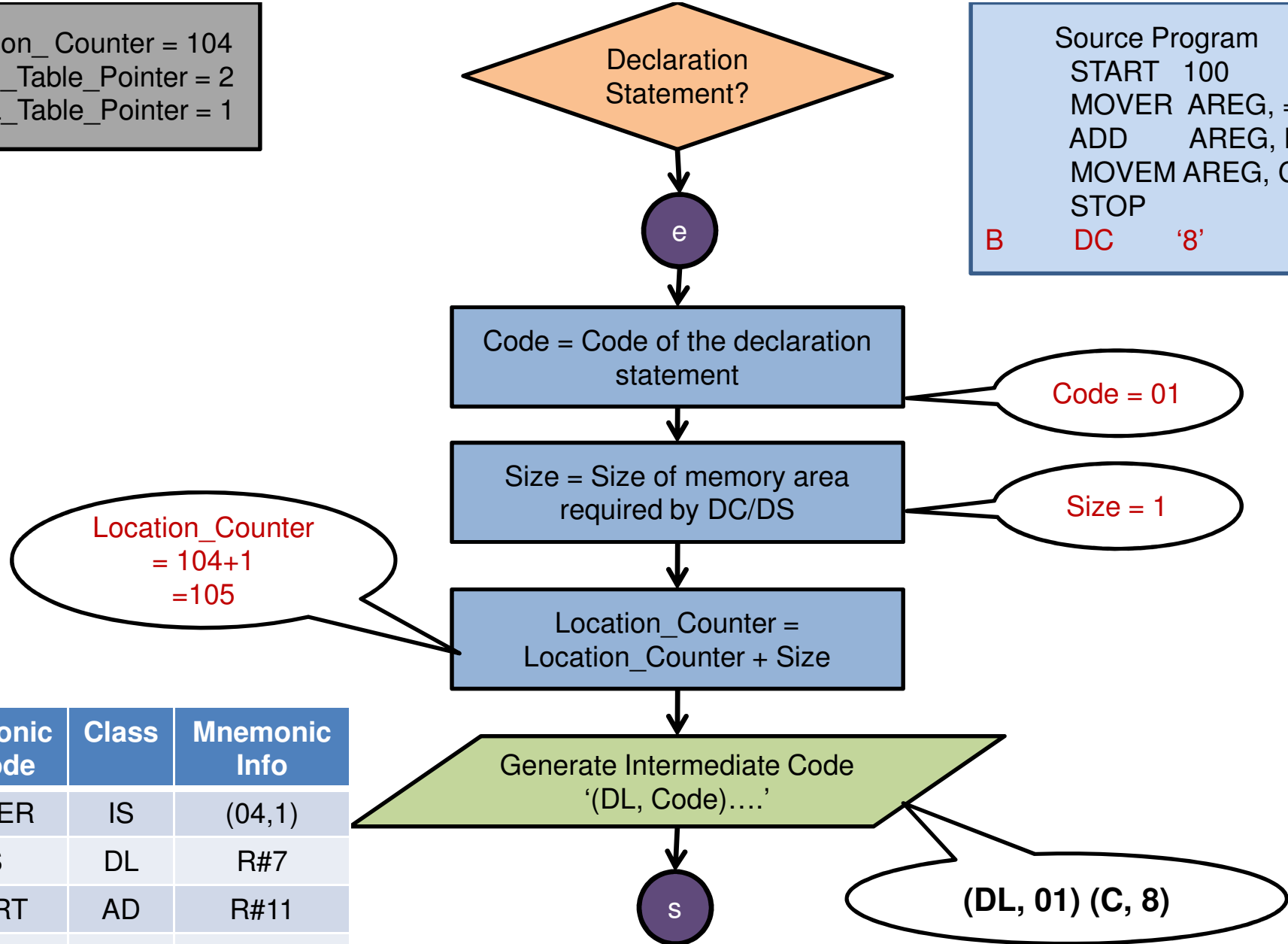


```
START 100
MOVER AREG, =5'
ADD    AREG, B
MOVEM AREG, C
STOP
B      DC    '8'
```

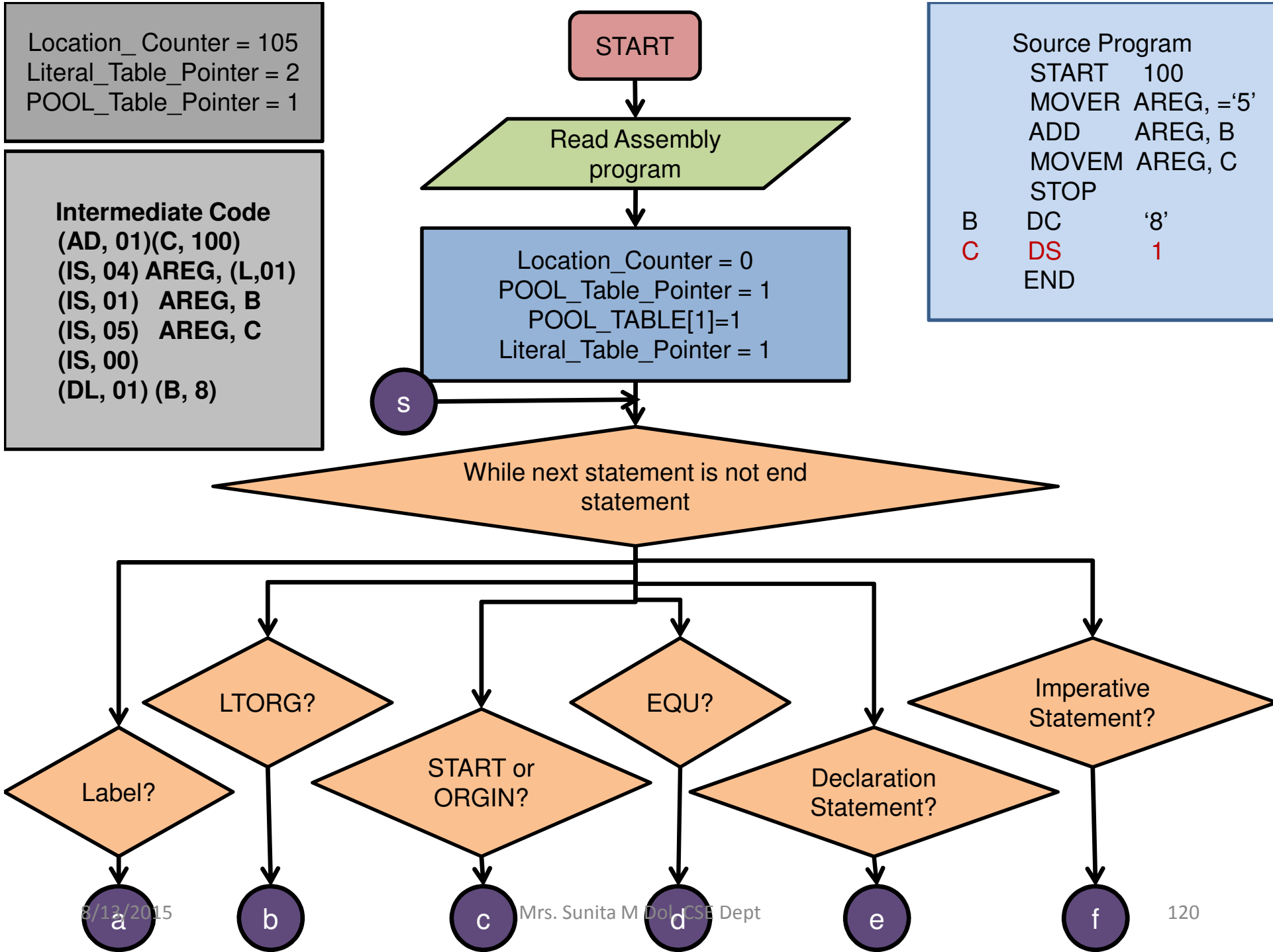


Location_Counter = 104
 Literal_Table_Pointer = 2
 POOL_Table_Pointer = 1

Source Program
 START 100
 MOVER AREG, =5'
 ADD AREG, B
 MOVEM AREG, C
 STOP
 B DC '8'



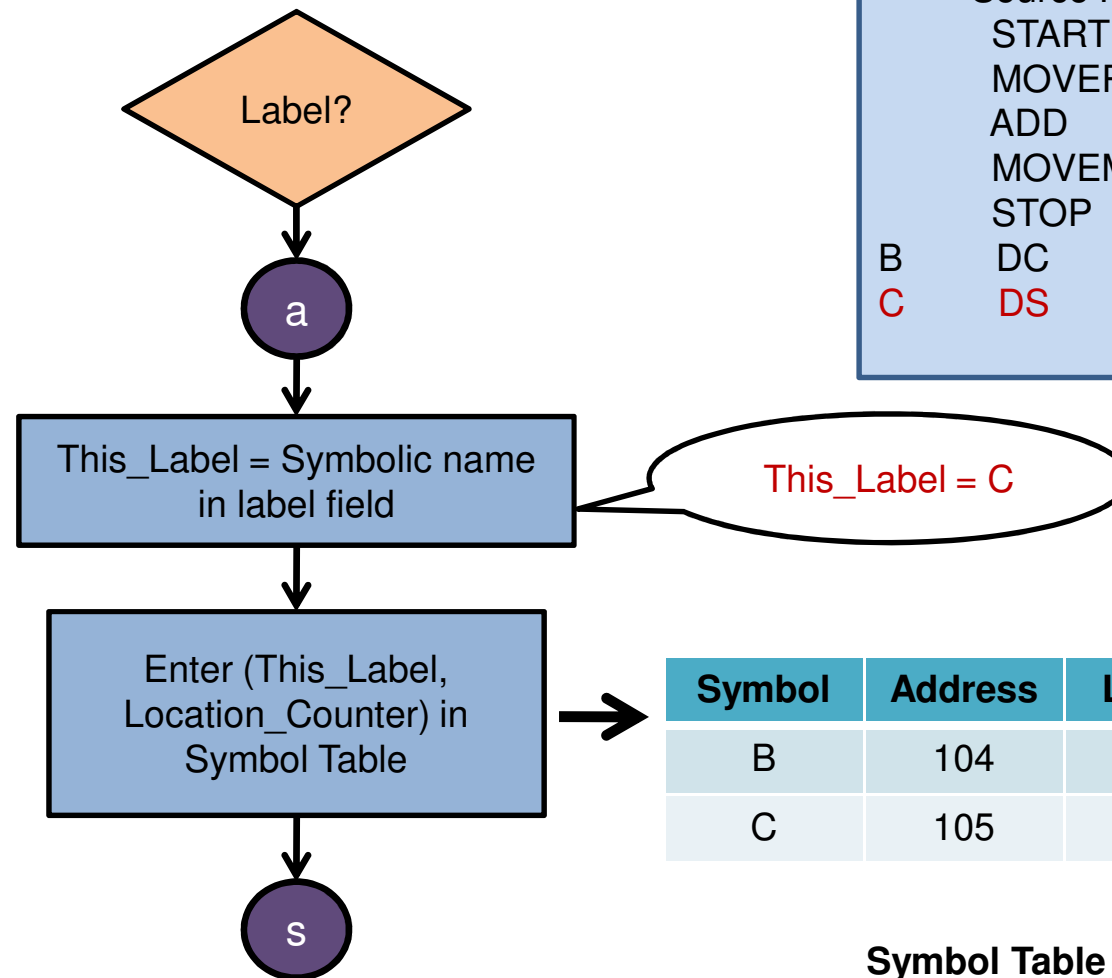
Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	



Location_Counter = 105
Literal_Table_Pointer = 2
POOL_Table_Pointer = 1

Source Program
START 100
MOVER AREG, =5'
ADD AREG, B
MOVEM AREG, C
STOP

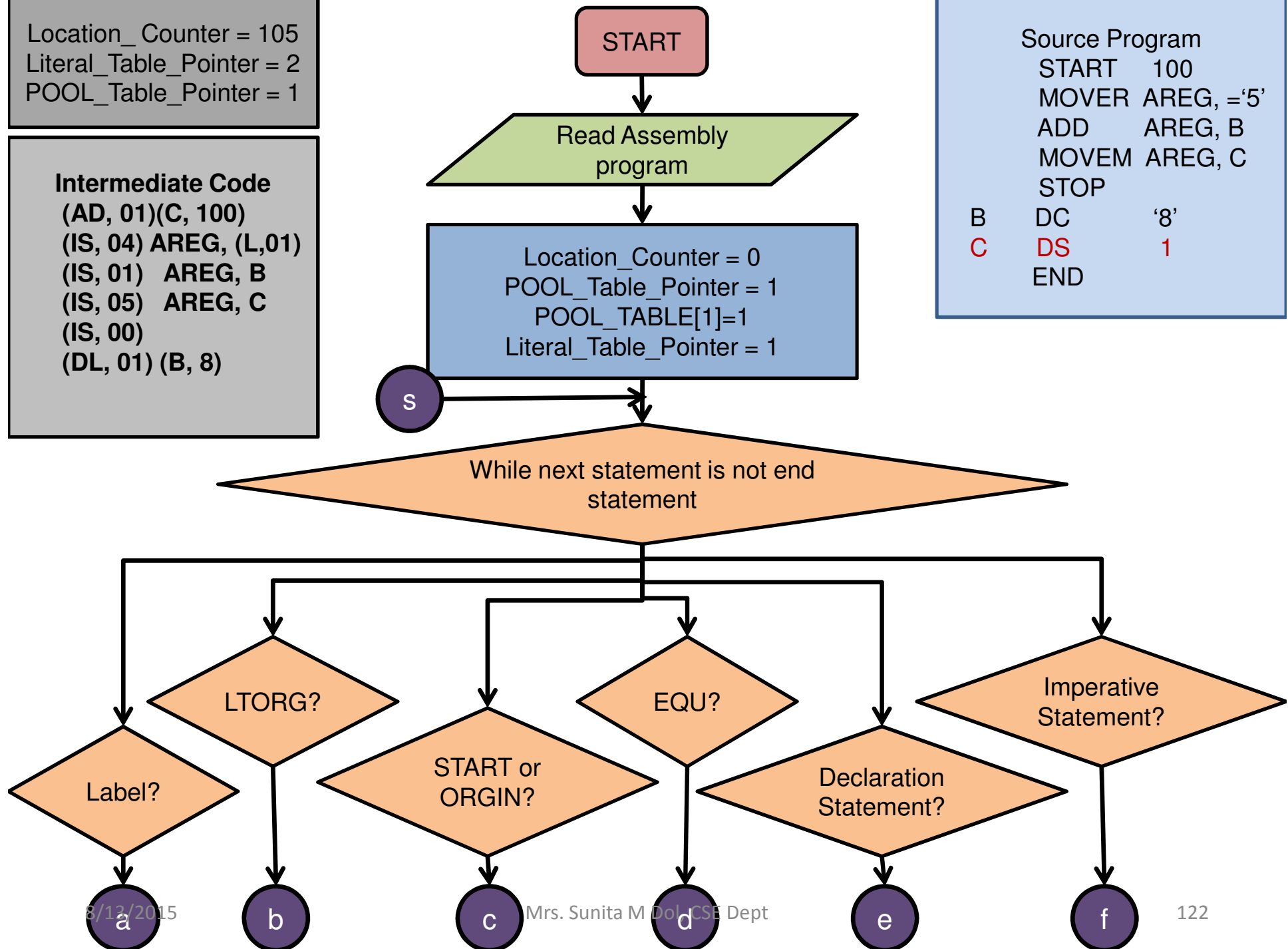
B DC '8'
C DS 1



Location_Counter = 105
 Literal_Table_Pointer = 2
 POOL_Table_Pointer = 1

Intermediate Code
 (AD, 01)(C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (B, 8)

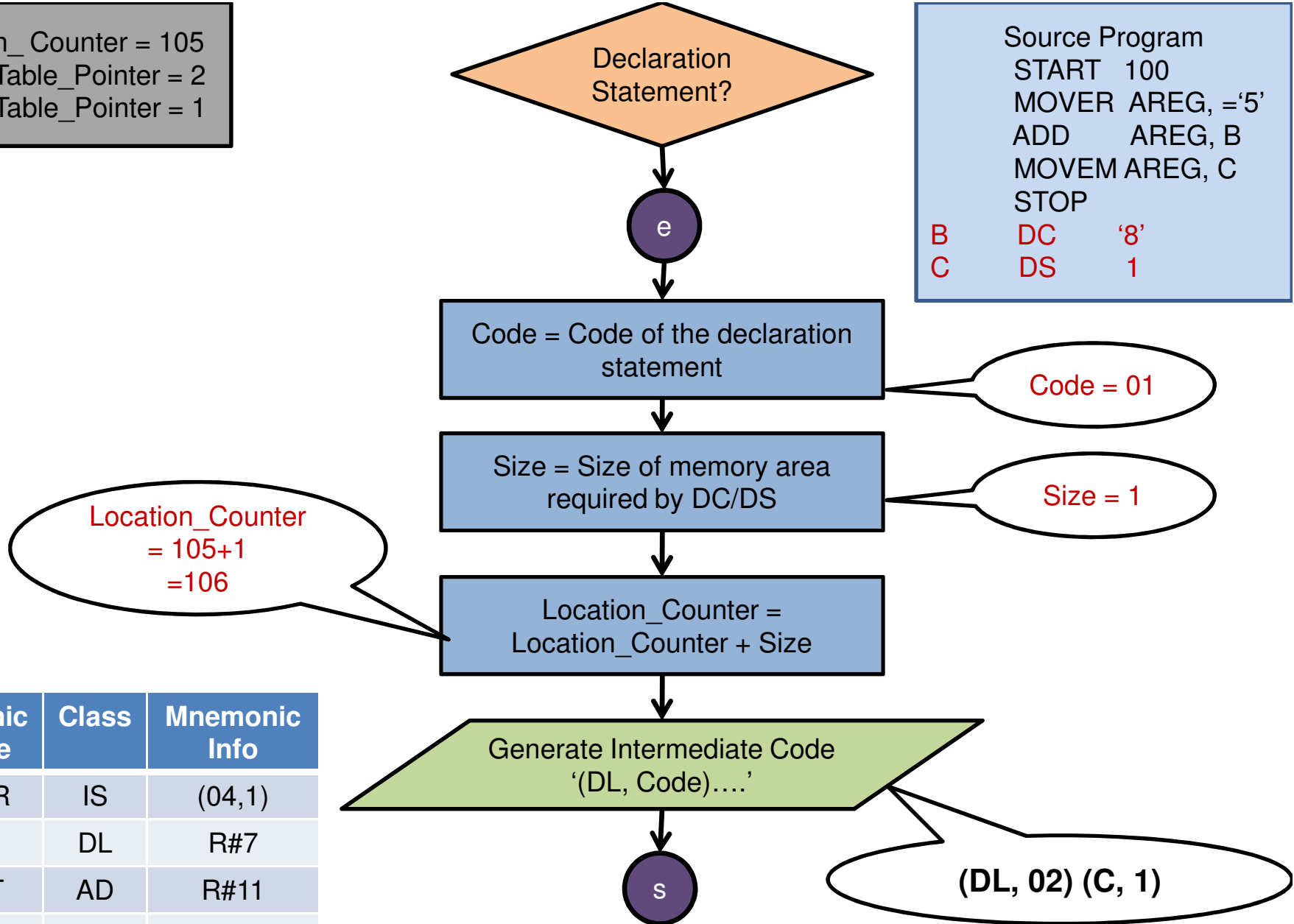
Source Program
 START 100
 MOVER AREG, =5'
 ADD AREG, B
 MOVEM AREG, C
 STOP
 B DC '8'
 C DS 1
 END



Location_Counter = 105
 Literal_Table_Pointer = 2
 POOL_Table_Pointer = 1

Source Program
 START 100
 MOVER AREG, =5'
 ADD AREG, B
 MOVEM AREG, C
 STOP

B DC '8'
 C DS 1



Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

Location_Counter = 106
Literal_Table_Pointer = 2
POOL_Table_Pointer = 1

Intermediate Code
(AD, 01)(C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

START

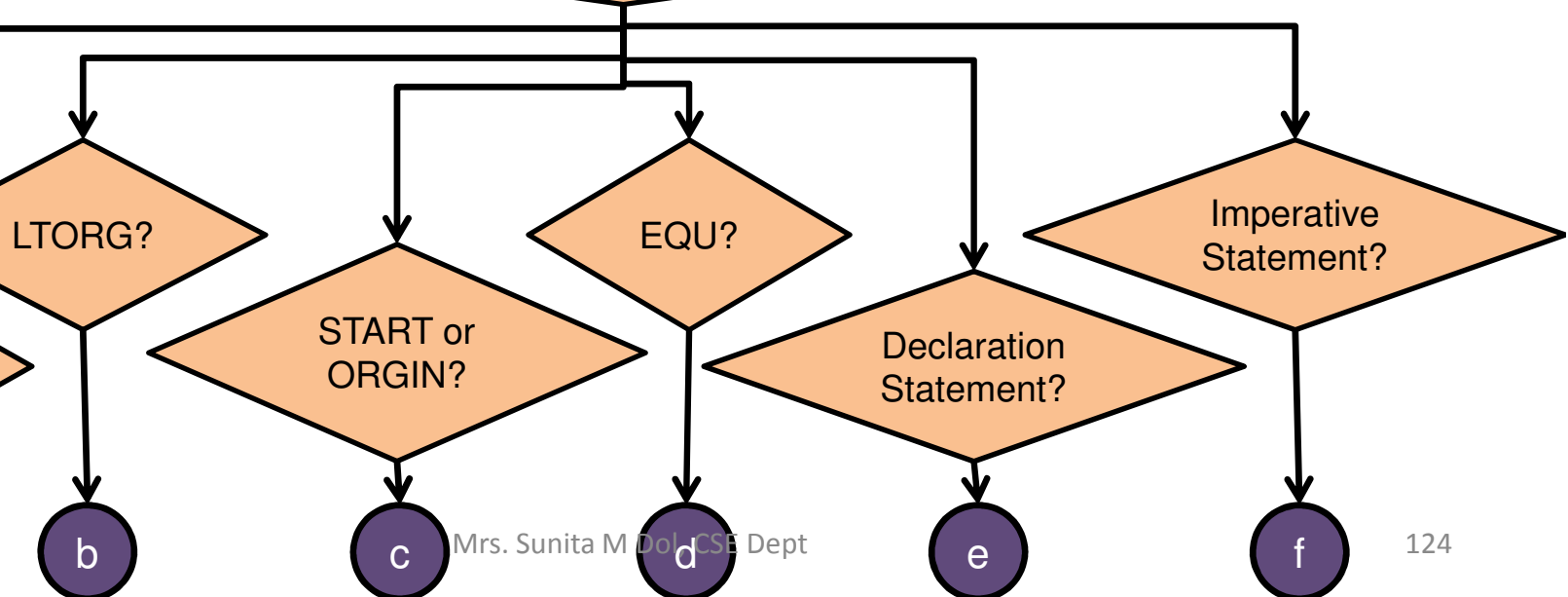
Read Assembly
program

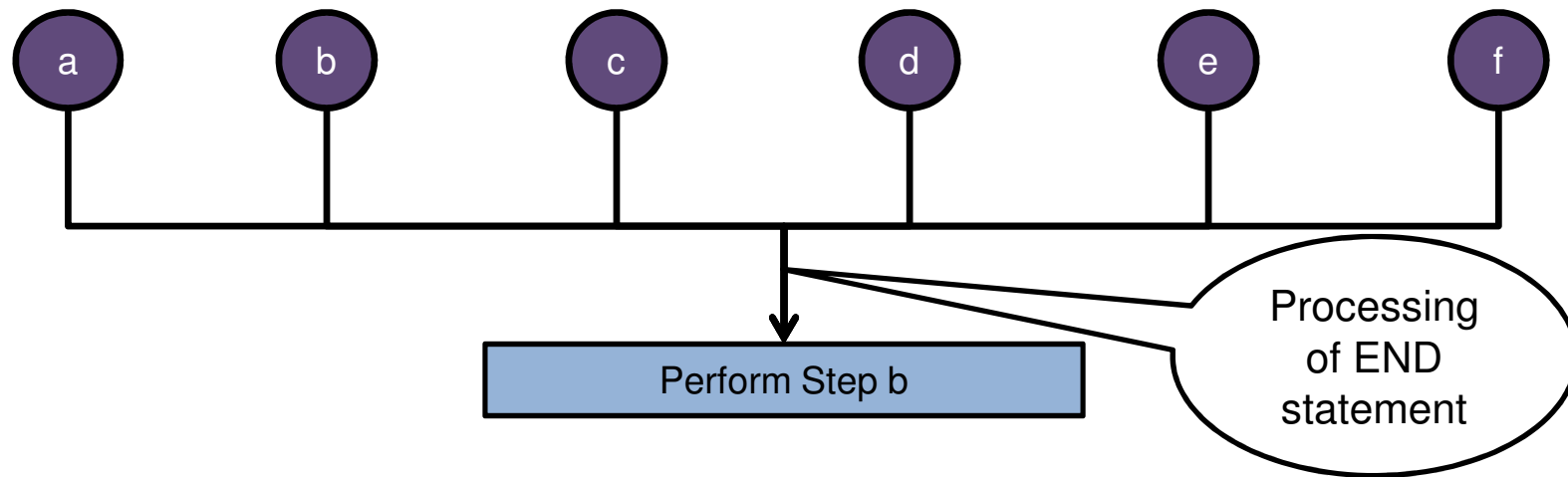
Location_Counter = 0
POOL_Table_Pointer = 1
POOL_TABLE[1]=1
Literal_Table_Pointer = 1

Source Program
START 100
MOVER AREG, =5'
ADD AREG, B
MOVEM AREG, C
STOP
B DC '8'
C DS 1
END

S

While next statement is not end
statement





Location_Counter = 106
 Literal_Table_Pointer = 2
 POOL_Table_Pointer = 1

Intermediate Code
 (AD, 01)(C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Source Program
 START 100
 MOVER AREG, =5'
 ADD AREG, B
 MOVEM AREG, C
 STOP
 B DC '8'
 C DS 1
 END 125

Location_Counter = 106
 Literal_Table_Pointer = 2
 POOL_Table_Pointer = 1

Intermediate Code
 (AD, 01)(C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)



Source Program
 START 100
 MOVER AREG, =‘5’
 ADD AREG, B
 MOVEM AREG, C
 STOP
 B DC ‘8’
 C DS 1
 END

Process literals
 LITERAL_TABLE[POOL_TABLE[POOL_Table_Pointer]]...LITER
 AL_TABLE[Literal_Table_Pointer – 1]] to allocate memory and put
 addresses in address field

Location_Counter
 = 106+1 =107

Update Location_Counter

POOL_Table_Pointer
 = 1+1 = 2

POOL_Table_Pointer =
 POOL_Table_Pointer + 1

POOL_TABLE
 [POOL_Table_Pointer]
 =2

POOL_TABLE [POOL_Table_Pointer]
 = Literal_Table_Pointer

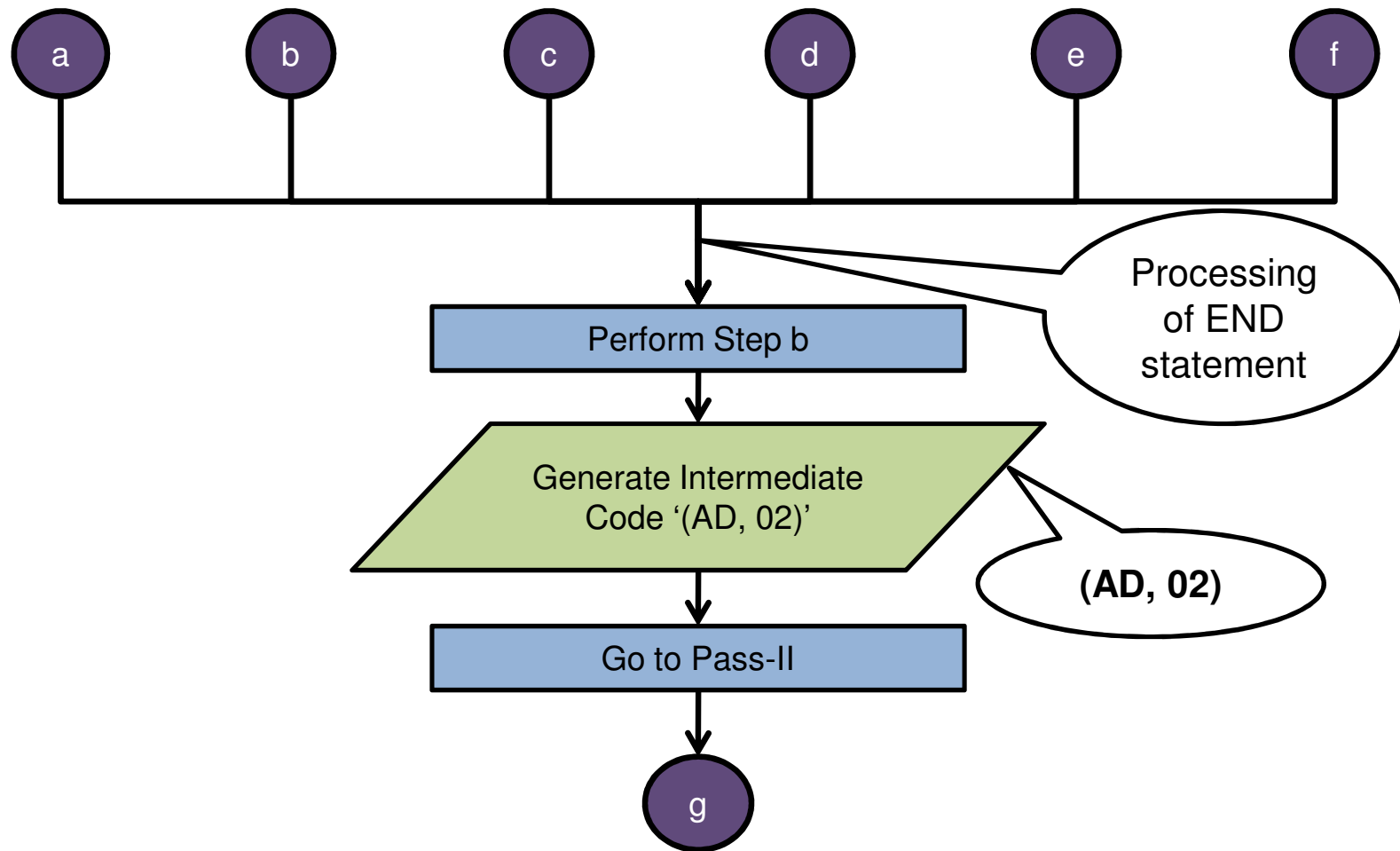
Literal_ Table_P ointer	Literal	Address
1	=‘5’	106
2		

Literal Table

POOL_Table_P ointer	Literal_Table_P ointer
1	1
2	2

POOL Table





Source Program

```

START 100
MOVER AREG, ='5'
ADD    AREG, B
MOVEM AREG, C
STOP
B      DC      '8'
C      DS      1
END

```

Intermediate Code

```

(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)
(AD, 02)

```

Symbol	Address	Length
B	104	1
C	105	

Symbol Table

Literal_Table_P ointer	Literal	Address
1	= '5'	106
2		

Literal Table

POOL_Table_ Pointer	Literal_Table_ Pointer
1	1
2	2

POOL Table

Intermediate Code Forms

- The intermediate code consist of a set of IC unit, each IC unit consisting of following three fields
 - Address
 - Representation of the mnemonic opcode
 - Representation of operands

Address	Opcode	Operands
---------	--------	----------

Intermediate Code Forms

- Mnemonic field
 - Mnemonic field contains a pair of the form (statement class, code) where statement class can be one of the following
 - IS – Imperative statement
 - DL – Declaration statement
 - AD – Assembler directivesand code is the instruction opcode in the machine language.

Intermediate Code Forms

- Mnemonic field

Declaration statement

DC	01
DS	02

Assembler Directives

START	01
END	02
ORIGIN	03
EQU	04
LTORG	05

Intermediate Code Forms

- Intermediate code for Imperative statement
 - The first operand is represented by a single digit
 - 1-4 for AREG-DREG
 - 1-6 for LT-ANY
 - The second operand which is a memory operand is represented by
(operand class, code)
where operand class is one of C, S, L.

Intermediate Code Forms

- Intermediate code for Imperative statement
 - For constant, the code field contains the internal representation of the constant itself.
 - For symbol or literals, code field contain the ordinal number of the operand's entry in SYMTAB or LITTAB

Intermediate Code Forms

- Variant-I

	Source Program		Intermediate Code
	START 100	—————→	(AD, 01) (C, 100)
	MOVER AREG, ='5'	—————→	(IS, 04) (1), (L,01)
	ADD AREG, B	—————→	(IS, 01) (1), (S, 01)
	MOVEM AREG, C	—————→	(IS, 05) (1), (S, 02)
	STOP	—————→	(IS, 00)
B	DC '8'	—————→	(DL, 01) (C, 8)
C	DS 1	—————→	(DL, 02) (C, 1)
		—————→	(DL, 01) (C, 5)
	END	—————→	(AD, 02)

Intermediate Code Forms

- Variant-I
 - In Variant-I, two kinds of entries may exist in SYMTAB at any time for defined symbol and for forward references.

Intermediate Code Forms

- Variant-II
 - For declarative statements and assembler directives, processing of the operand fields is essential to support LC processing.
 - For imperative statements, the operand field is processed only to identify literal references.
 - Symbolic references in the source statement are not processed at all during Pass-I

Intermediate Code Forms

- Variant-I

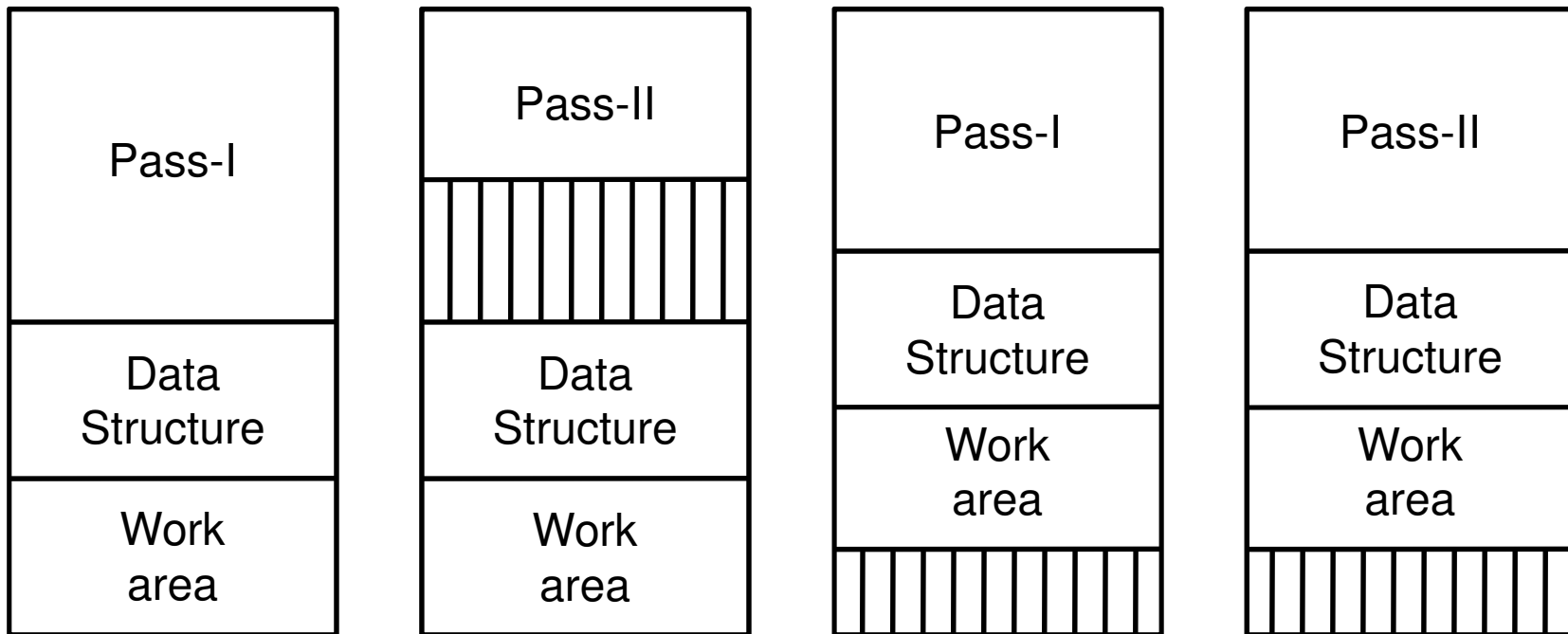
Source Program		Intermediate Code
	START 100	→ (AD, 01) (C, 100)
	MOVER AREG, ='5'	→ (IS, 04) AREG, (L,01)
	ADD AREG, B	→ (IS, 01) AREG, B
	MOVEM AREG, C	→ (IS, 05) AREG, C
	STOP	→ (IS, 00)
B	DC '8'	→ (DL, 01) (C, 8)
C	DS 1	→ (DL, 02) (C, 1)
		→ (DL, 01) (C, 5)
	END	→ (AD, 02)

Intermediate Code Forms

- Comparison of Variant-I and Variant-II
 - Variant-I of the intermediate code appears to require extra work in Pass-I.
 - IC is quite compact in Variant-I
 - Variant-II reduces the work of pass-I by transferring the burden of operand processing from Pass-I to Pass-II of the assembler.
 - IC is less compact in Variant-II.

Intermediate Code Forms

- Comparison of Variant-I and Variant-II
 - Memory requirement using Variant-I and Variant-II.



Intermediate Code Forms

- Processing of Declaration & Assembler Directives
 - **DC**: A DC statement must be represented in IC
 - **START or ORIGIN**: It is not necessary to retain START and ORIGIN statement in IC if the IC contains an address field.
 - **LTORG**: The IC for a literal can be made identical to the IC for a DC statement so that no special processing is required in Pass-II.

2. Assemblers

- Elements of Assembly Language Programming
- A Simple Assembly Scheme
- Pass Structure of Assemblers
- Design of a Two Pass Assembler
- A Single Pass Assembler for IBM PC

Algorithm for PASS-II

1. **code_area_address** := address of code_area;
 pooltab_ptr := 1;
 loc_cntr := 0;
2. **While next statement is not an END statement**
 - a) clear machine_code_buffer;
 - b) **If an LTORG statement**
 - i) Process literals in LITTAB[POOLTAB[pooltab_ptr]]
 ...LITTAB[POOLTAB[pooltab_ptr+1]]-1 similar to
 processing of constants in a DC statement i.e. assemble
 the literals in machine_code_buffer;
 - ii) size := size of memory area required for literals;
 - iii) pooltab_ptr := pooltab_ptr + 1;

c) If a START or ORIGIN statement then

- i) loc_cntr := value specified in operand field;
- ii) size := 0;

d) If a declaration statement

- i) If a DC statement then
 Assemble the constant in machine_code_buffer
- ii) size := size of memory area required by DC/DS;

e) If an imperative statement

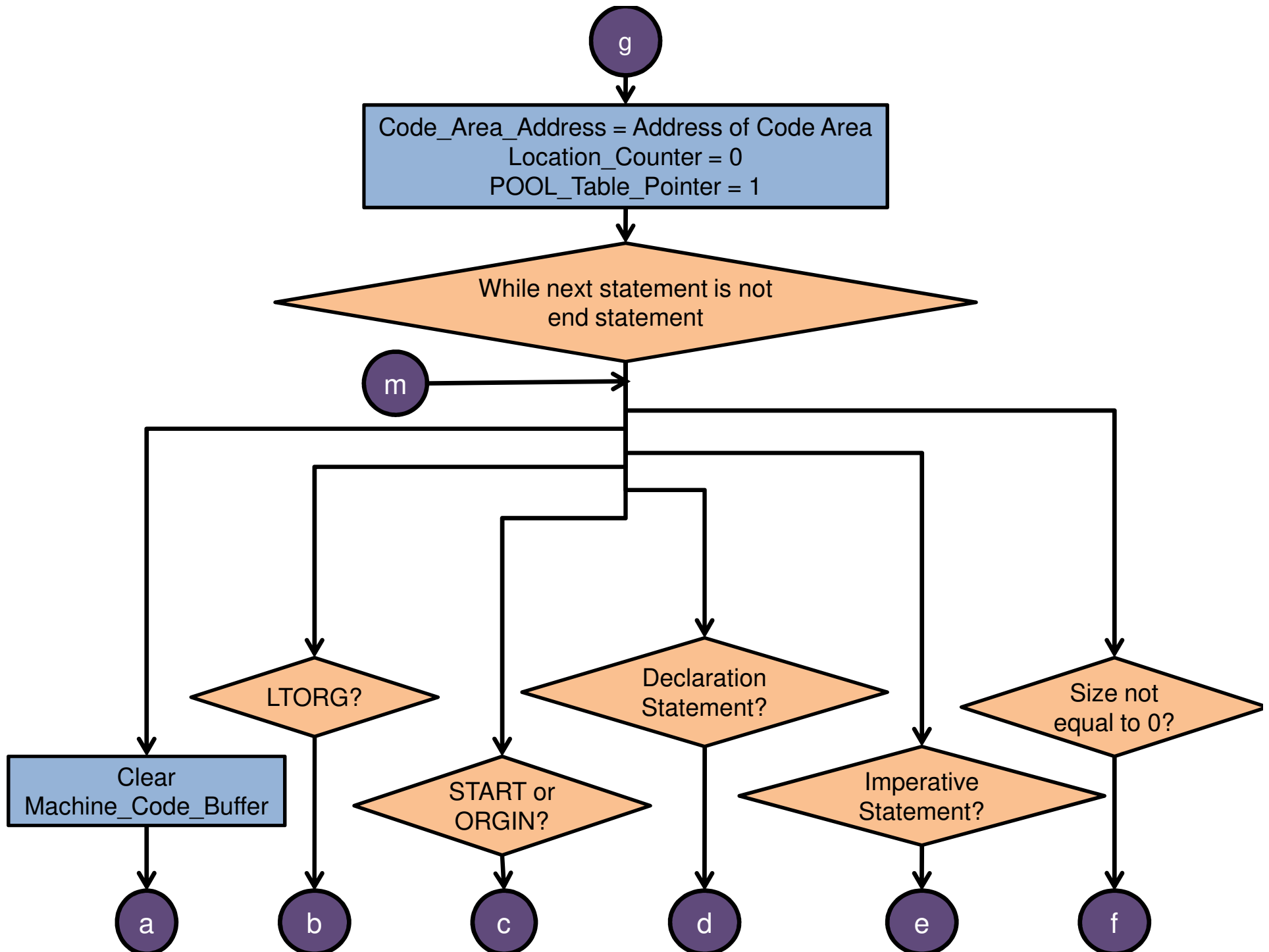
- i) Get the operand address from SYMTAB or LITAB.
- ii) Assemble instruction in machine_code_buffer.
- iii) size := size of instruction.

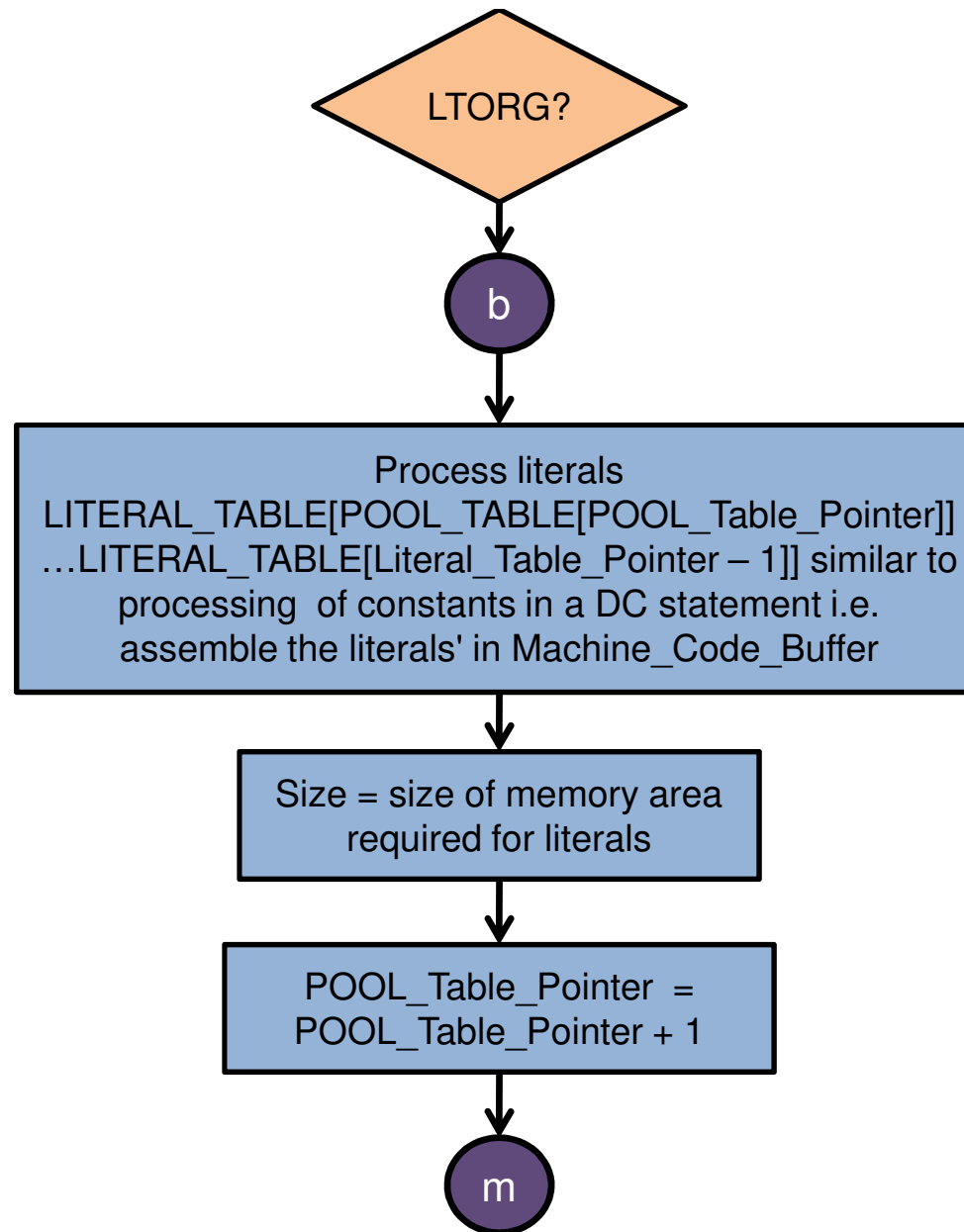
f) If size <> 0 then

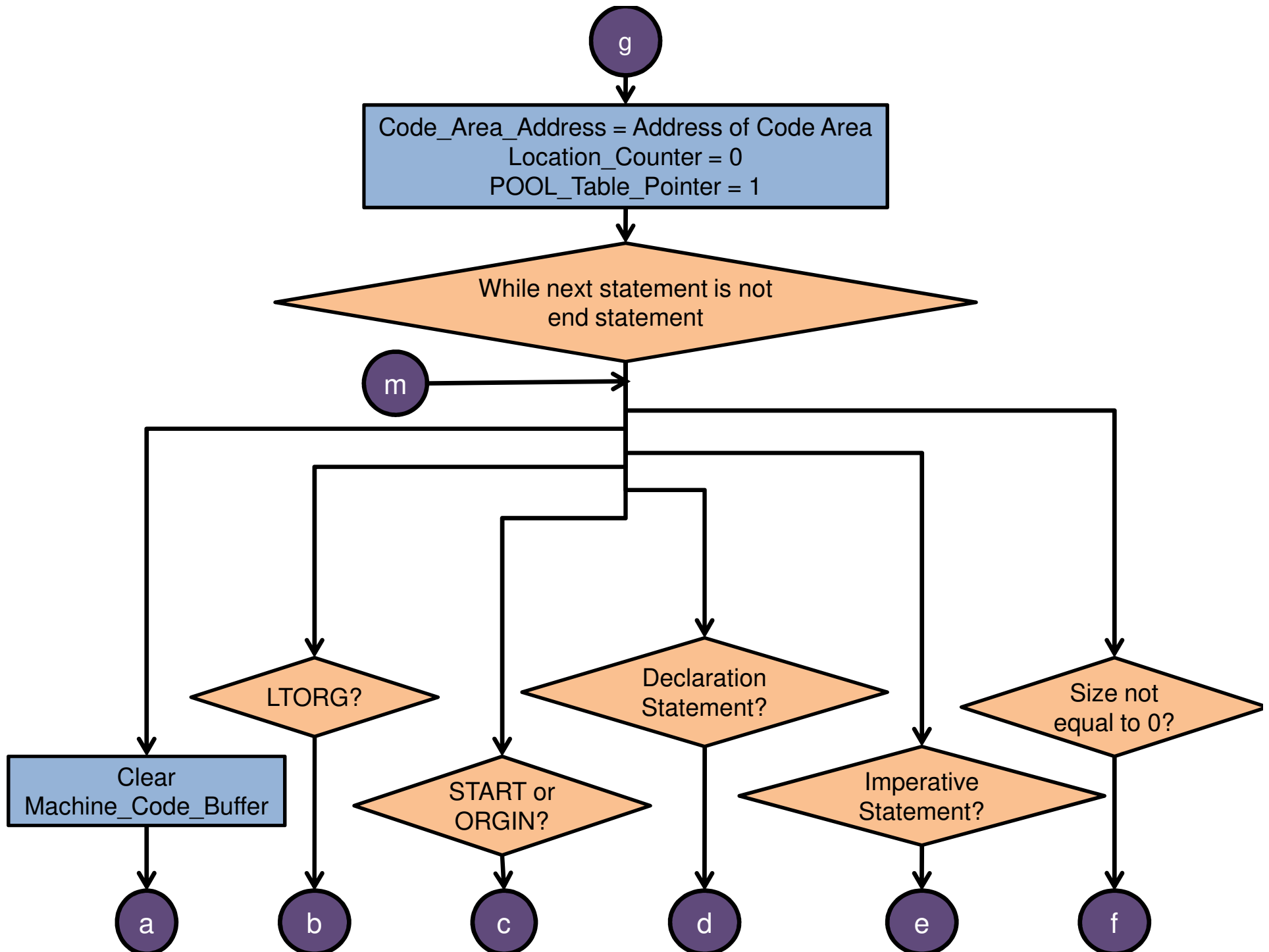
- i) Move the contents of machine_code_buffer to the address code_area_address + loc_cntr;
- ii) loc_cntr := loc_cntr + size;

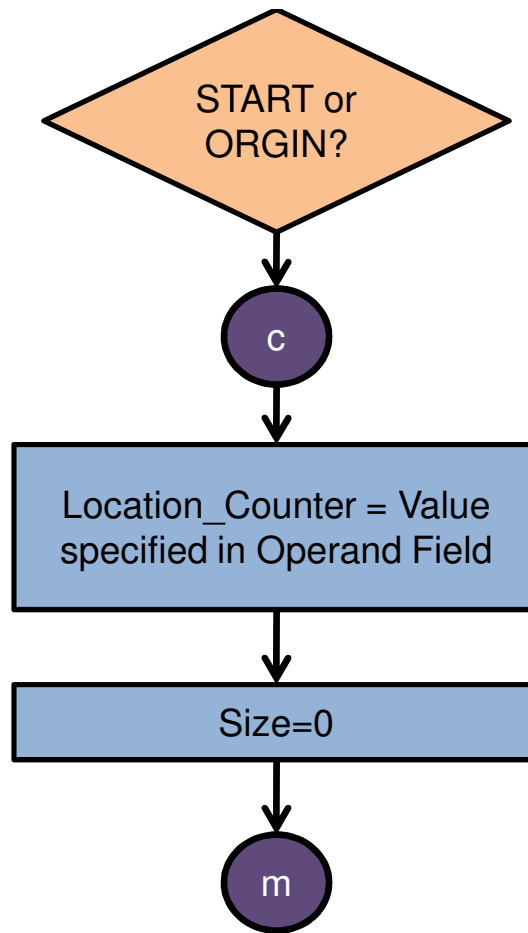
3. (Processing of an END statement)

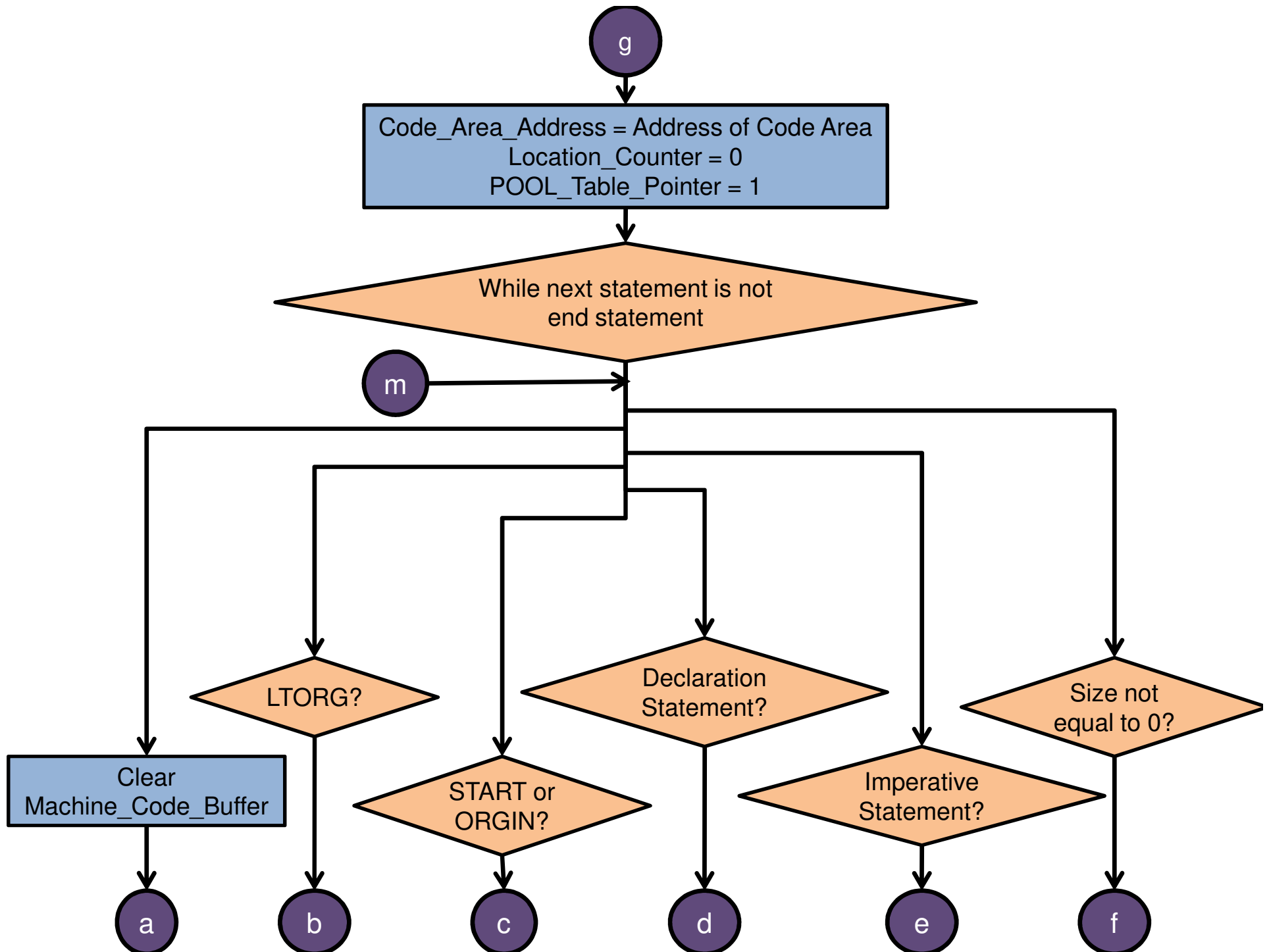
- a) Perform steps 2(b) & 2(f).**
- b) Write code_area into output file.**

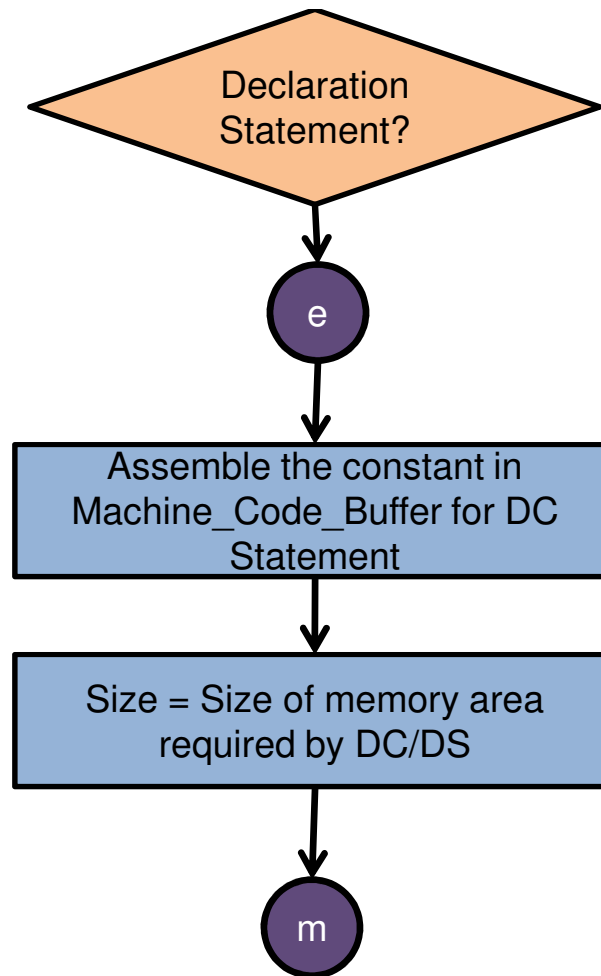


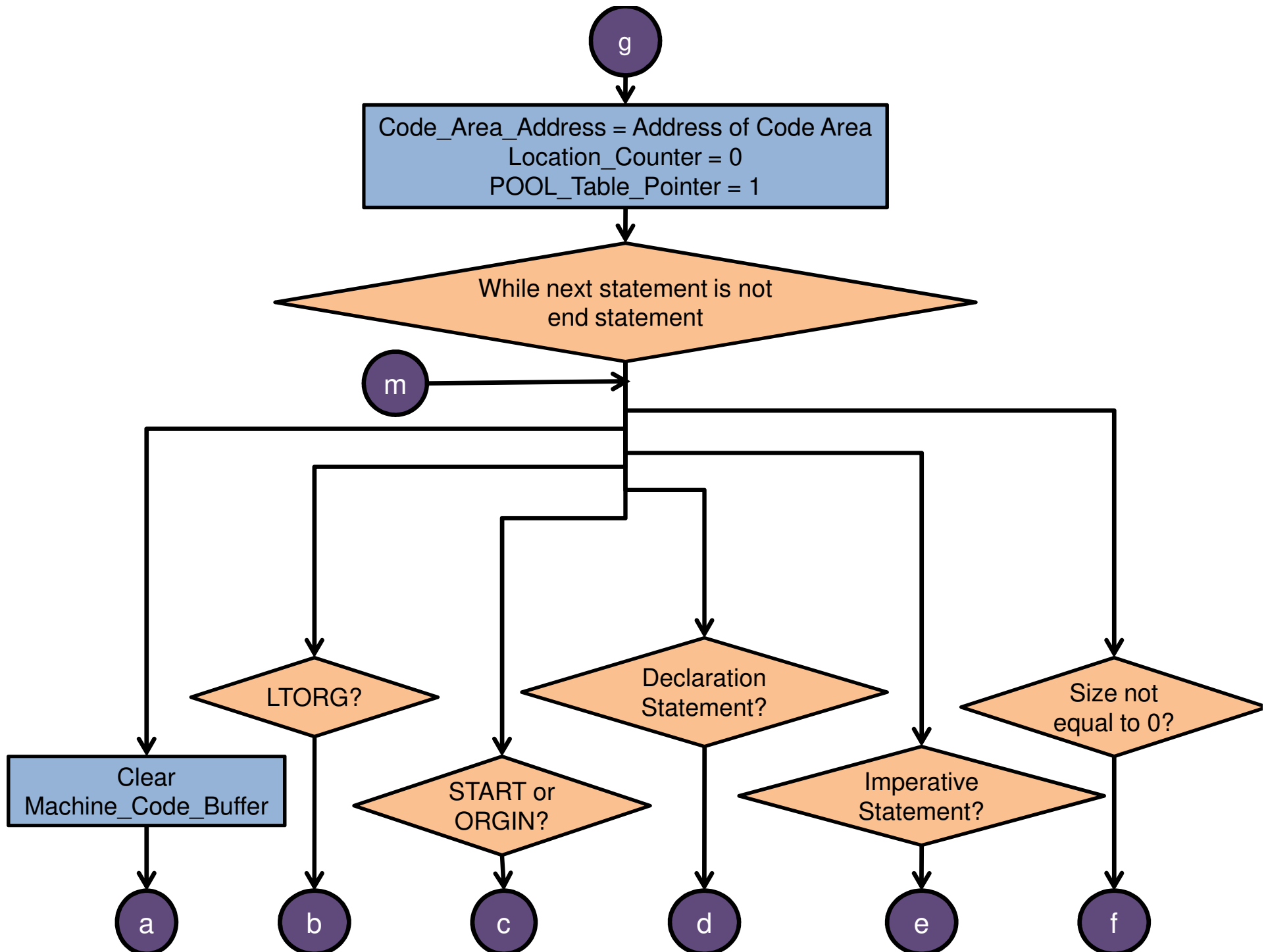


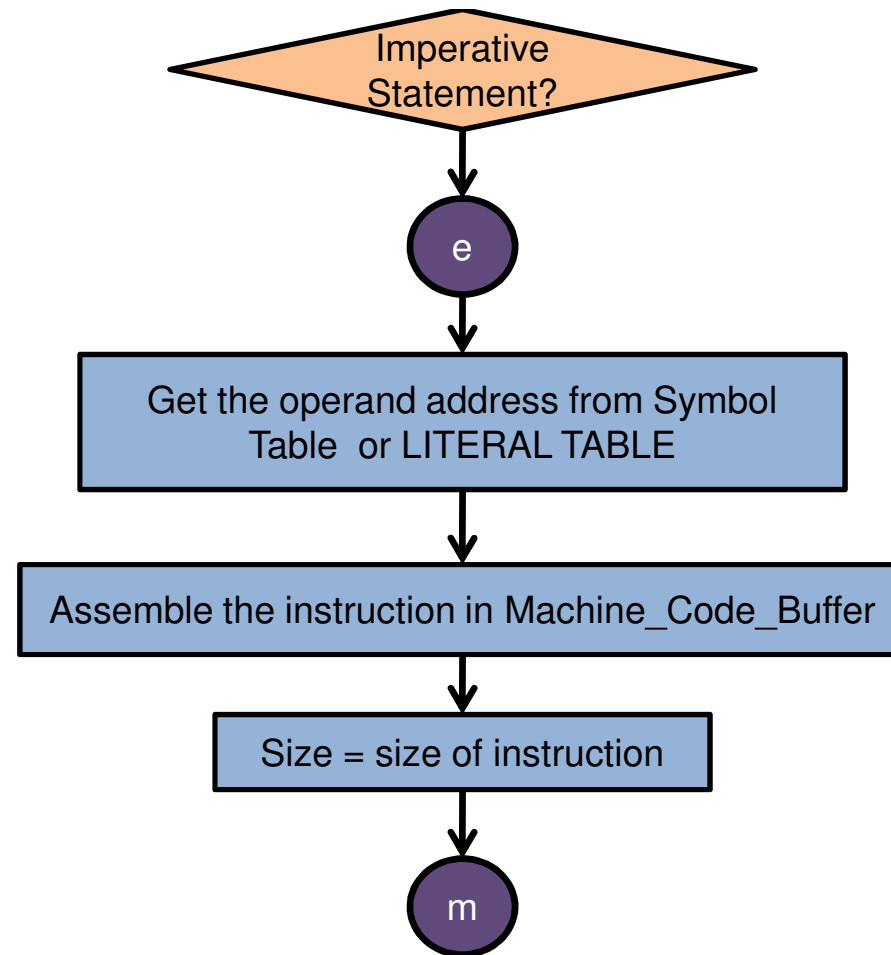


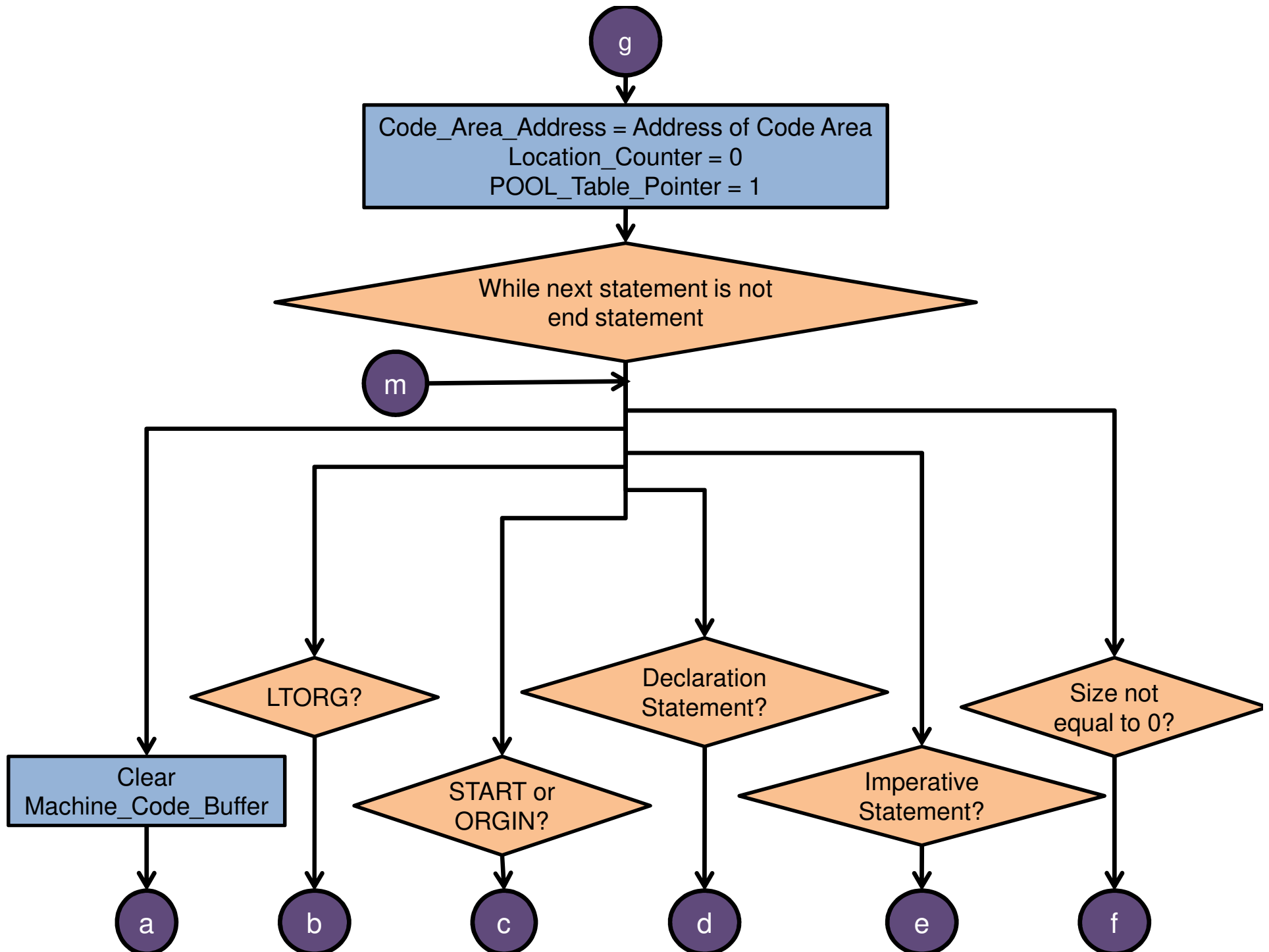


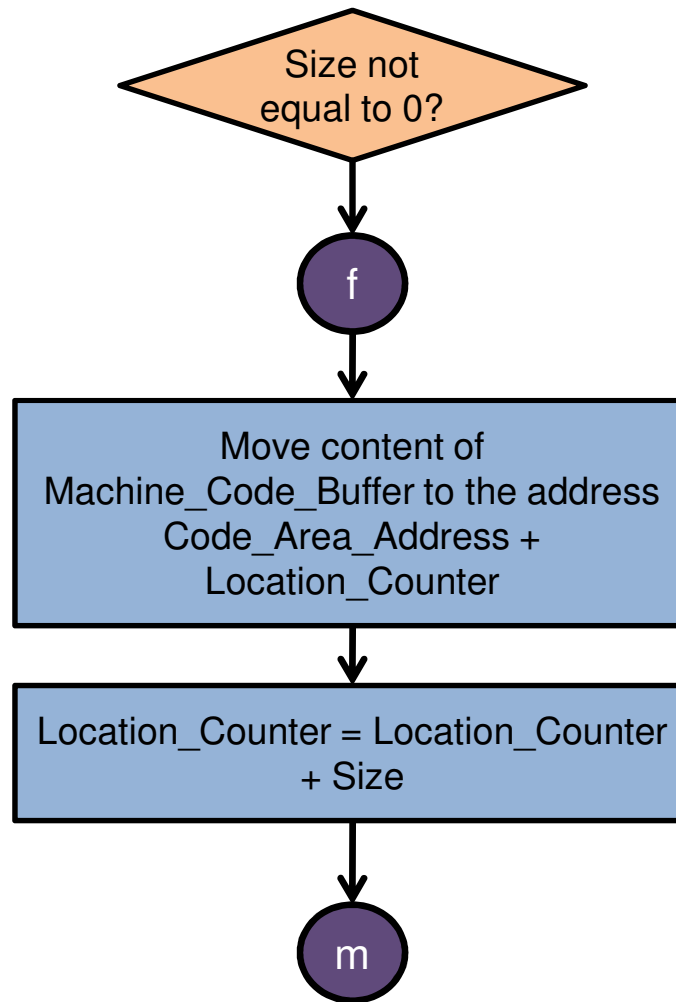


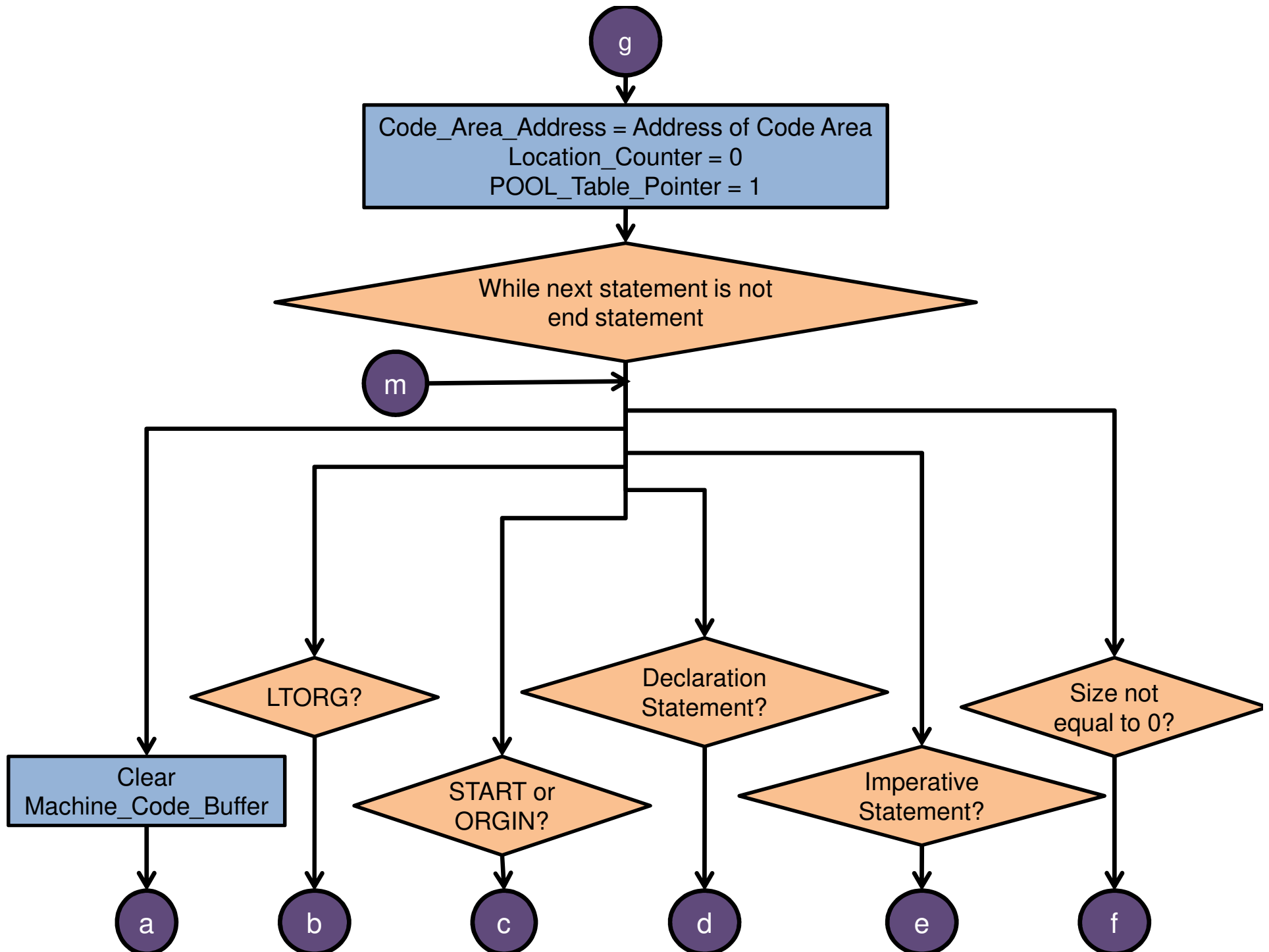


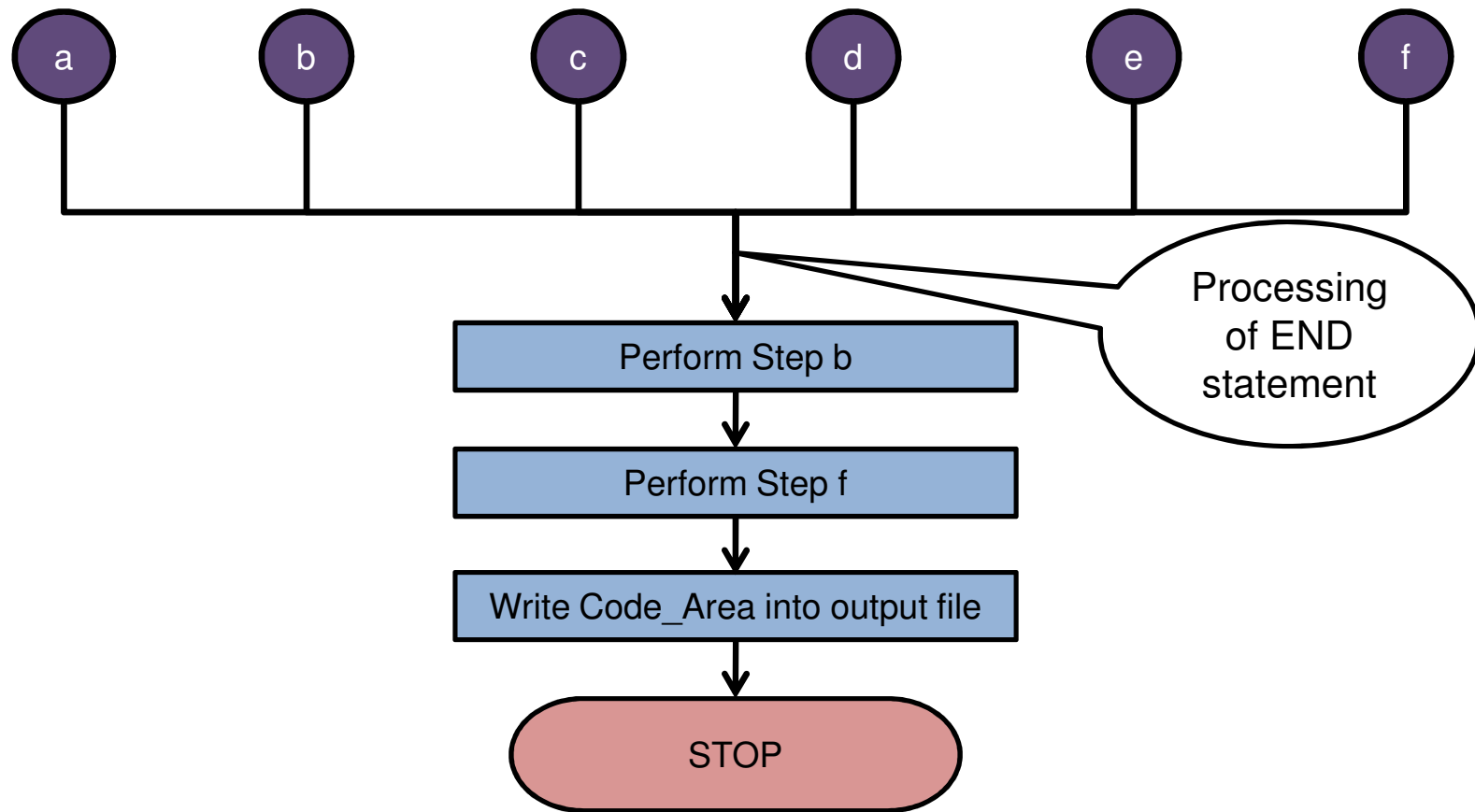






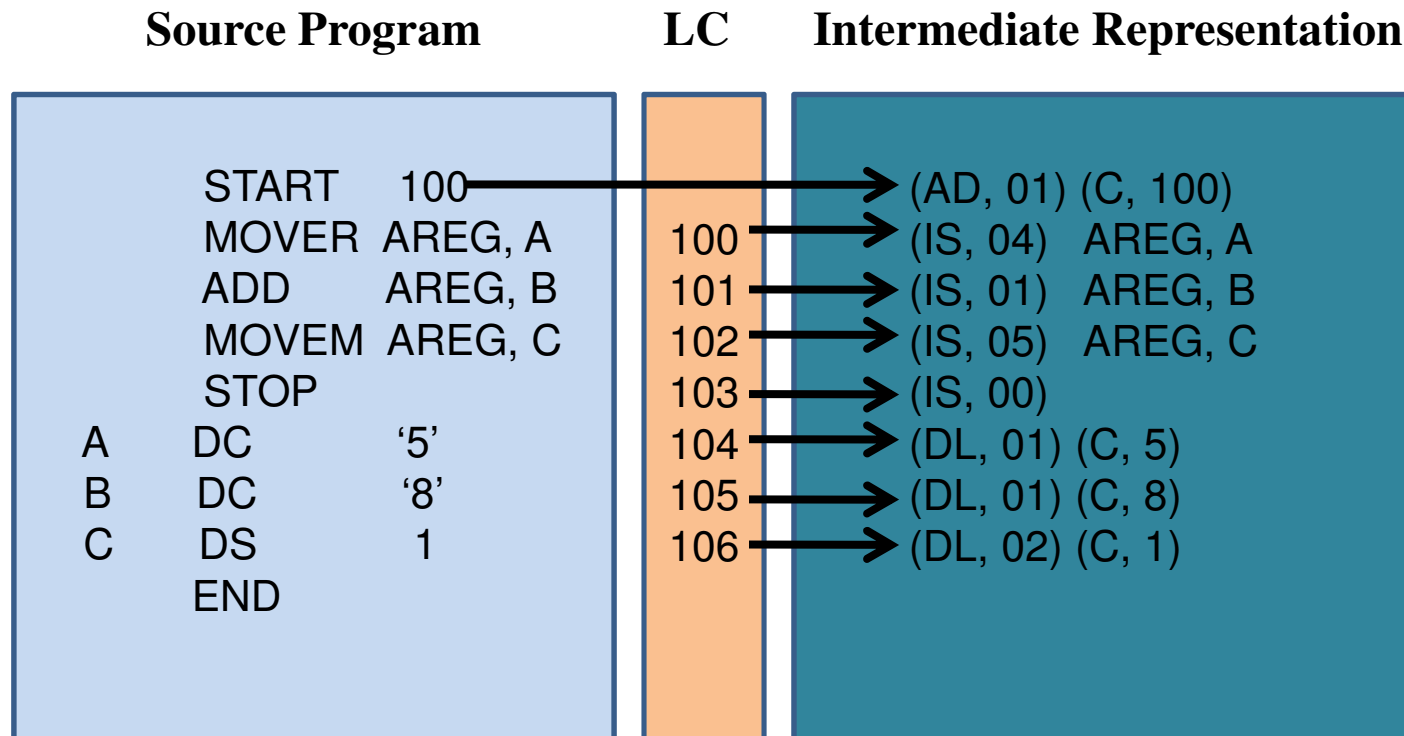






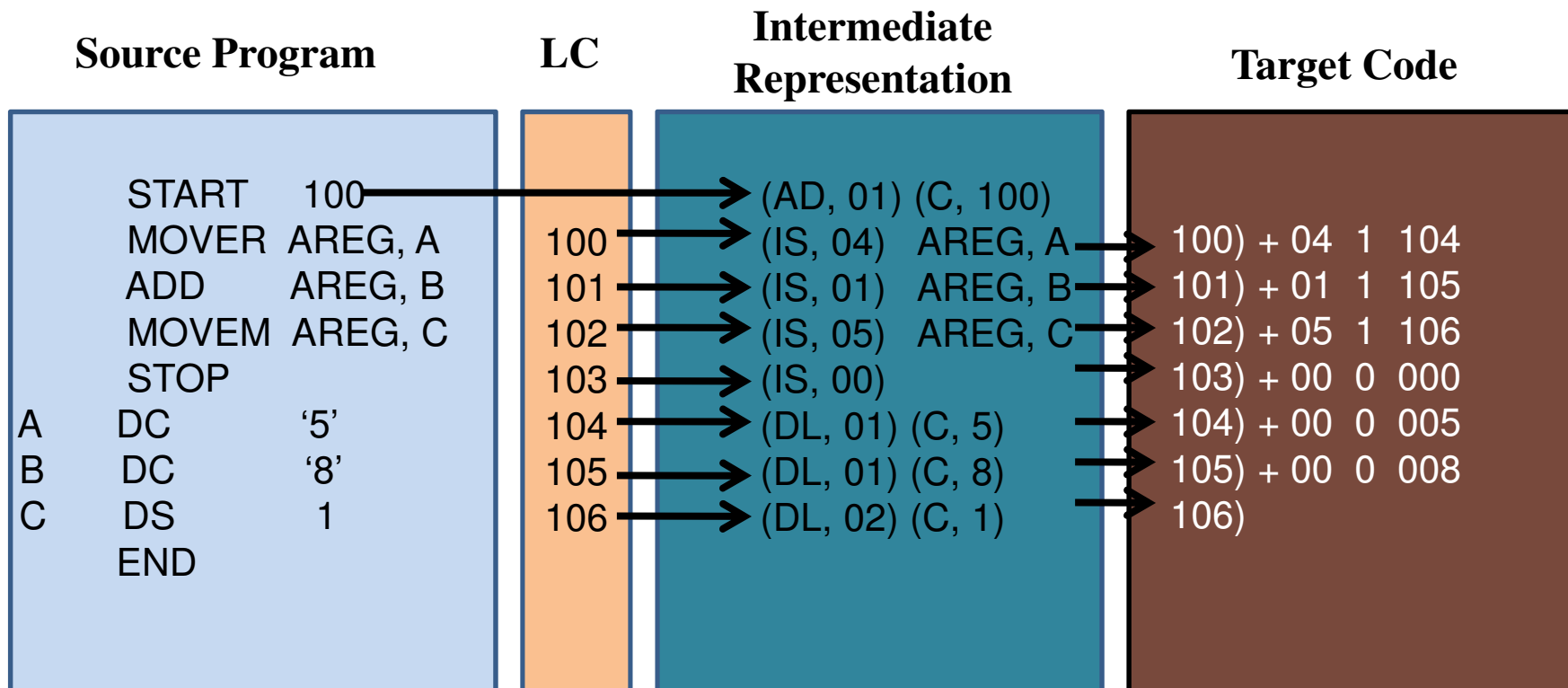
Source Program

```
START 100  
MOVER AREG, A  
ADD AREG, B  
MOVEM AREG, C  
STOP  
A DC '5'  
B DC '8'  
C DS 1  
END
```



Symbol	Address	Length
A	104	1
B	105	1
C	106	1

Symbol Table



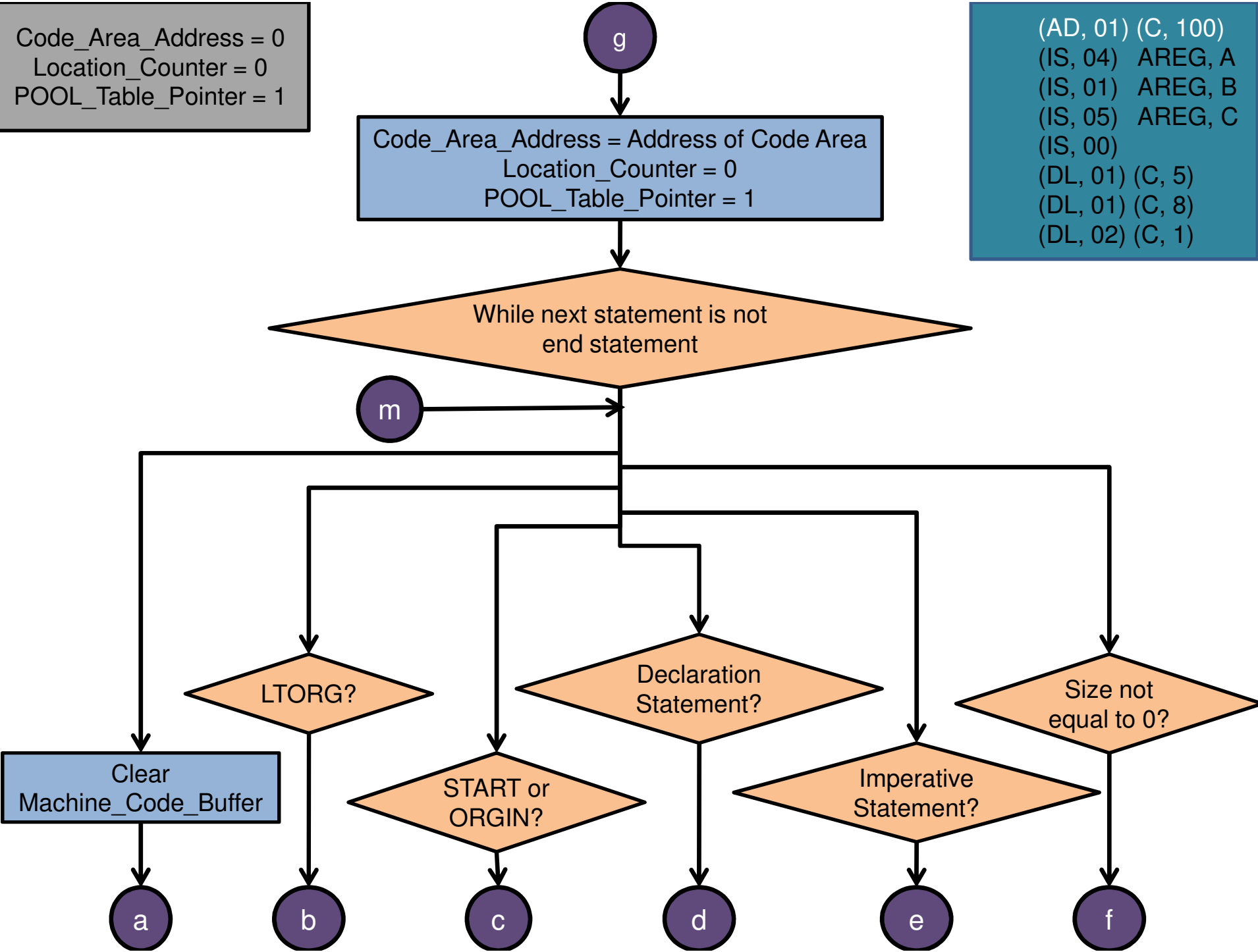
Symbol	Address	Length
A	104	1
B	105	1
C	106	1

Symbol Table

Code_Area_Address = 0
Location_Counter = 0
POOL_Table_Pointer = 1

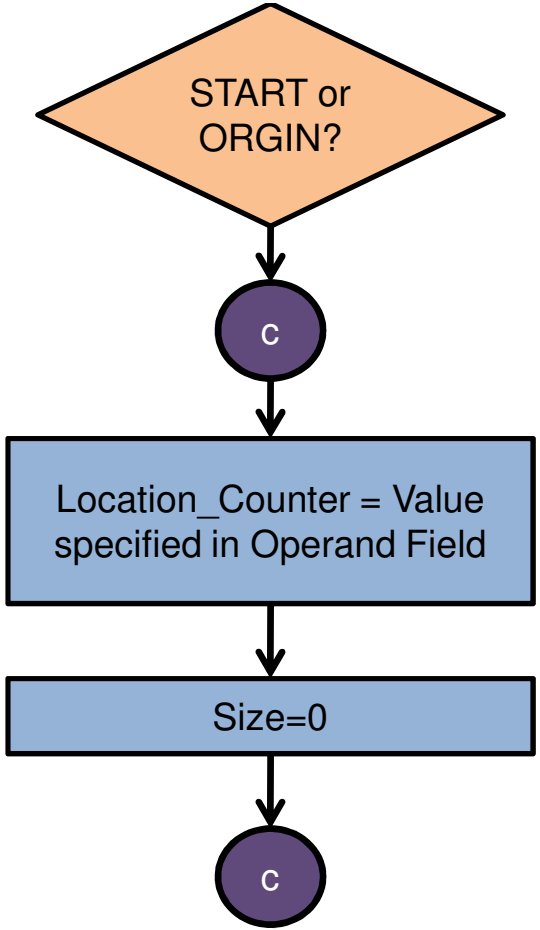
Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
Location_Counter = 0
POOL_Table_Ponter = 1
Size=0

Location_Counter
=100

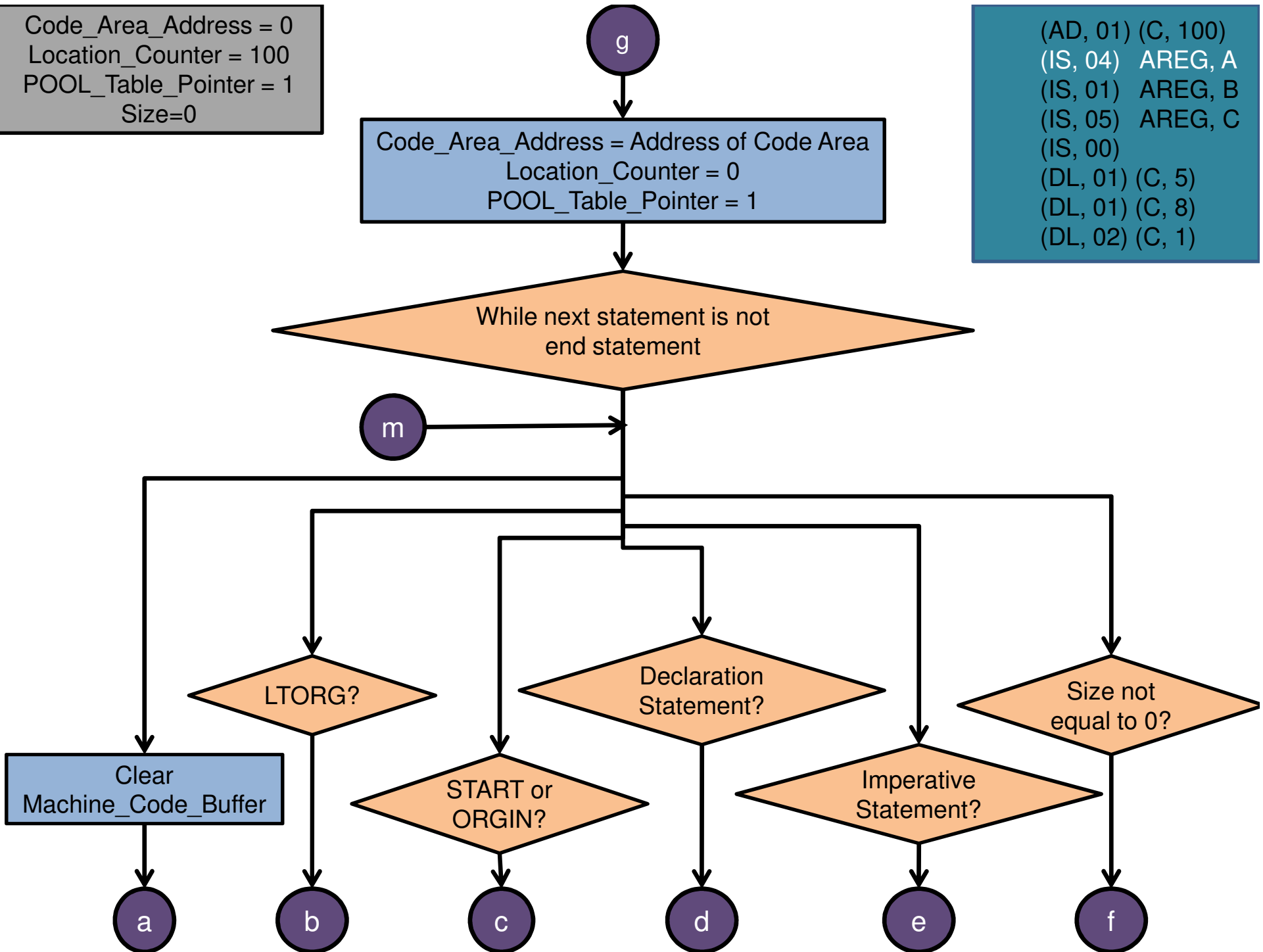


(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

Code_Area_Address = 0
Location_Counter = 100
POOL_Table_Pointer = 1
Size=0

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
 Location_Counter = 100
 POOL_Table_Pointer = 1
 Size=0

100) + 04 1 104 **MCB**

Imperative
Statement?

e

(AD, 01) (C, 100)
 (IS, 04) AREG, A
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 5)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Address of A =
104

Get the operand address from Symbol
Table or LITERAL TABLE

Assemble the instruction in Machine_Code_Buffer
(MCB)

100) + 04 1 104

Size = 1

Size = size of instruction

m

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015
OP Table

Register	Code
AREG	1
BREG	2
CREG	3
DREG	4

Symbol	Address	Length
A	104	1
B	105	1
C	106	1

Symbol Table

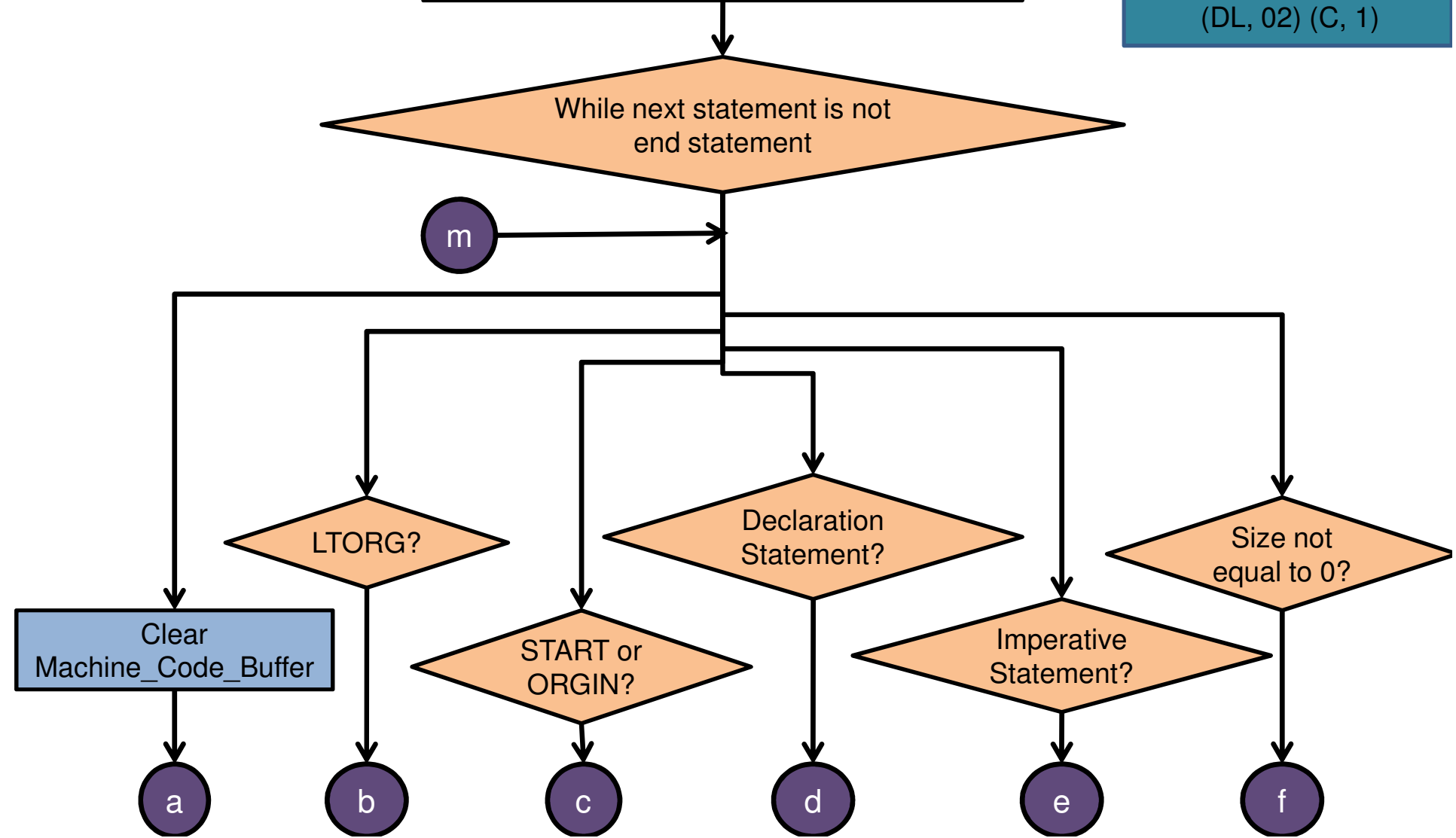
Code_Area_Address = 0
Location_Counter = 100
POOL_Table_Pointer = 1
Size=1

100) + 04 1 104 **MCB**

g

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

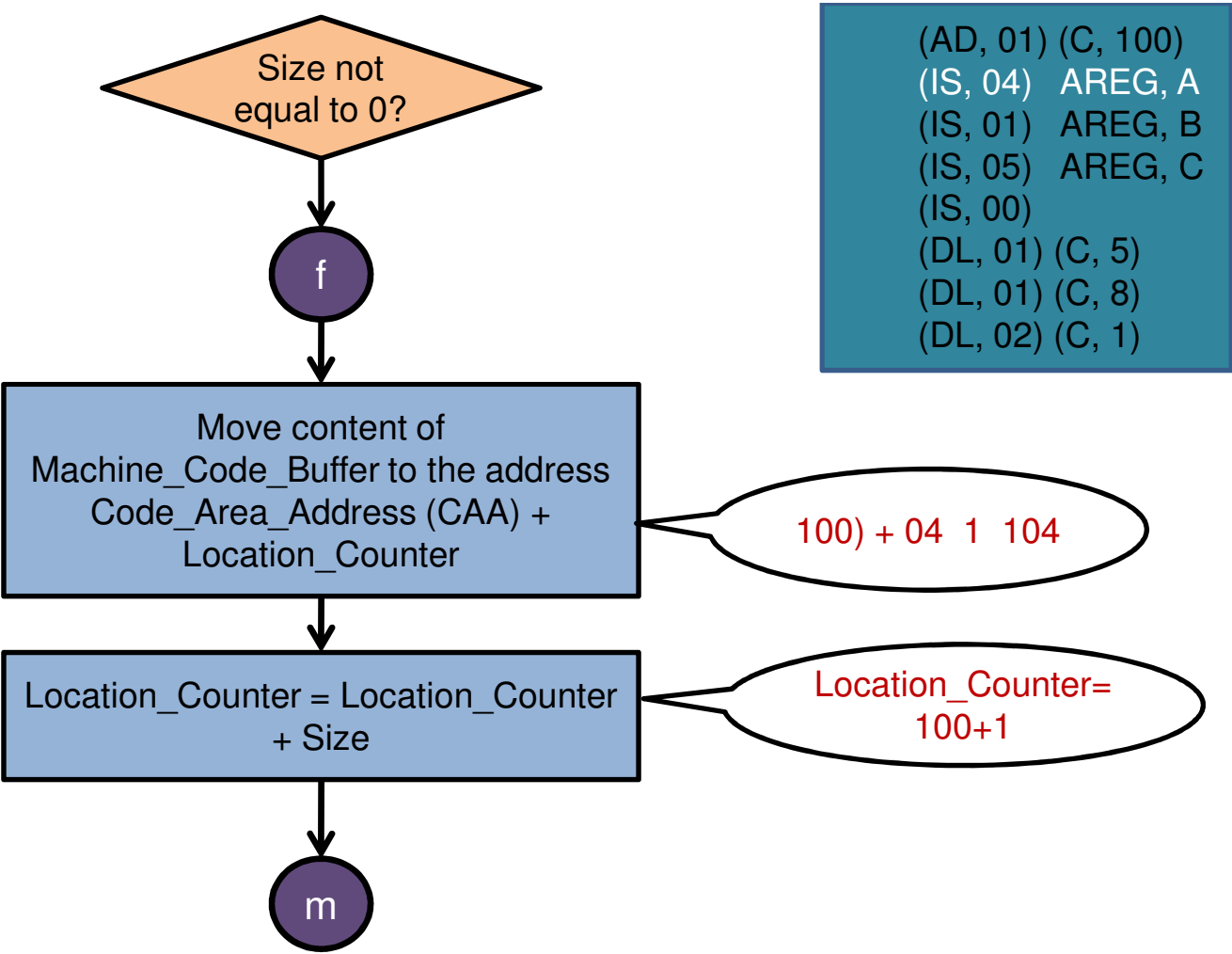
(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
Location_Counter = 100
POOL_Table_Pointer = 1
Size=1

100) + 04 1 104	MCB
-----------------	------------

100) + 04 1 104	CAA
-----------------	------------

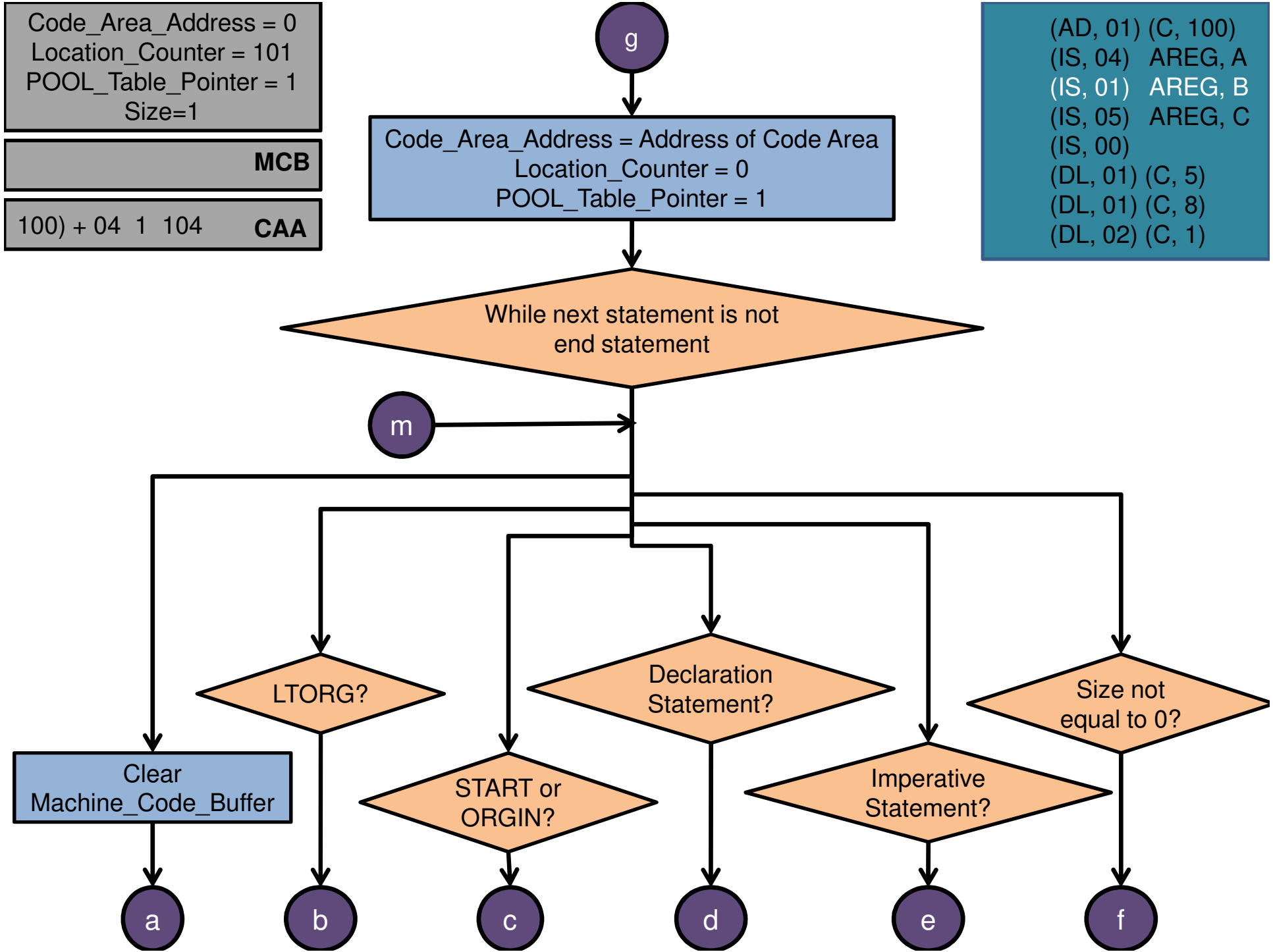


(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0 Location_Counter = 101 POOL_Table_Pointer = 1 Size=1	
	MCB
100) + 04 1 104	CAA

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
 Location_Counter = 101
 POOL_Table_Pointer = 1
 Size=1

101) + 01 1 105 **MCB**

100) + 04 1 104 **CAA**

Imperative
Statement?

e

Get the operand address from Symbol
Table or LITERAL TABLE

Address of B =
105

(AD, 01) (C, 100)
 (IS, 04) AREG, A
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 5)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Assemble the instruction in Machine_Code_Buffer

101) + 01 1 105

Size = 1

Size = size of instruction

m

Symbol	Address	Length
A	104	1
B	105	1
C	106	1

Symbol Table

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

OP Table

Register	Code
AREG	1
BREG	2
CREG	3
DREG	4

Code_Area_Address = 0	
Location_Counter = 101	
POOL_Table_Pointer = 1	
Size=1	

101) + 01 1 105	MCB
-----------------	------------

100) + 04 1 104	CAA
-----------------	------------

g

Code_Area_Address = Address of Code Area
Location_Counter =0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)

(IS, 04) AREG, A

(IS, 01) AREG, B

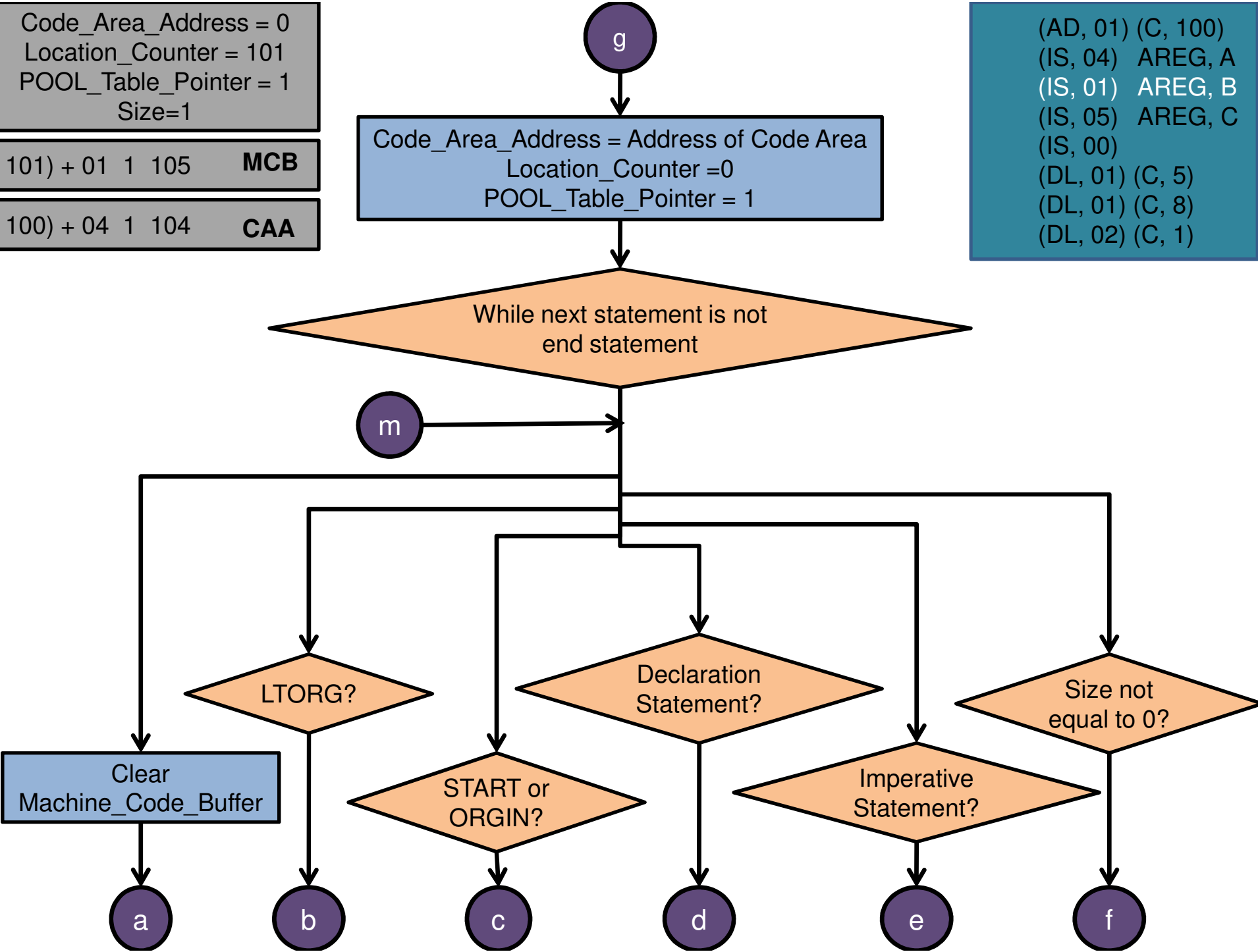
(IS, 05) AREG, C

(IS, 00)

(DL, 01) (C, 5)

(DL, 01) (C, 8)

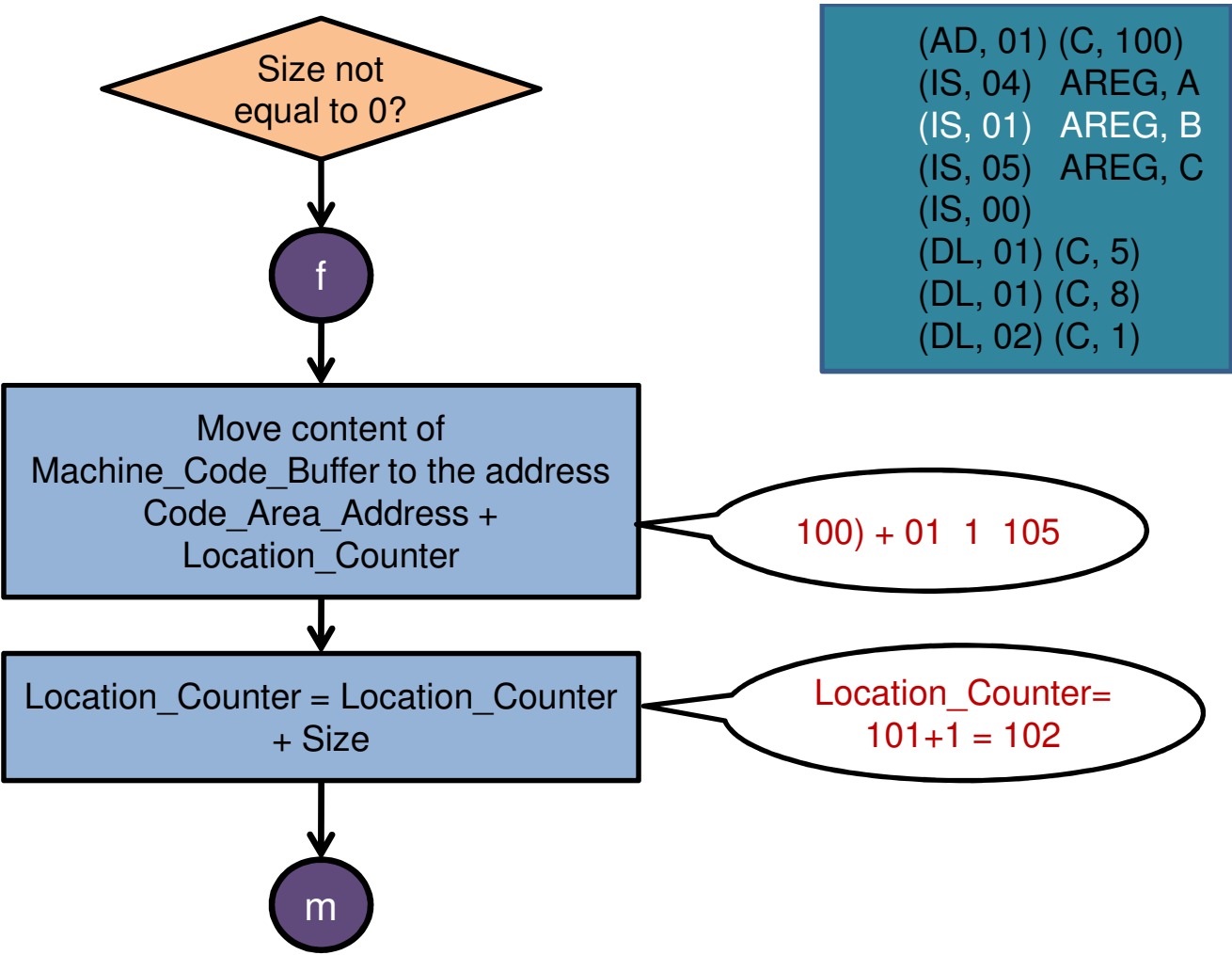
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 101	
POOL_Table_Pointer = 1	
Size=1	

101) + 01 1 105	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	

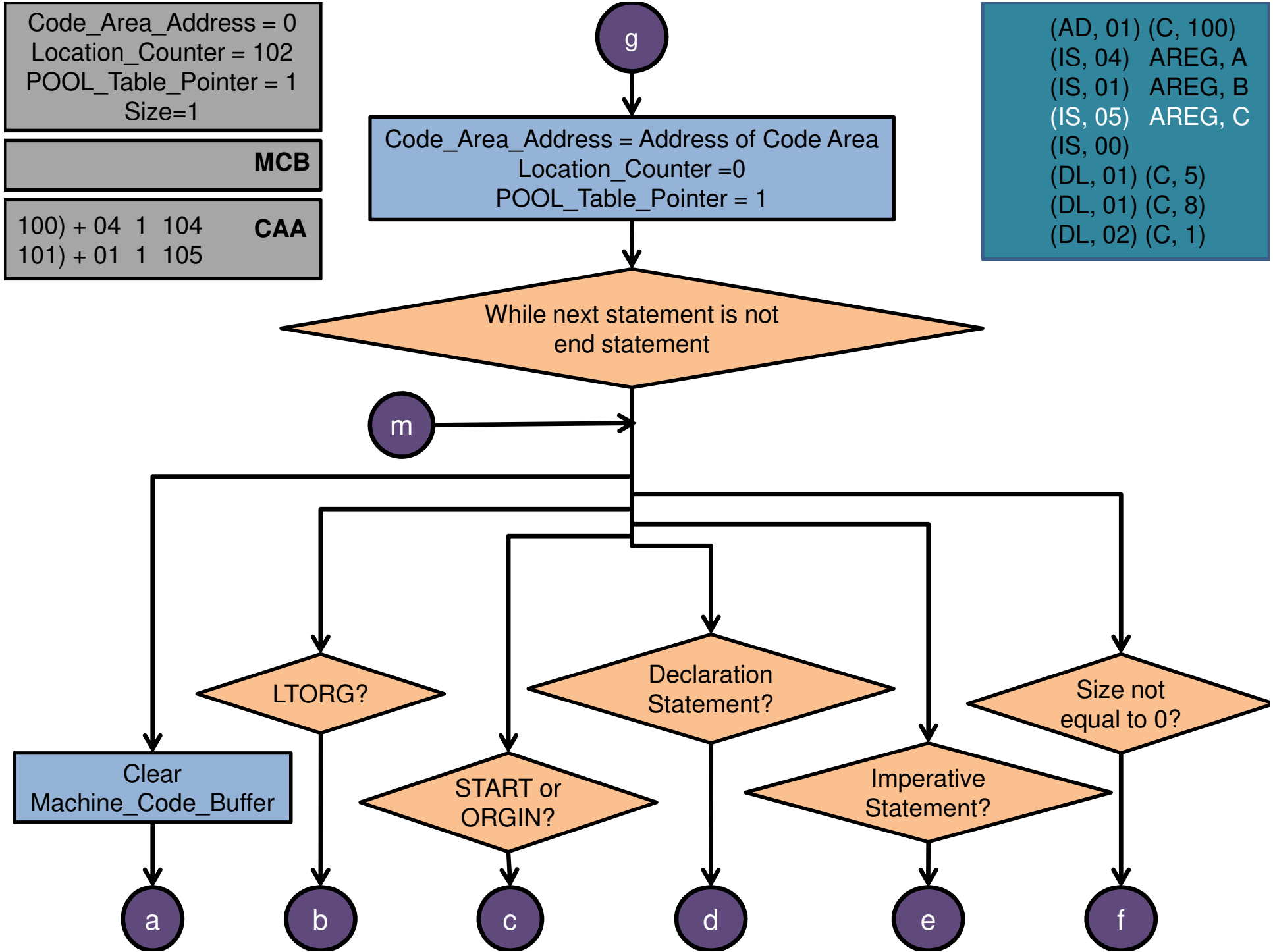


(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0 Location_Counter = 102 POOL_Table_Pointer = 1 Size=1	
	MCB
100) + 04 1 104 101) + 01 1 105	CAA

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
 Location_Counter = 102
 POOL_Table_Pointer = 1
 Size=1

102) + 05 1 106 **MCB**

100) + 04 1 104 **CAA**
 101) + 01 1 105

Imperative
Statement?

e

Get the operand address from Symbol
Table or LITERAL TABLE

Address of B =
106

Assemble the instruction in Machine_Code_Buffer

102) + 05 1 106

Size = 1

Size = size of instruction

m

(AD, 01) (C, 100)
 (IS, 04) AREG, A
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 5)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

OP Table

Register	Code
AREG	1
BREG	2
CREG	3
DREG	4

Mrs. Sunita M Dol, CSE Dept

Symbol	Address	Length
A	104	1
B	105	1
C	106	1

Symbol Table

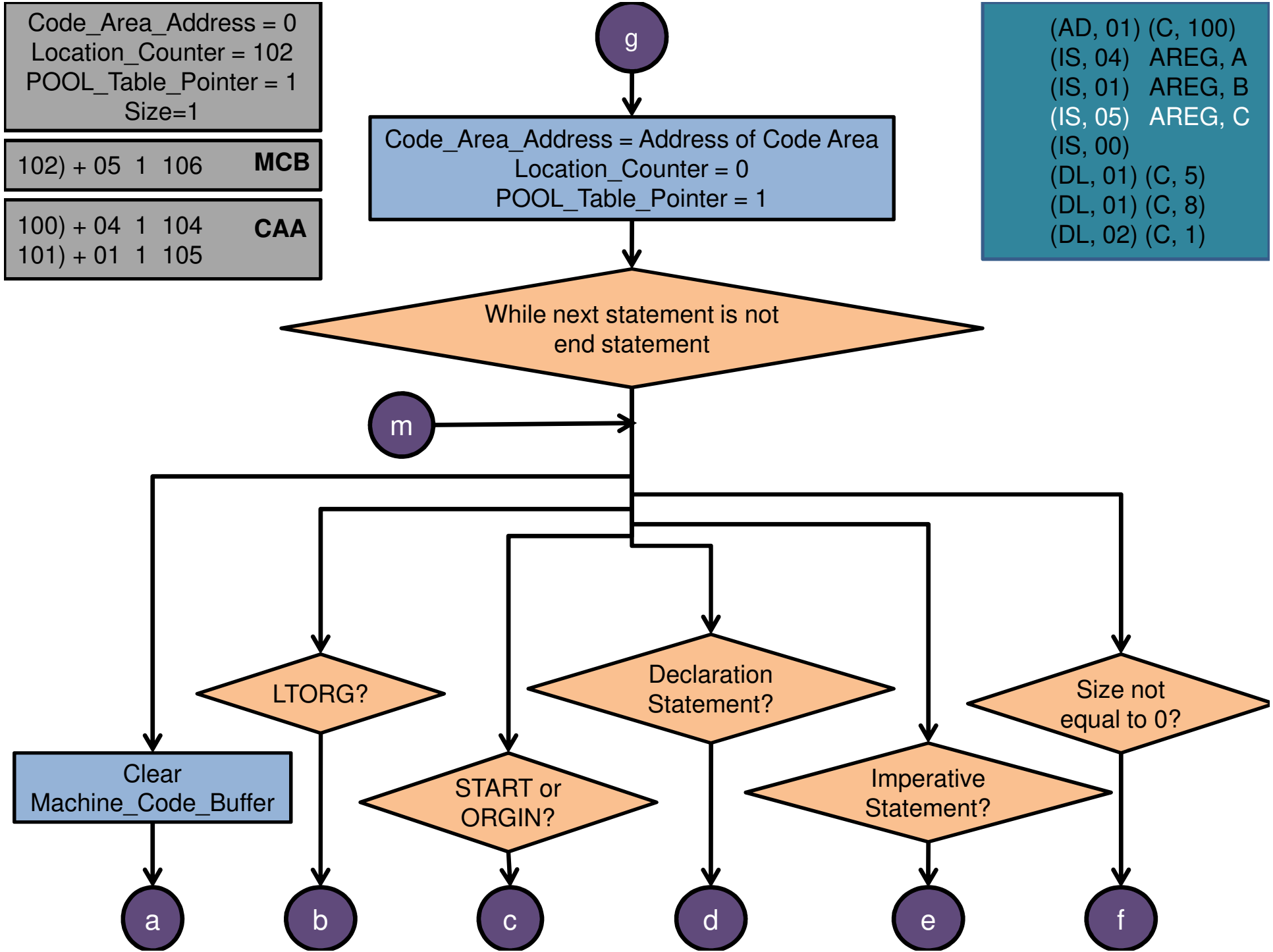
Code_Area_Address = 0	
Location_Counter = 102	
POOL_Table_Pointer = 1	
Size=1	

102) + 05 1 106	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

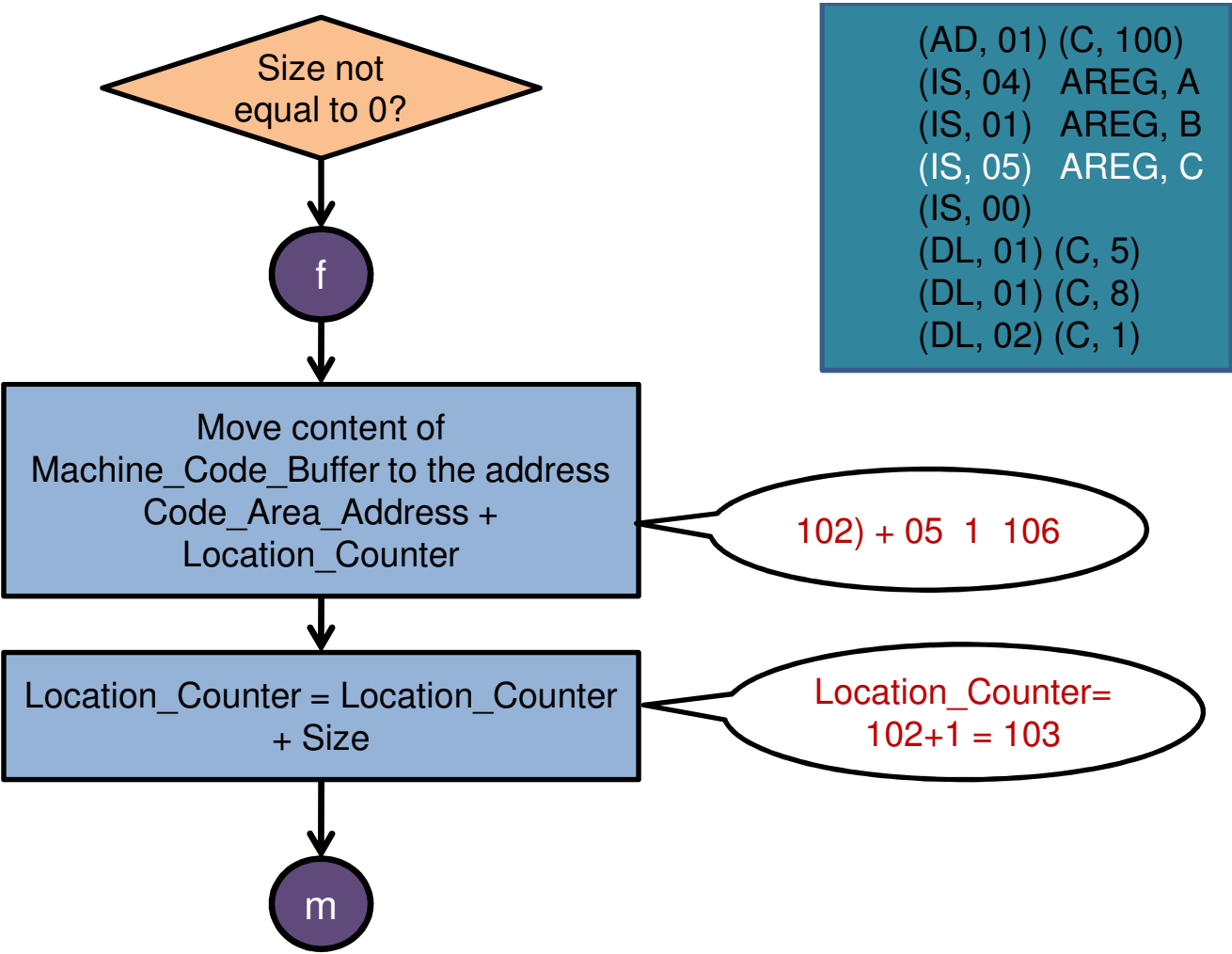
(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 102	
POOL_Table_Pointer = 1	
Size=1	

102) + 05 1 106	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	



(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

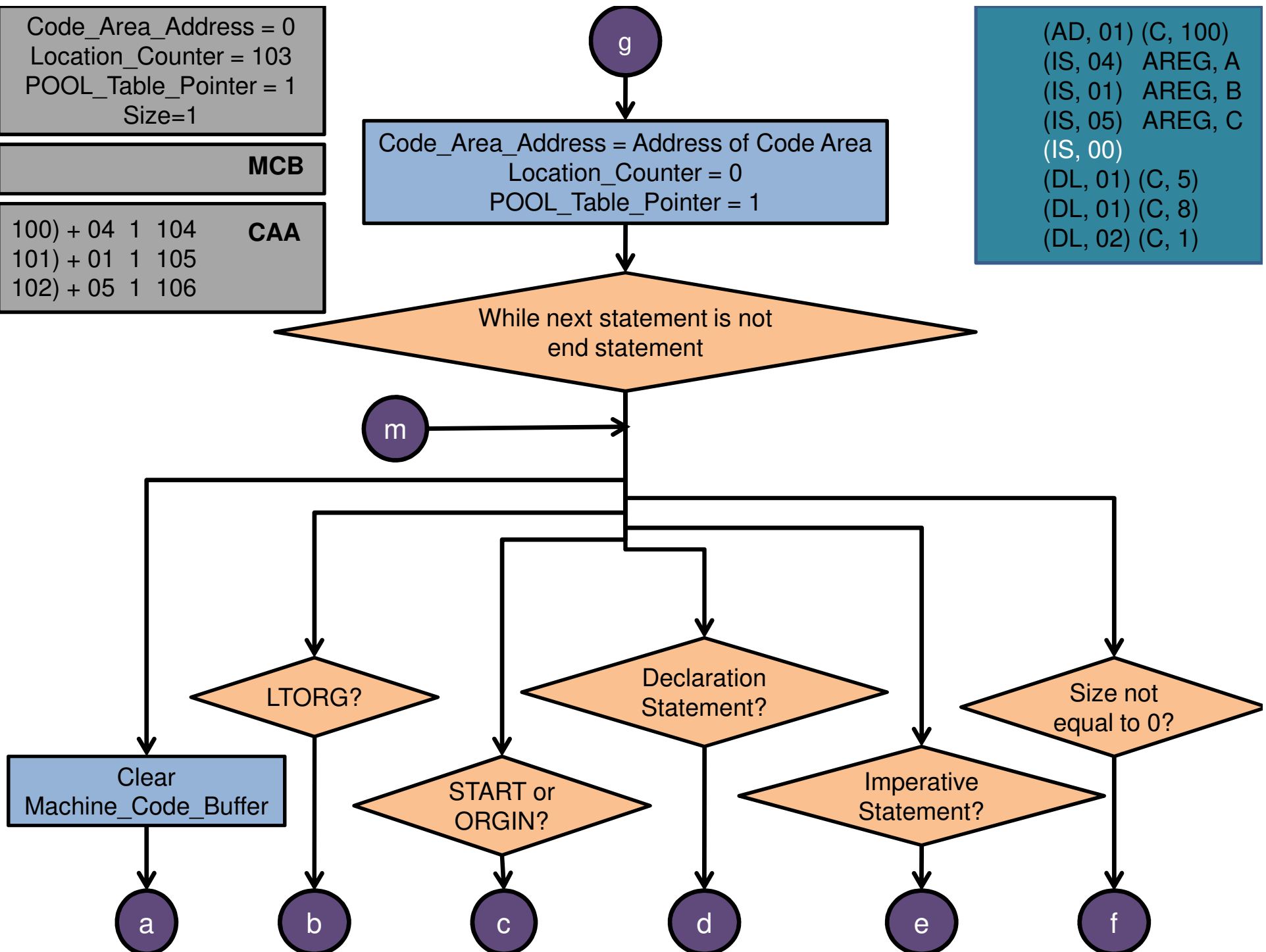
Code_Area_Address = 0
Location_Counter = 103
POOL_Table_Pointer = 1
Size=1

MCB

100) + 04 1 104 **CAA**
101) + 01 1 105
102) + 05 1 106

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

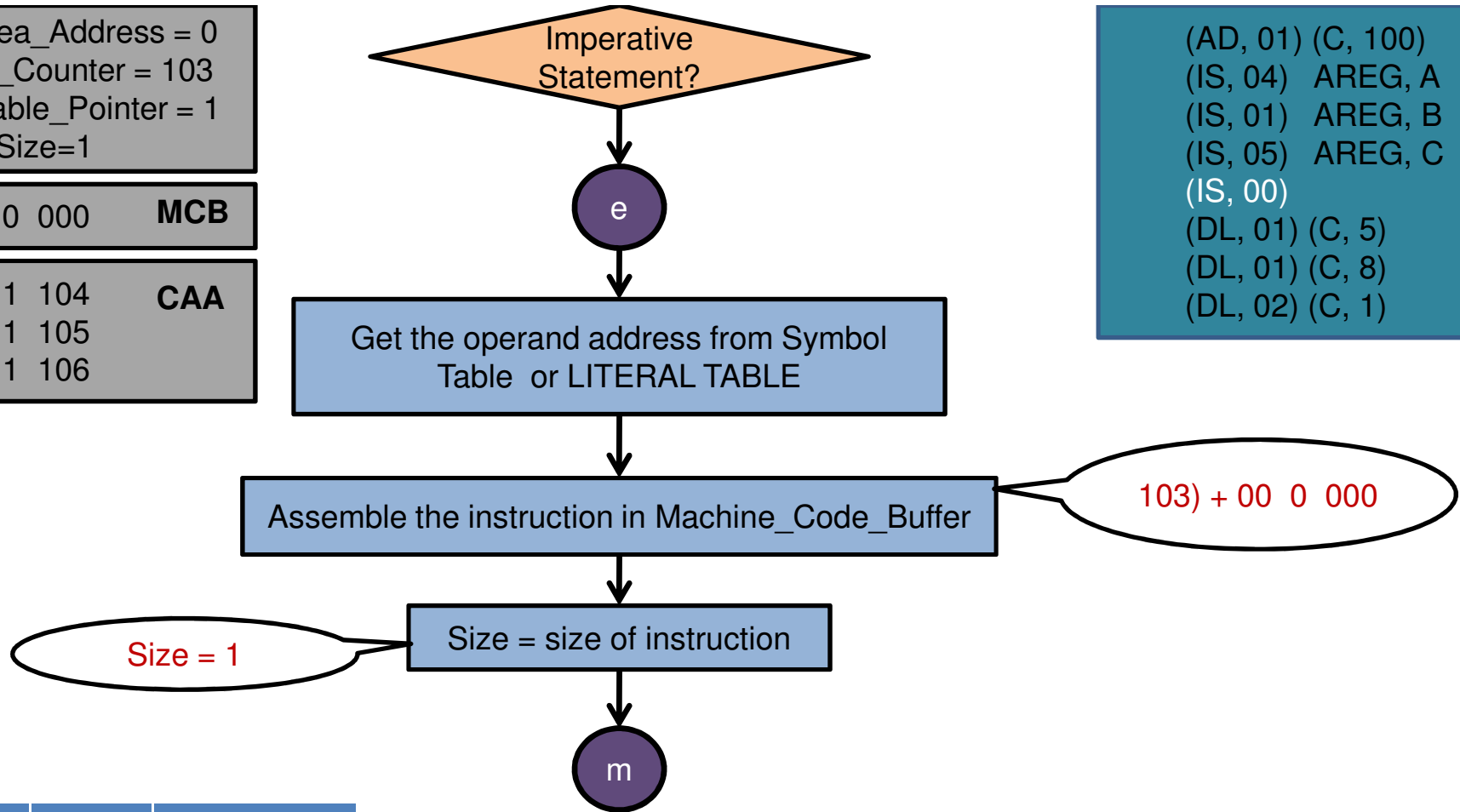


Code_Area_Address = 0
 Location_Counter = 103
 POOL_Table_Pointer = 1
 Size=1

103) + 00 0 000 **MCB**

100) + 04 1 104 **CAA**
 101) + 01 1 105
 102) + 05 1 106

(AD, 01) (C, 100)
 (IS, 04) AREG, A
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 5)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)



Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

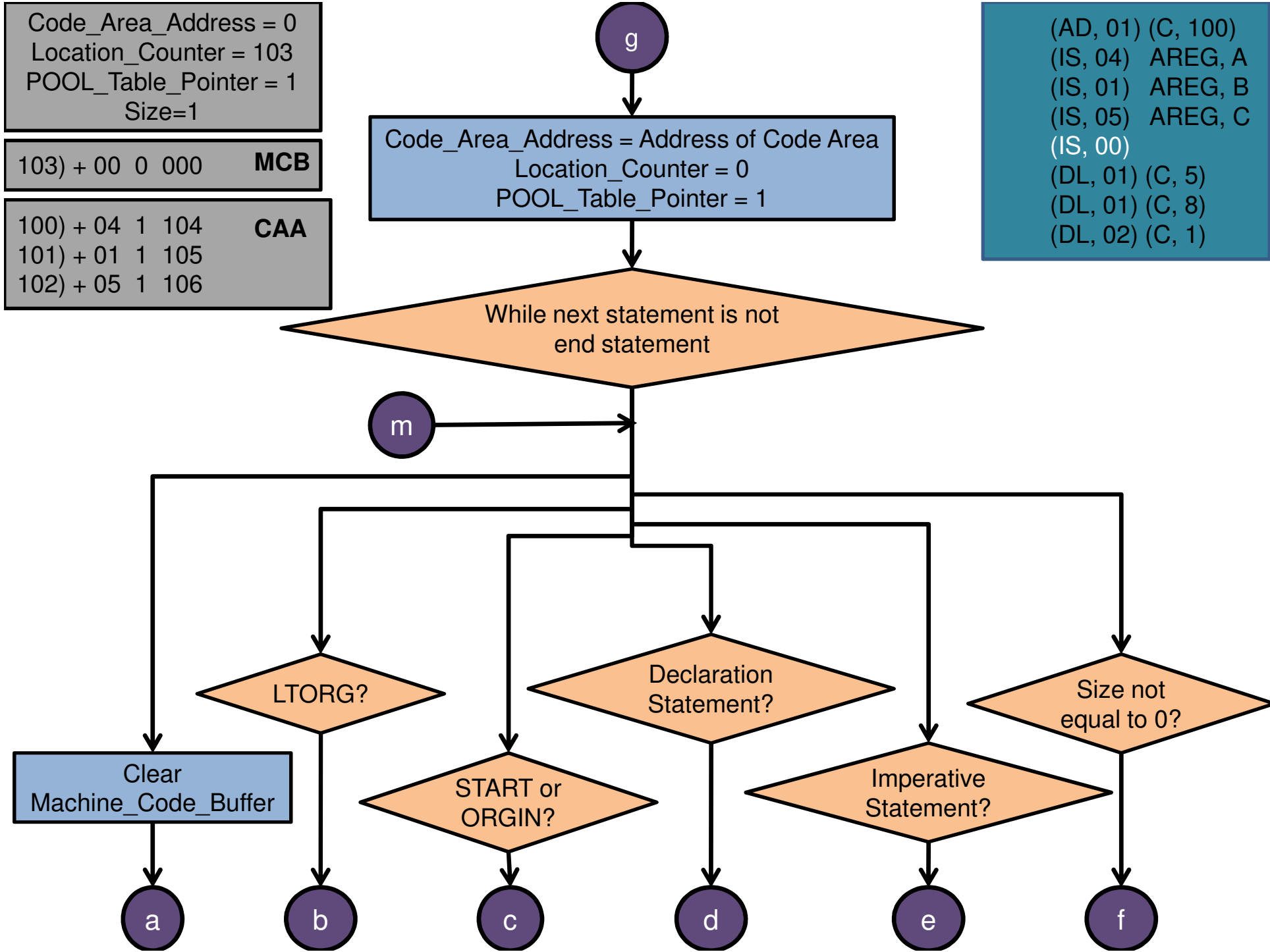
Code_Area_Address = 0
Location_Counter = 103
POOL_Table_Pointer = 1
Size=1

103) + 00 0 000 **MCB**

100) + 04 1 104 **CAA**
101) + 01 1 105
102) + 05 1 106

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

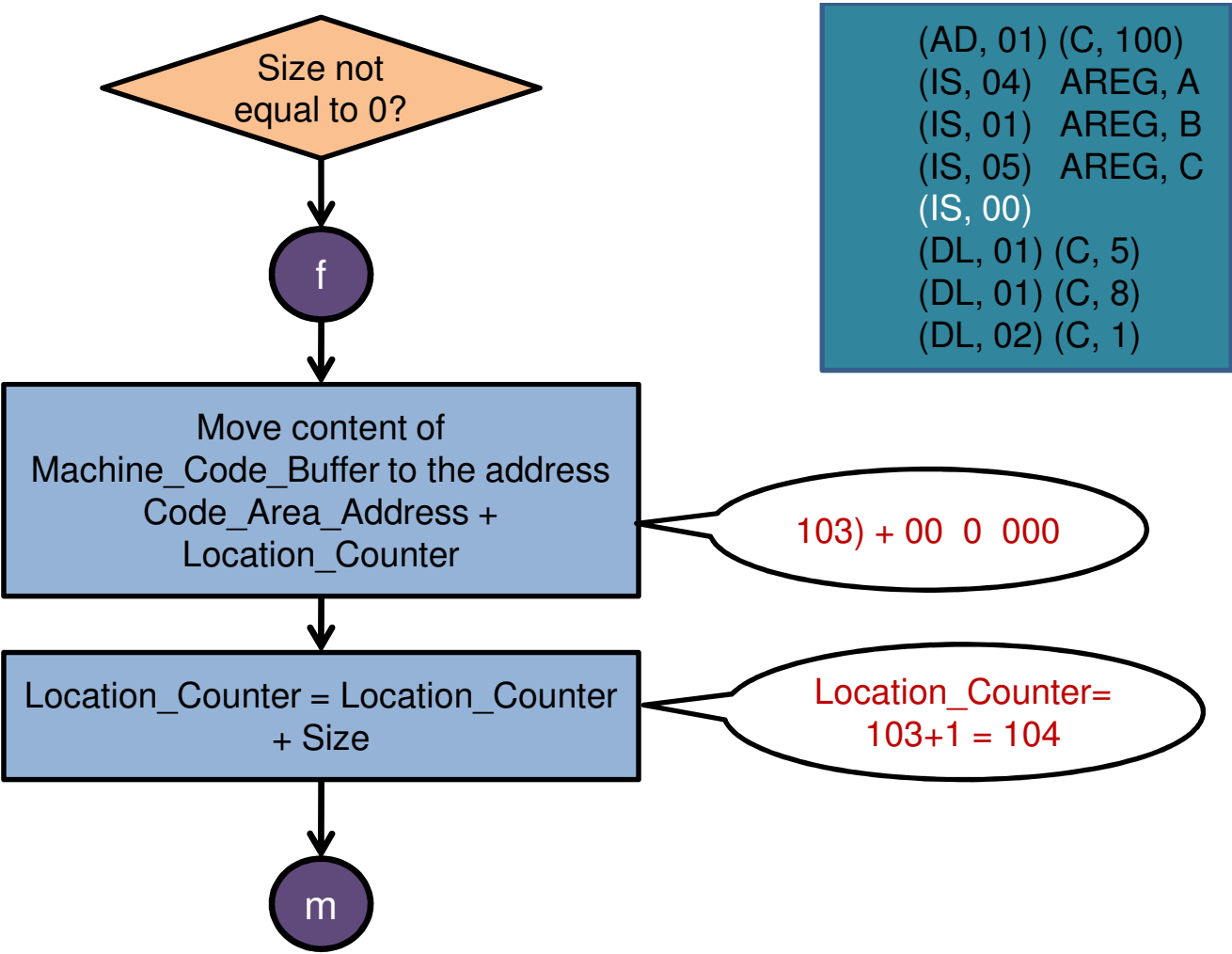
(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 103	
POOL_Table_Pointer = 1	
Size=1	

103) + 00 0 000	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	



(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0
Location_Counter = 104
POOL_Table_Pointer = 1
Size=1

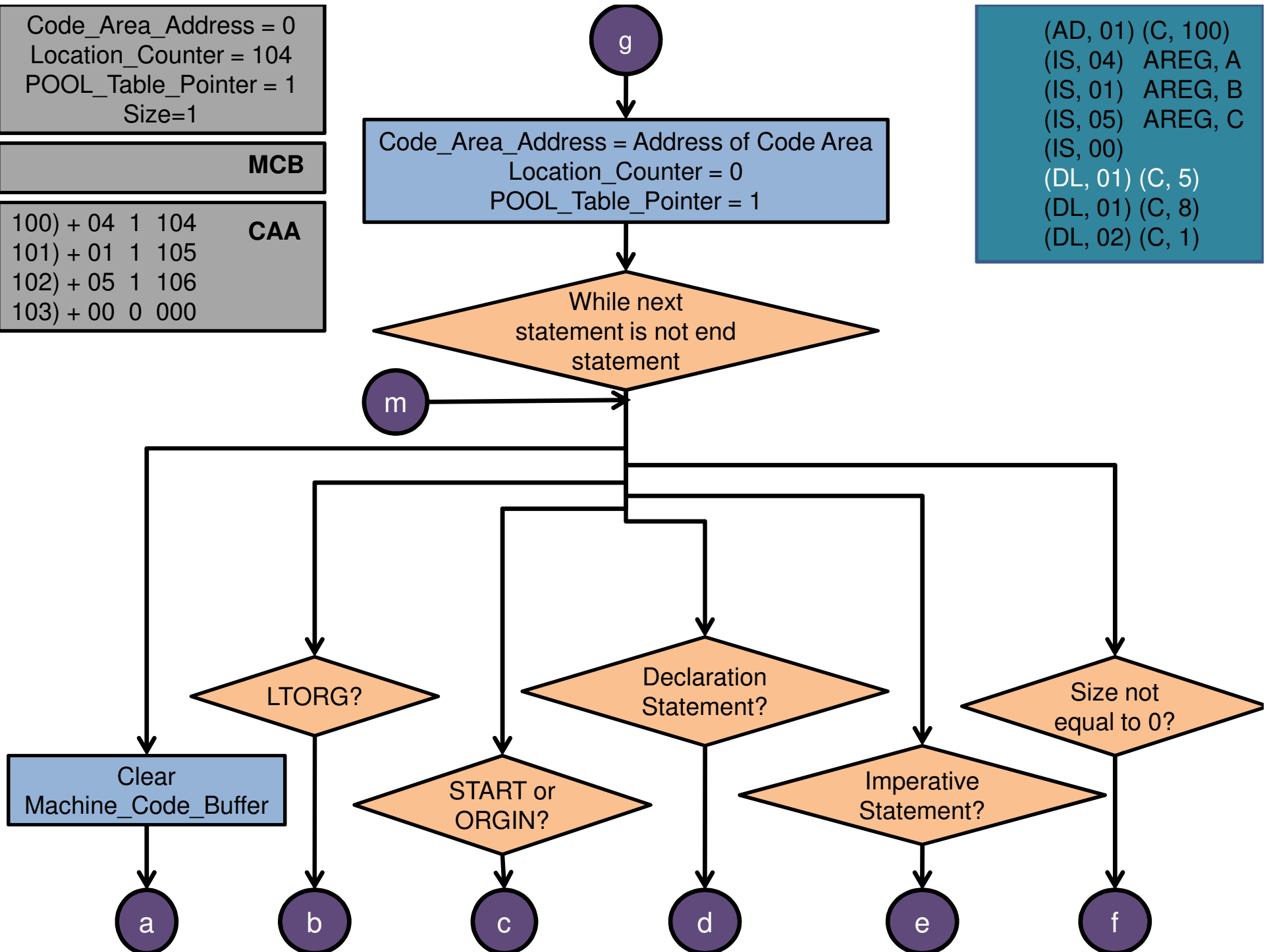
MCB

100) + 04 1 104
101) + 01 1 105
102) + 05 1 106
103) + 00 0 000

CAA

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

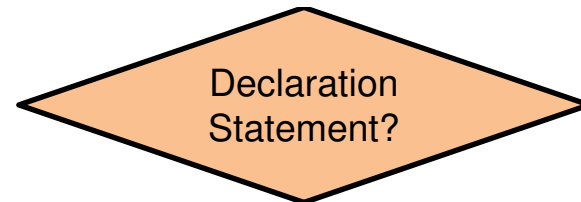
(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
Location_Counter = 104
POOL_Table_Pointer = 1
Size=1

104) + 00 0 005 **MCB**

100) + 04 1 104 **CAA**
101) + 01 1 105
102) + 05 1 106
103) + 00 0 000



e

Assemble the constant in
Machine_Code_Buffer for DC
Statement

104) + 00 0 005

Size = Size of memory area
required by DC/DS

Size = 1

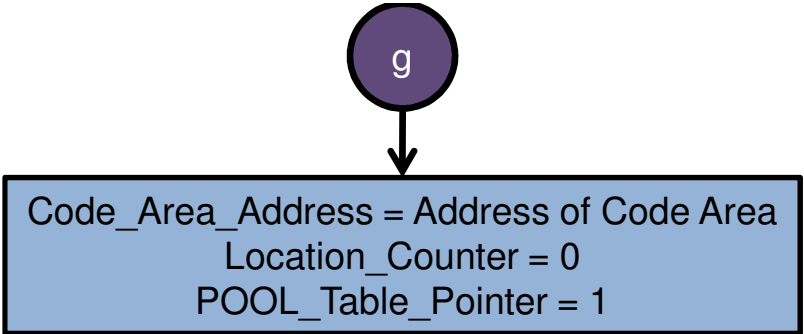
m

(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

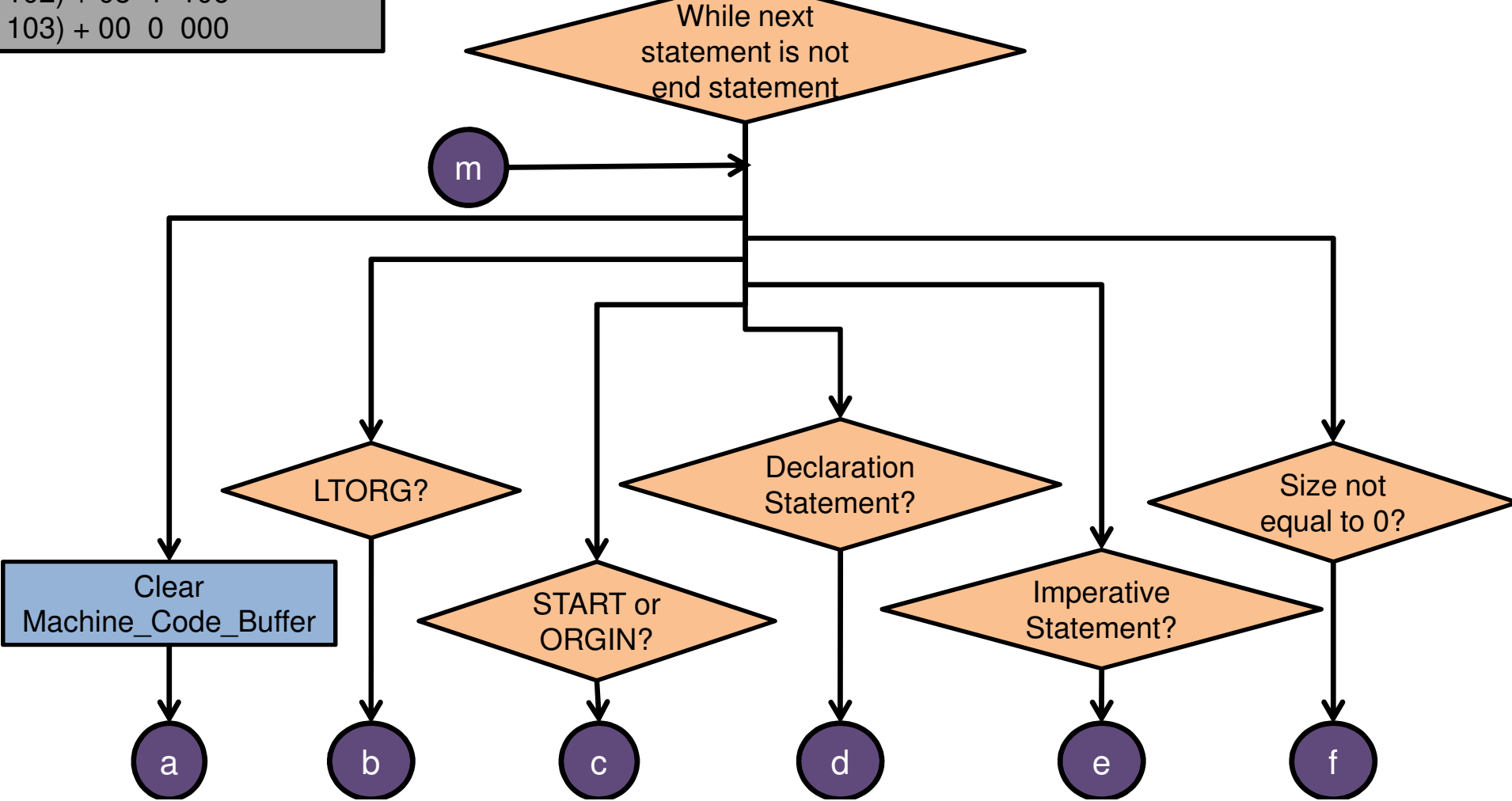
Code_Area_Address = 0	
Location_Counter = 104	
POOL_Table_Pointer = 1	
Size=1	

104) + 00 0 005	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	



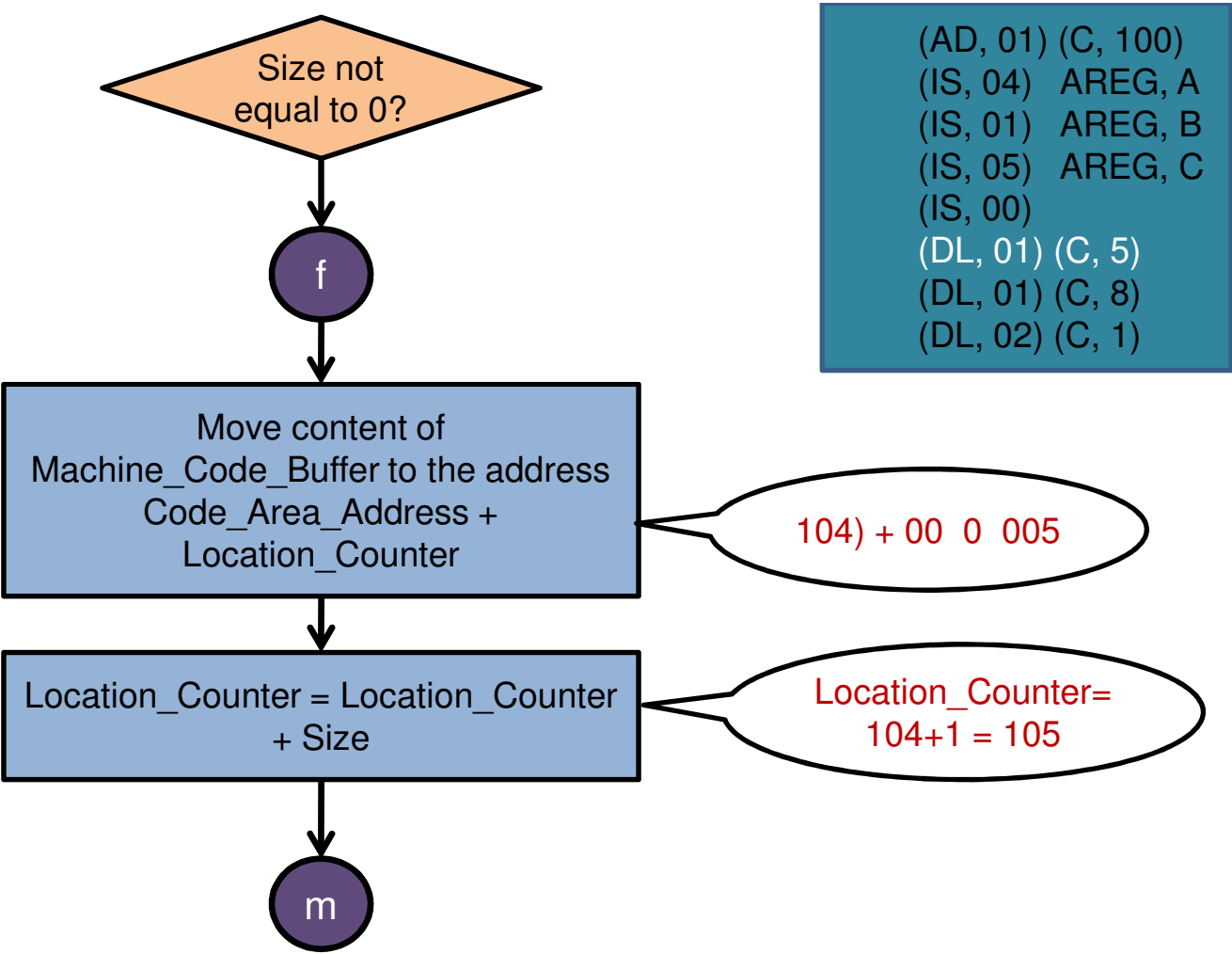
(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)



Code_Area_Address = 0	
Location_Counter = 104	
POOL_Table_Ponter = 1	
Size=1	

104) + 00 0 005	MCB
-----------------	------------

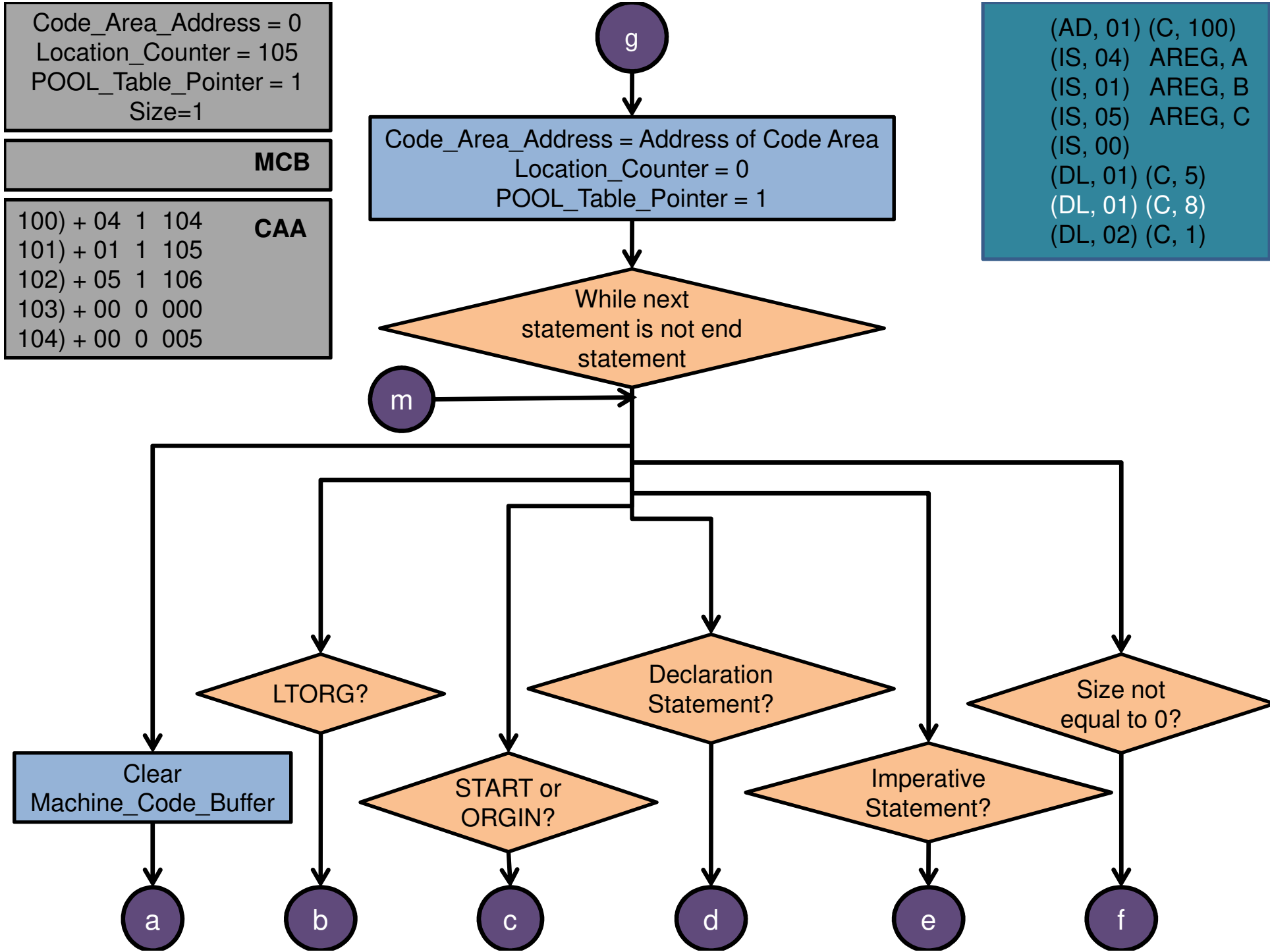
100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	
104) + 00 0 005	



(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0 Location_Counter = 105 POOL_Table_Pointer = 1 Size=1	
MCB	
100) + 04 1 104 101) + 01 1 105 102) + 05 1 106 103) + 00 0 000 104) + 00 0 005	CAA

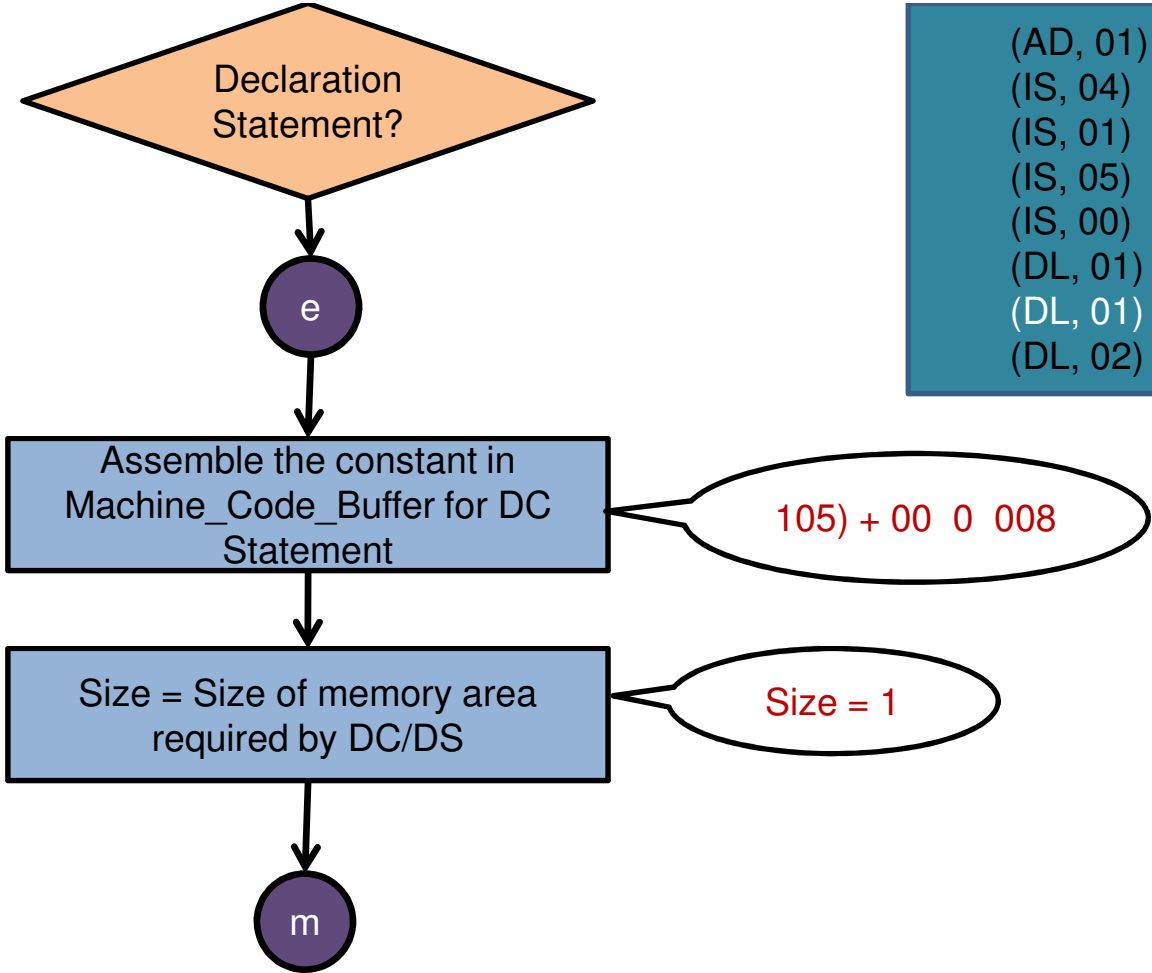
(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
Location_Counter = 105
POOL_Table_Pointer = 1
Size=1

105) + 00 0 008	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	
104) + 00 0 005	



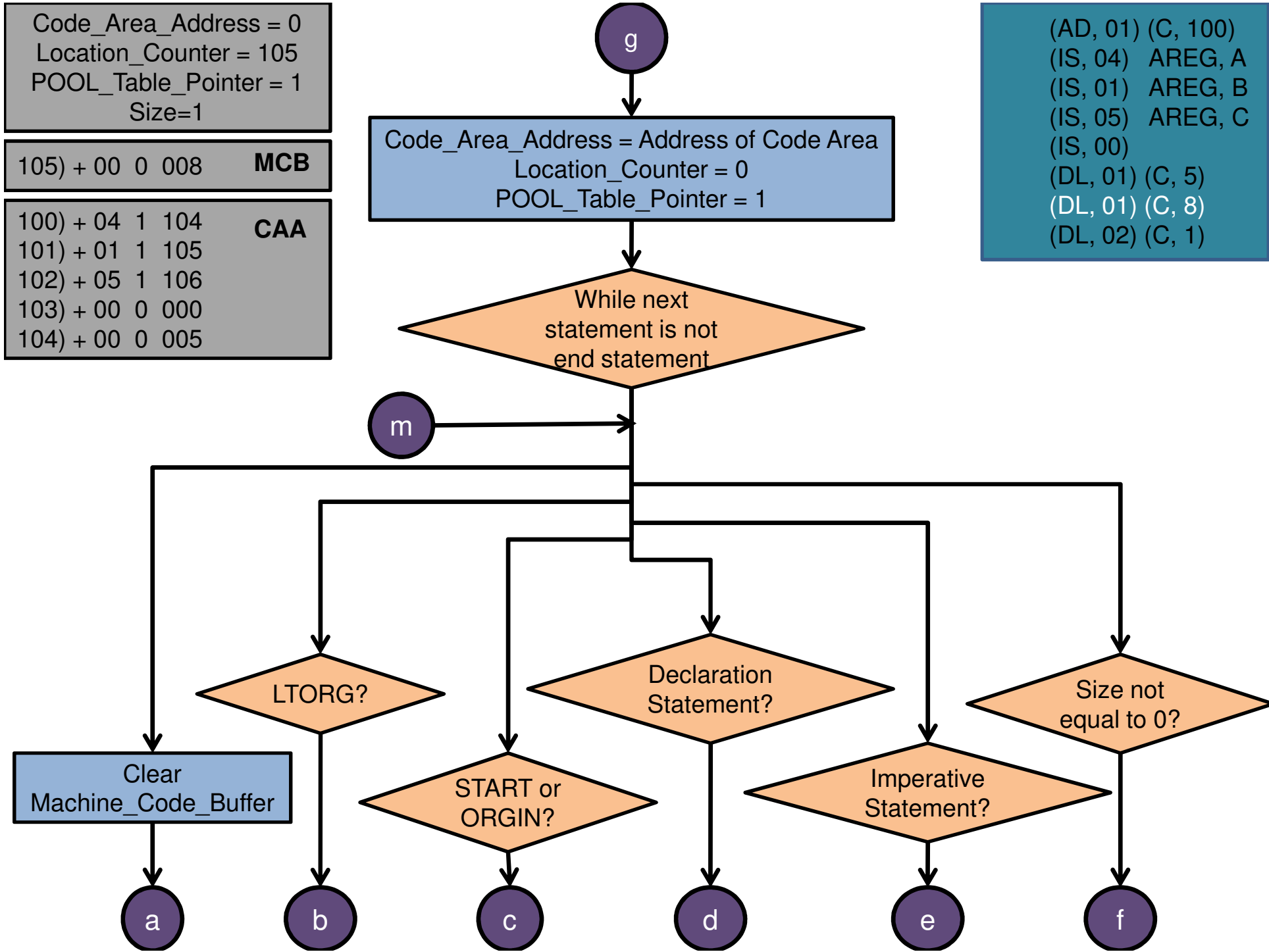
(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0	
Location_Counter = 105	
POOL_Table_Pointer = 1	
Size=1	

105) + 00 0 008	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	
104) + 00 0 005	

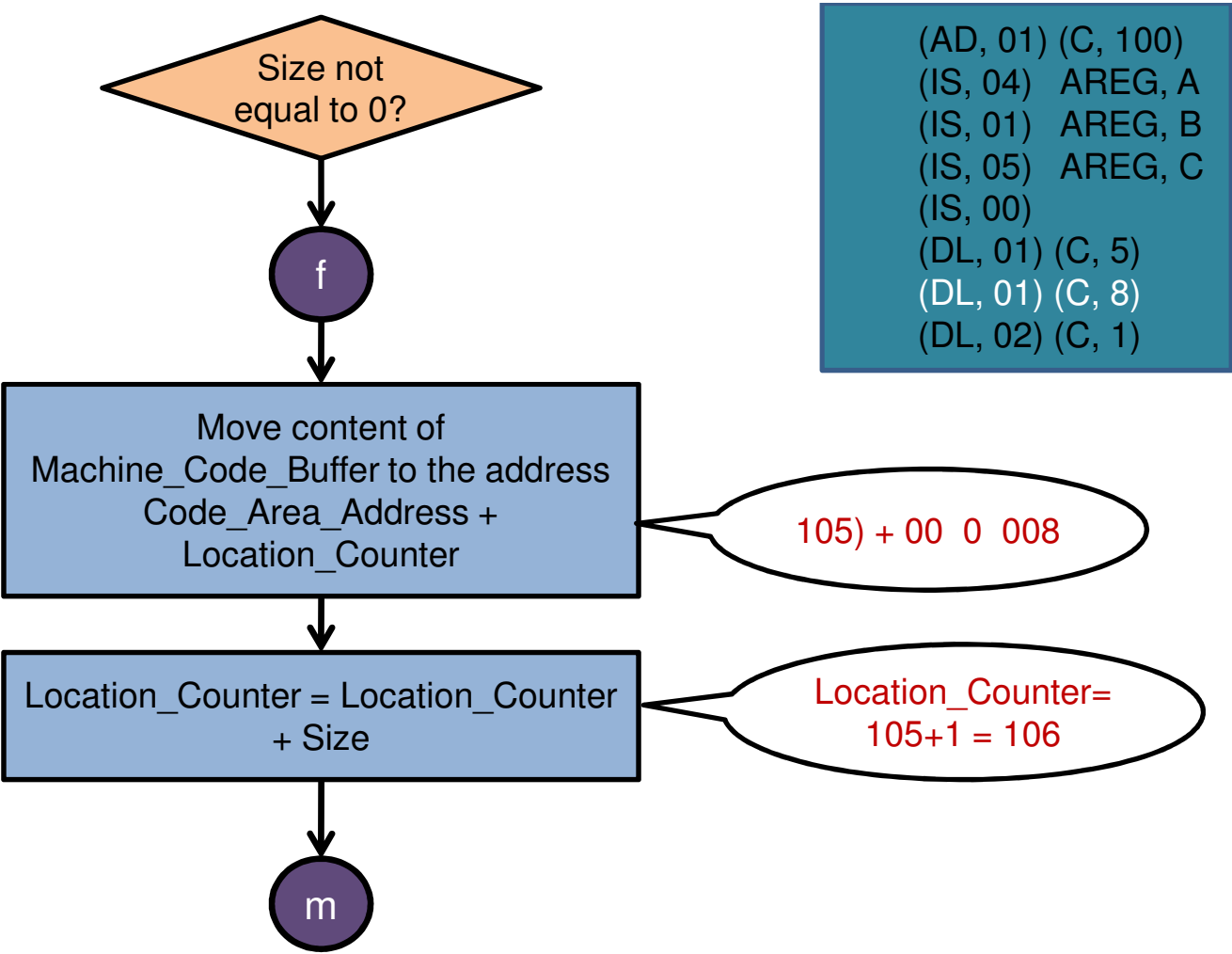
(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)



Code_Area_Address = 0	
Location_Counter = 105	
POOL_Table_Pointer = 1	
Size=1	

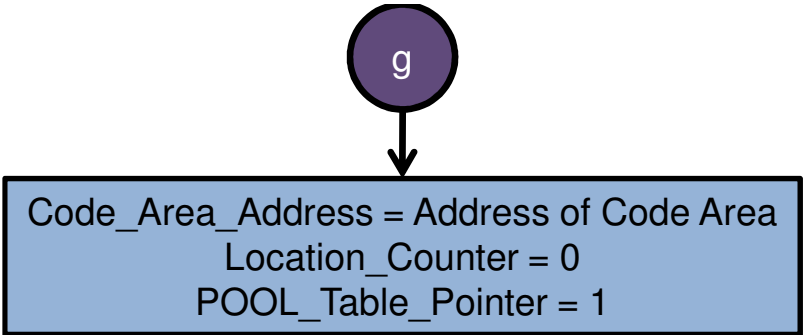
105) + 00 0 008	MCB
-----------------	------------

100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	
104) + 00 0 005	
105) + 00 0 008	

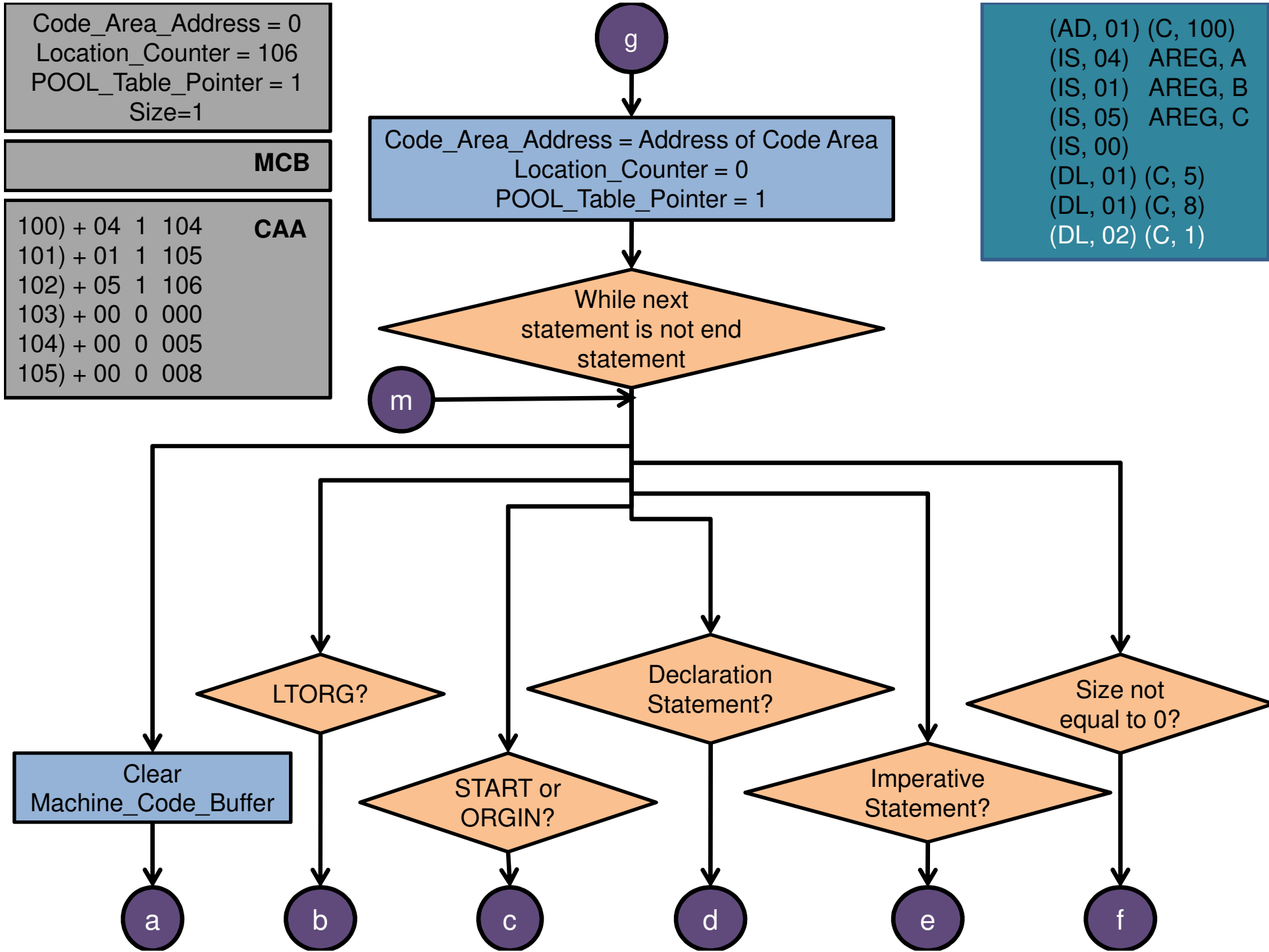


(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

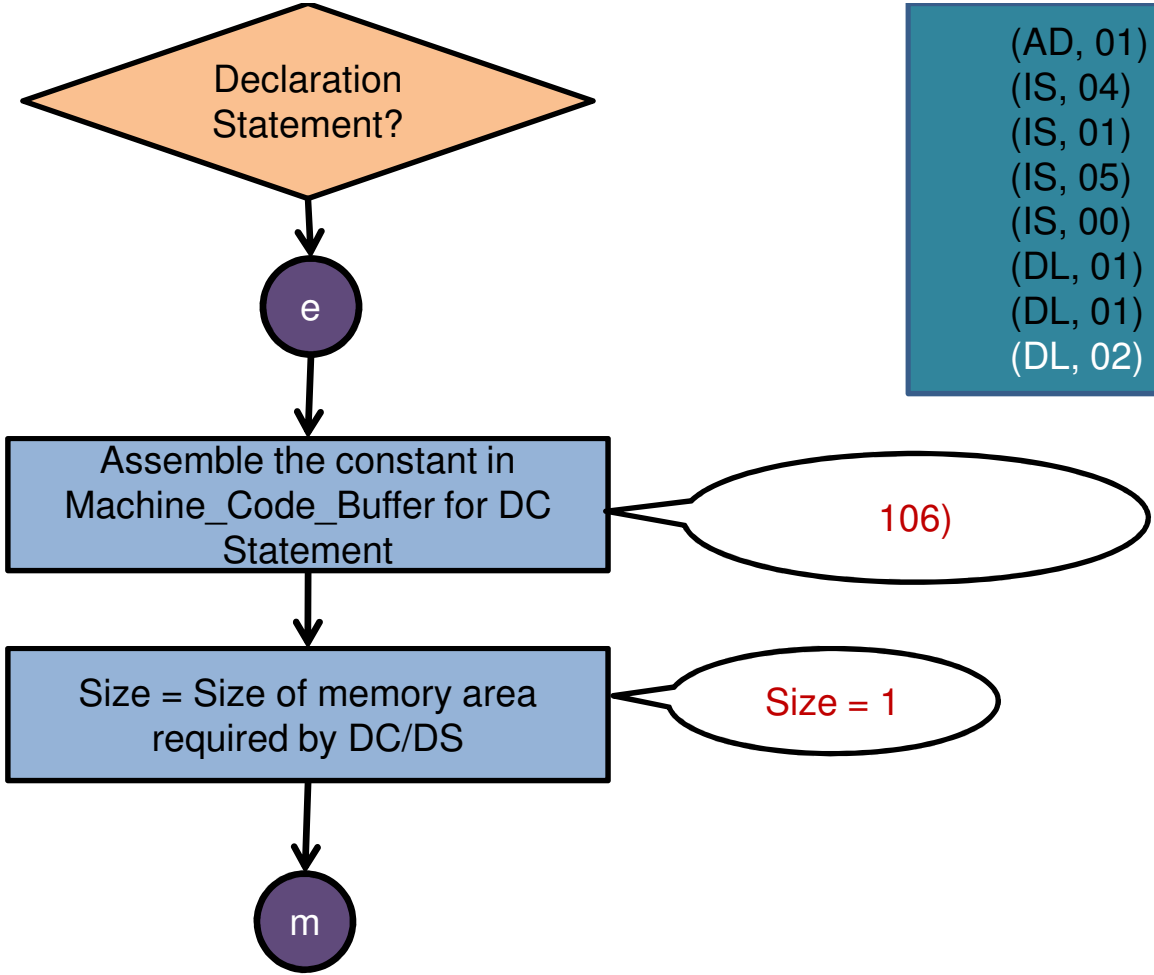
Code_Area_Address = 0			
Location_Counter = 106			
POOL_Table_Pointer = 1			
Size=1			
MCB			
100) + 04	1	104	CAA
101) + 01	1	105	
102) + 05	1	106	
103) + 00	0	000	
104) + 00	0	005	
105) + 00	0	008	



(AD, 01) (C, 100)
 (IS, 04) AREG, A
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 5)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)



Code_Area_Address = 0			
Location_Counter = 106			
POOL_Table_Pointer = 1			
Size=1			
106)	MCB		
100) + 04	1	104	CAA
101) + 01	1	105	
102) + 05	1	106	
103) + 00	0	000	
104) + 00	0	005	
105) + 00	0	008	

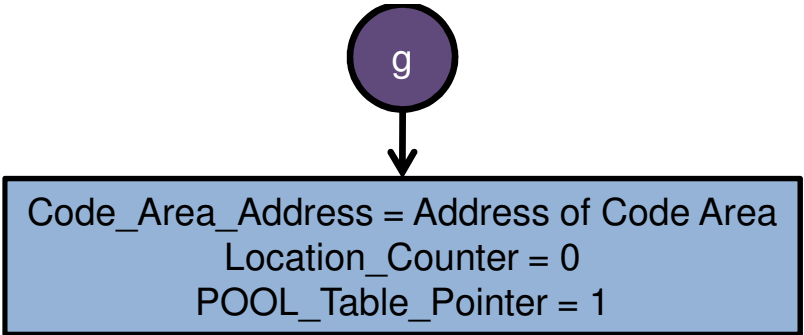


(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

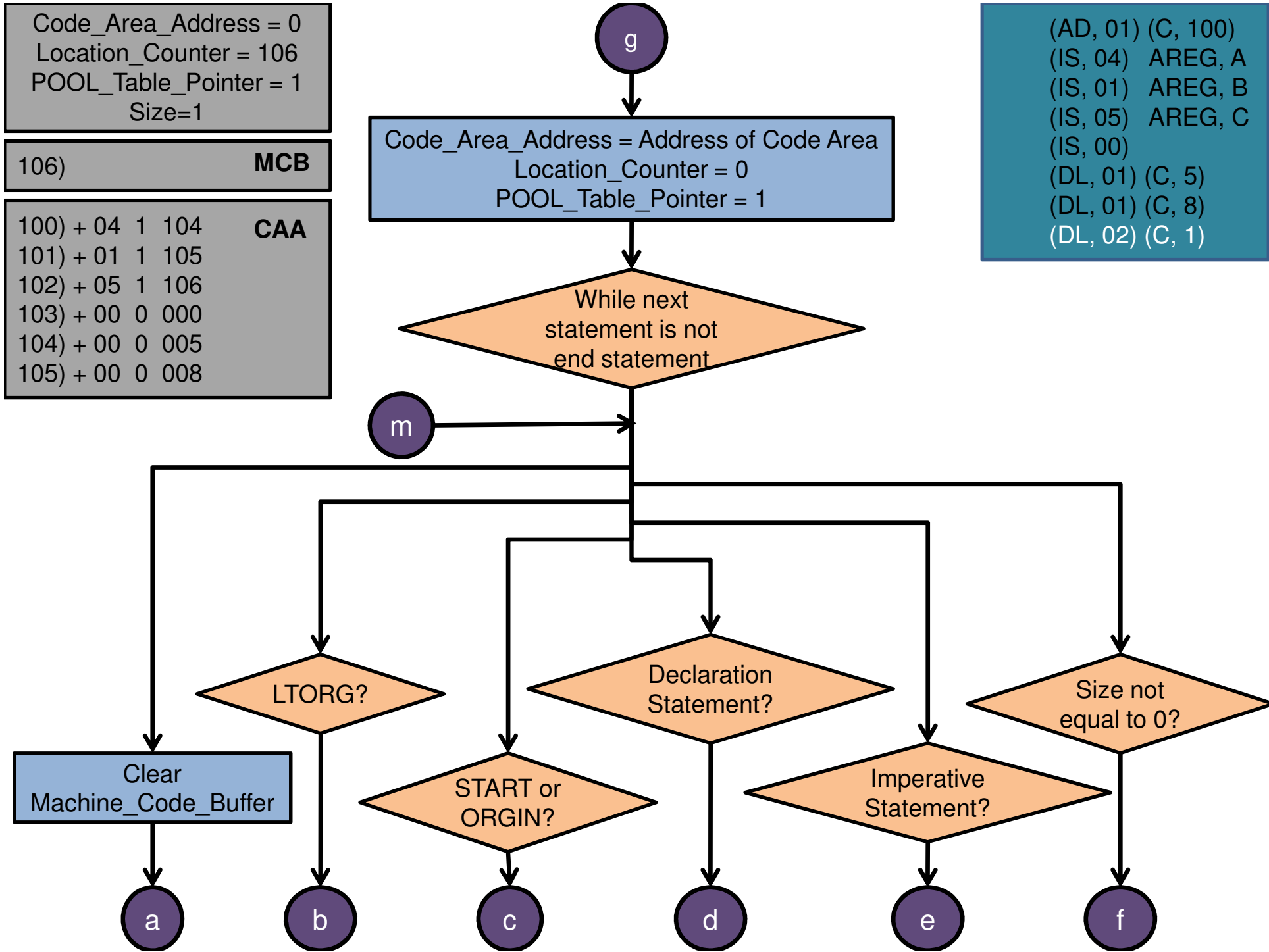
Code_Area_Address = 0		
Location_Counter = 106		
POOL_Table_Pointer = 1		
Size=1		

106)	MCB	
------	------------	--

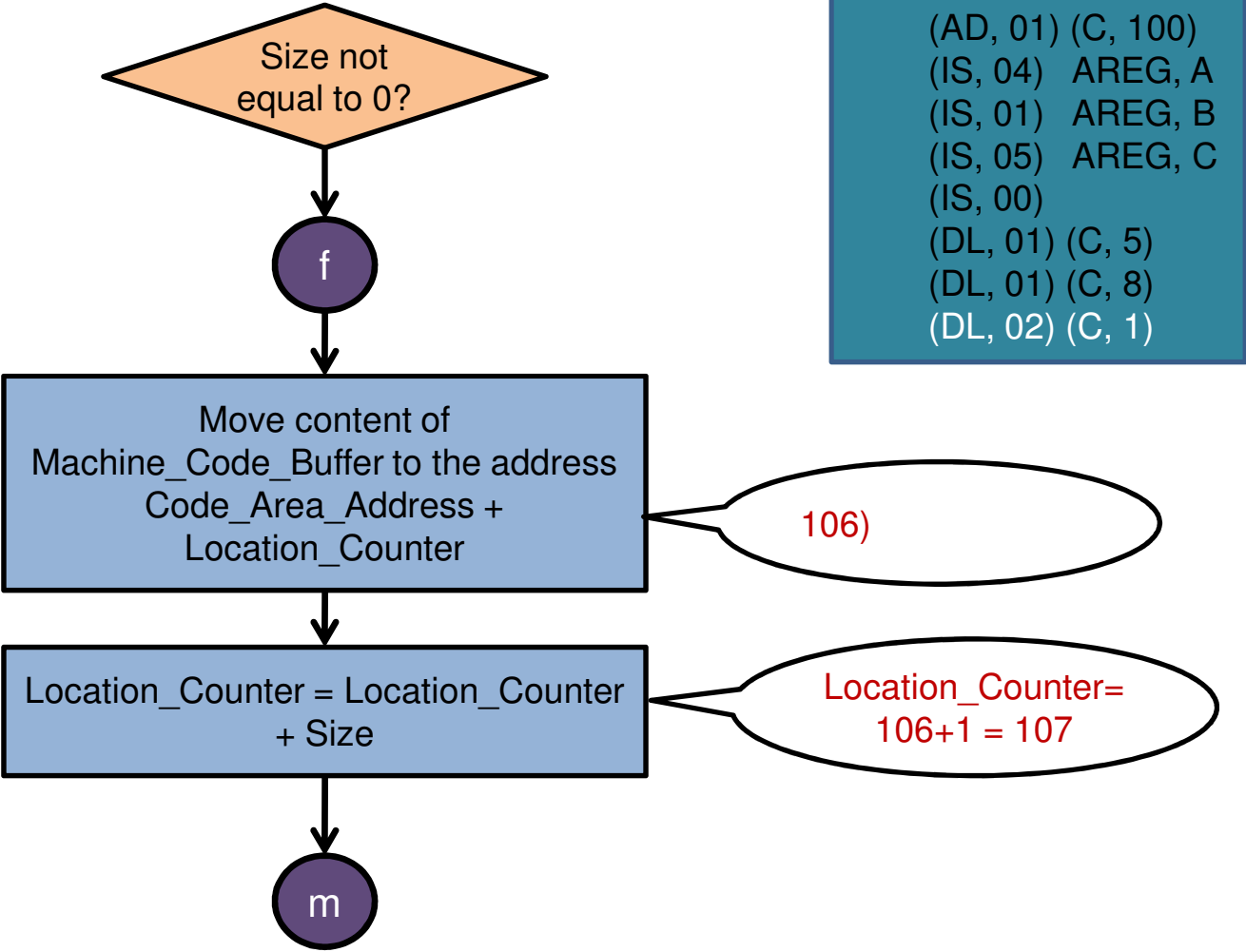
100)	+ 04	1	104	CAA
101)	+ 01	1	105	
102)	+ 05	1	106	
103)	+ 00	0	000	
104)	+ 00	0	005	
105)	+ 00	0	008	



(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

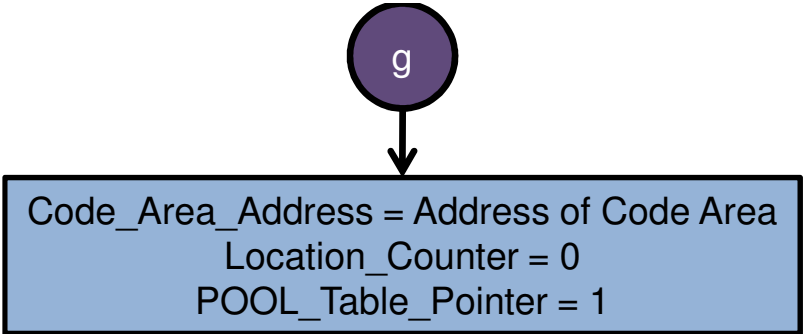


Code_Area_Address = 0		
Location_Counter = 105		
POOL_Table_Pointer = 1		
Size=1		
106)	MCB	
100) + 04 1 104	CAA	
101) + 01 1 105		
102) + 05 1 106		
103) + 00 0 000		
104) + 00 0 005		
105) + 00 0 008		
106)		

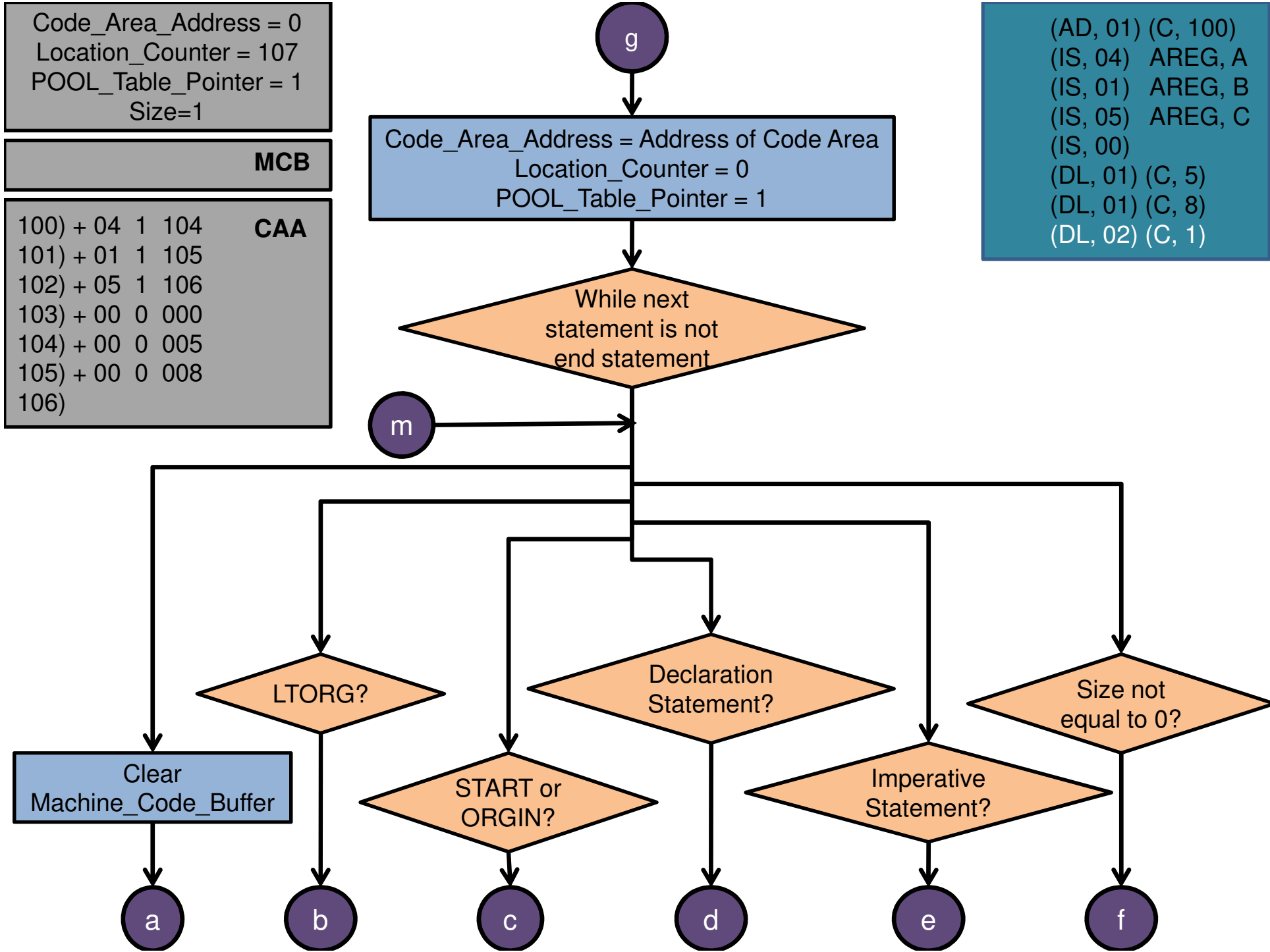


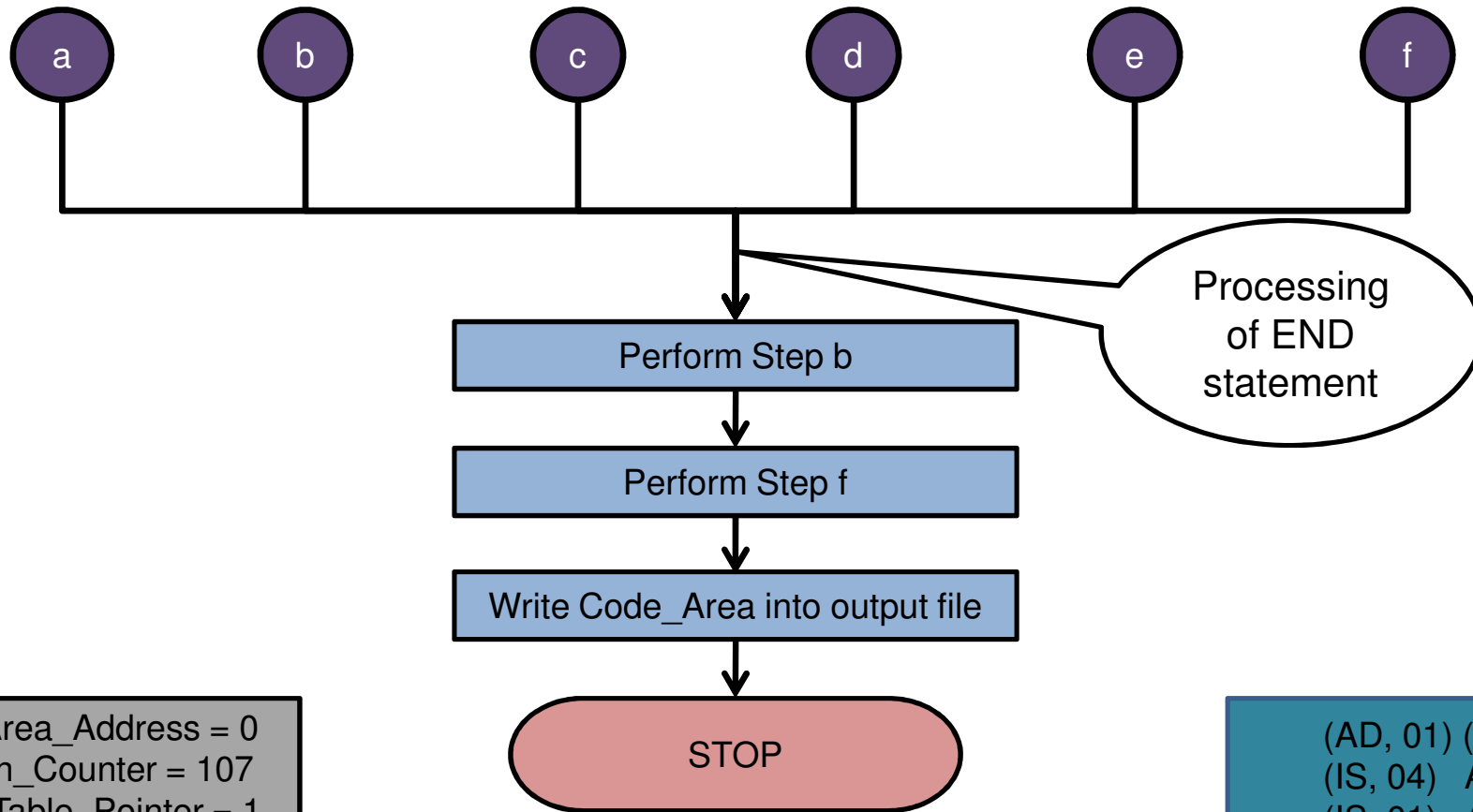
(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0	
Location_Counter = 107	
POOL_Table_Pointer = 1	
Size=1	
MCB	
100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	
104) + 00 0 005	
105) + 00 0 008	
106)	



(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)





Code_Area_Address = 0
 Location_Counter = 107
 POOL_Table_Pointer = 1
 Size=1

MCB

100) + 04 1 104	CAA
101) + 01 1 105	
102) + 05 1 106	
103) + 00 0 000	
104) + 00 0 005	
105) + 00 0 008	
106)	

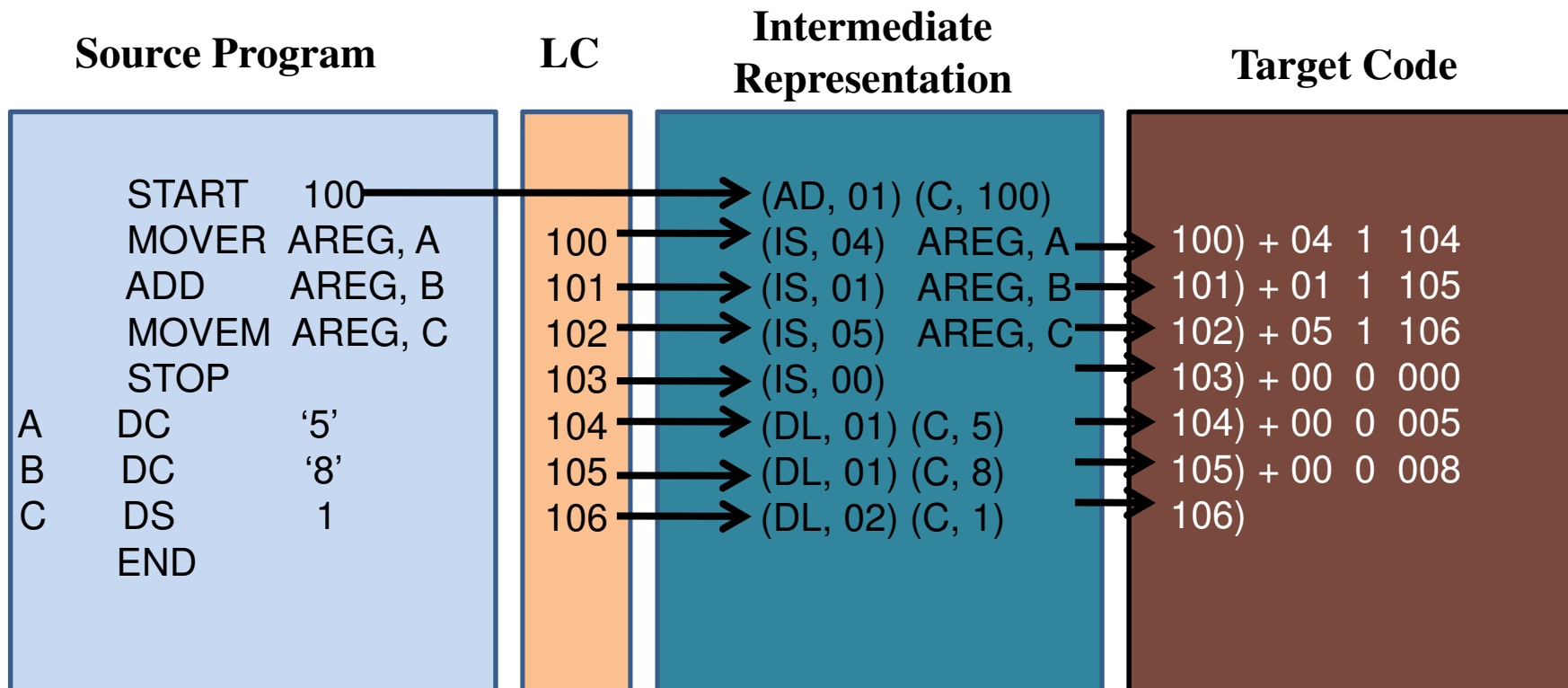
Output File

```

100) + 04 1 104
101) + 01 1 105
102) + 05 1 106
103) + 00 0 000
104) + 00 0 005
105) + 00 0 008
106)
  
```

```

(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)
  
```



Symbol	Address	Length
A	104	1
B	105	1
C	106	1

Symbol Table

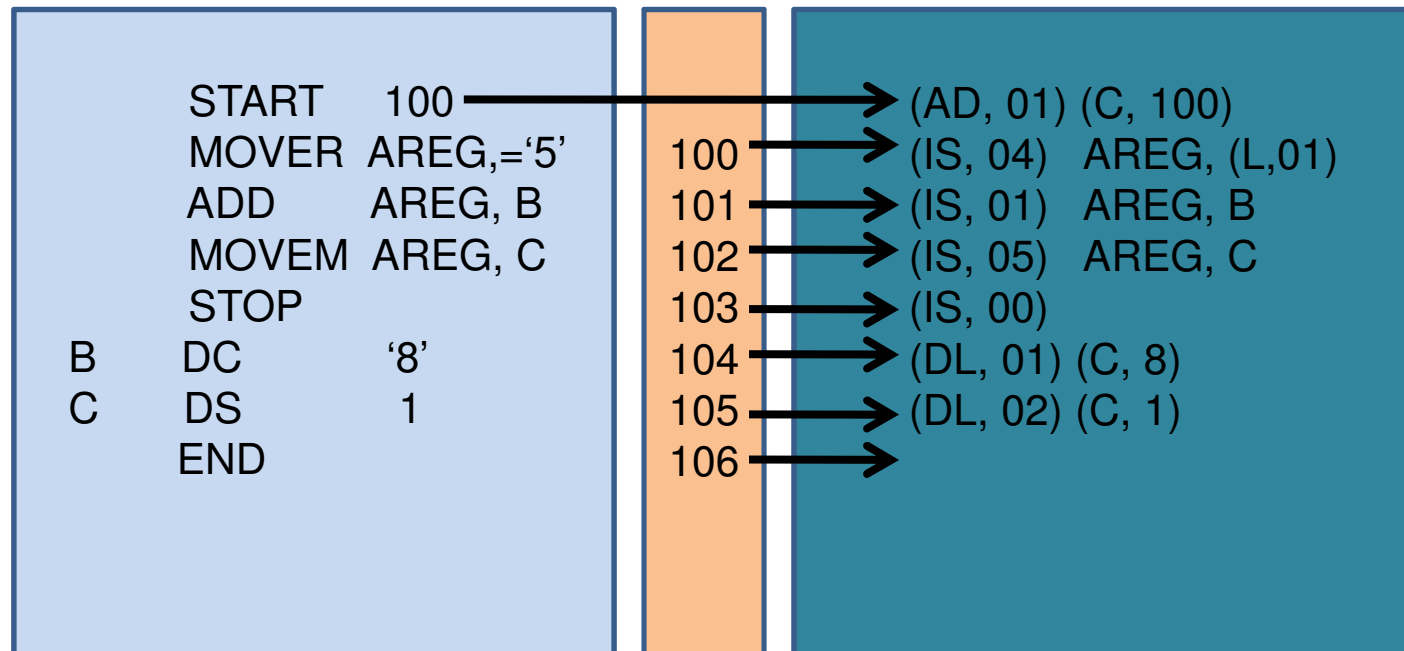
Mnemonic Opcode	Class	.Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

```
START 100
MOVER AREG, ='5'
ADD    AREG, B
MOVEM  AREG, C
STOP
B      DC      '8'
C      DS      1
END
```

Source Program

LC

Intermediate Representation



Literal_Table_P ointer	Literal	Address
1	= '5'	106
2		

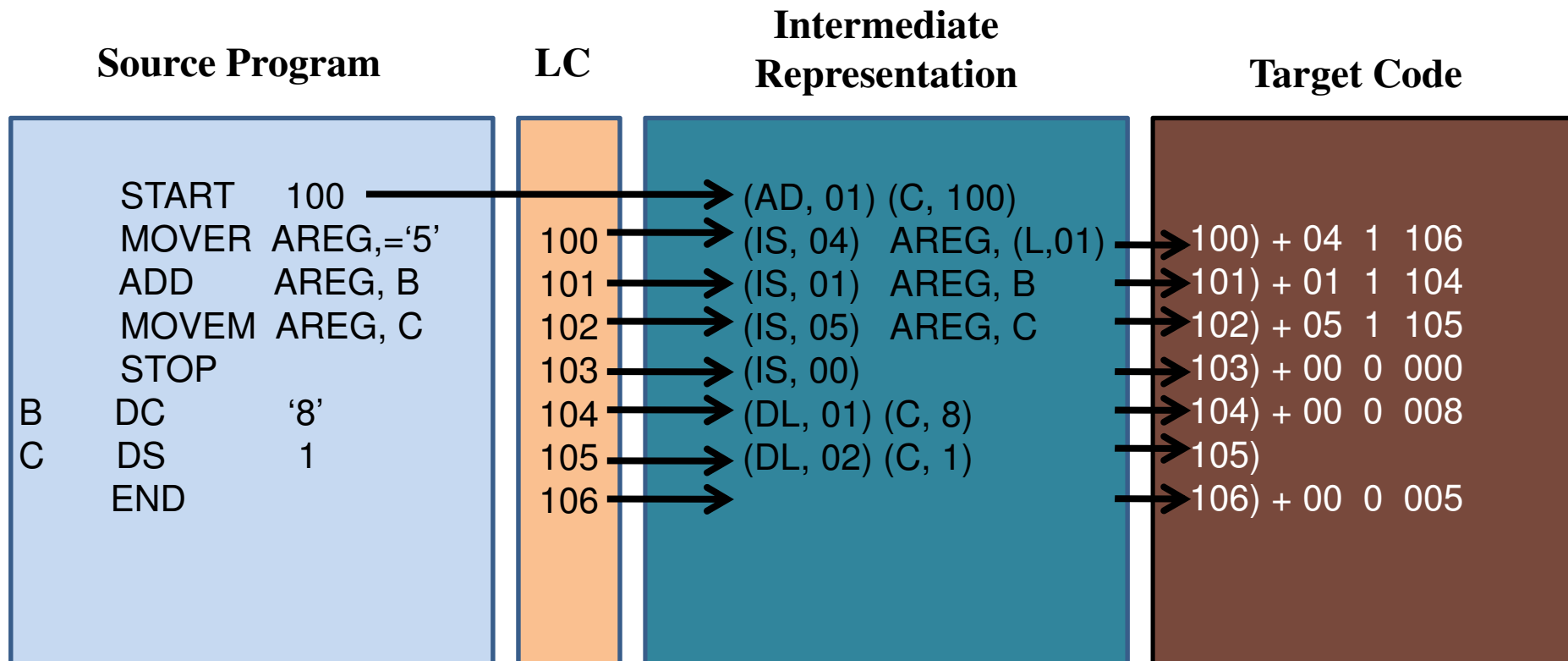
Literal Table

POOL_Table_ Pointer	Literal_Table_ Pointer
1	1
2	2

POOL Table

Symbol	Address	Length
B	105	1
C	106	1

Symbol Table



Literal_Table_P ointer	Literal	Address
1	= '5'	106
2		

Literal Table

POOL_Table_ Pointer	Literal_Table_ Pointer
1	1
2	2

POOL Table

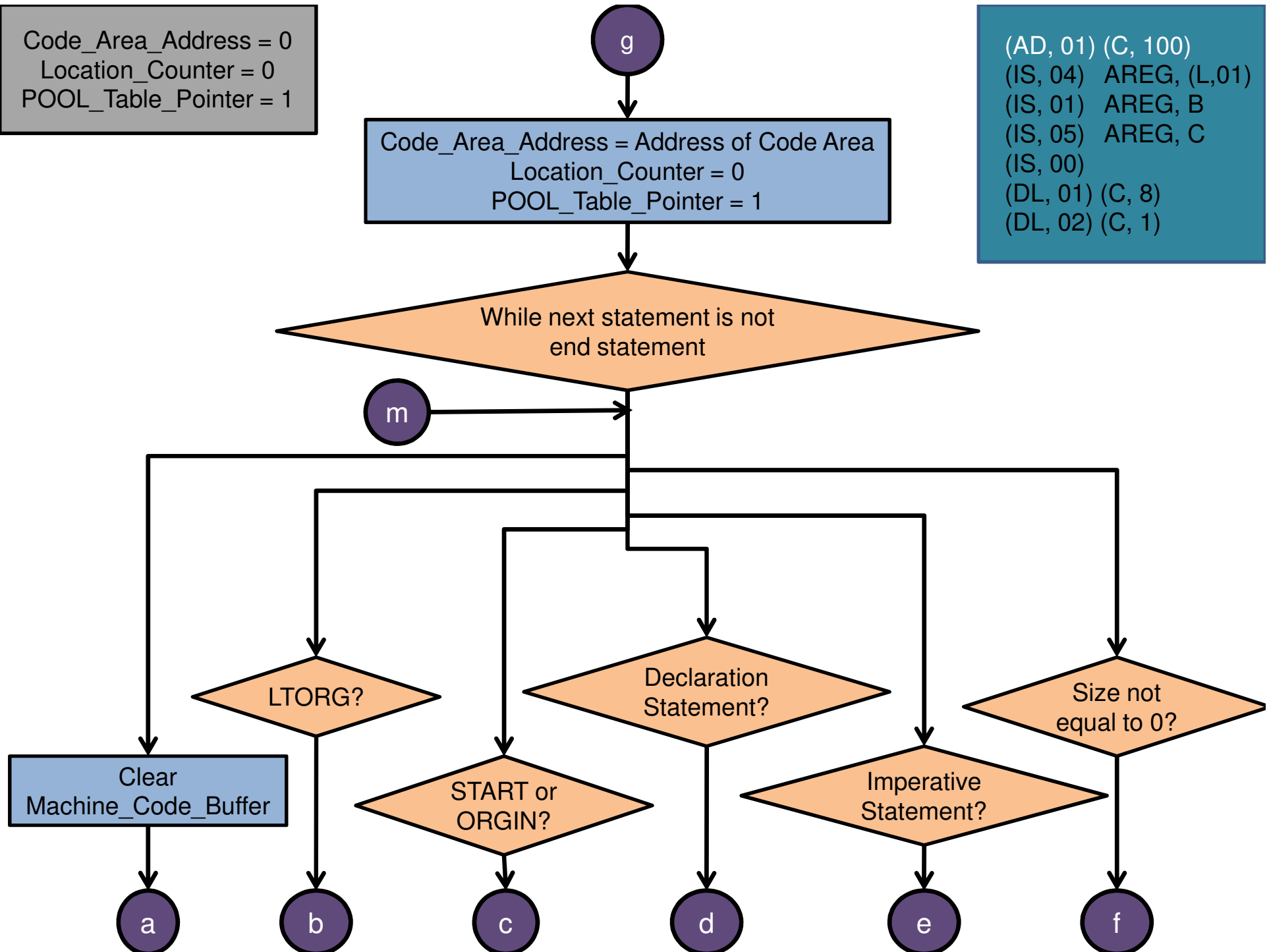
Symbol	Address	Length
B	104	1
C	105	1

Symbol Table

Code_Area_Address = 0
Location_Counter = 0
POOL_Table_Pointer = 1

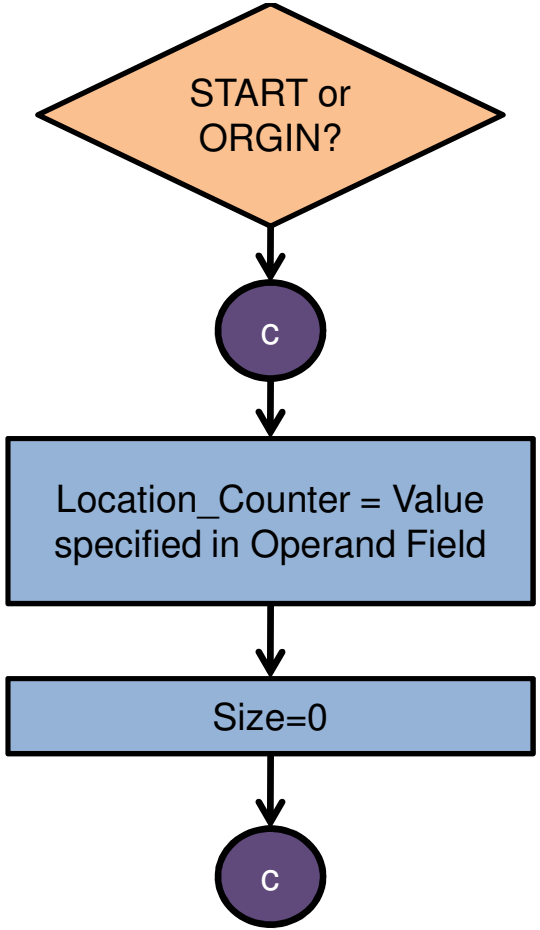
Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
Location_Counter = 0
POOL_Table_Ponter = 1
Size=0

Location_Counter
=100

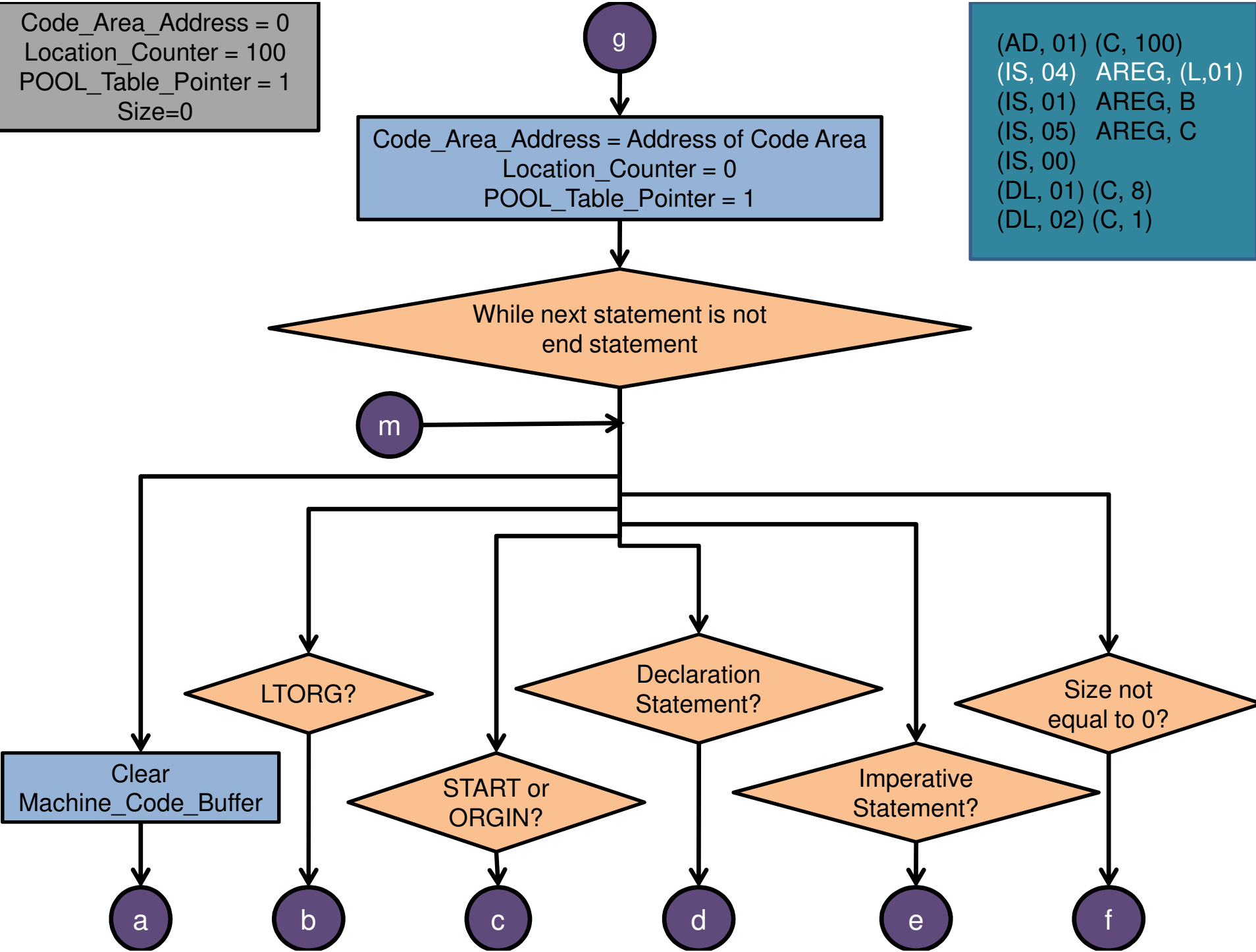


(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

Code_Area_Address = 0
Location_Counter = 100
POOL_Table_Pointer = 1
Size=0

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
 Location_Counter = 100
 POOL_Table_Pointer = 1
 Size=0

100) + 04 1 106 **MCB**

Imperative
Statement?

e

(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Address of
Literal "5" = 106

Get the operand address from Symbol
Table or LITERAL TABLE

Assemble the instruction in Machine_Code_Buffer
(MCB)

100) + 04 1 106

Size = 1

Size = size of instruction

m

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

OP Table

Register	Code
AREG	1
BREG	2
CREG	3
DREG	4

Mrs. Sunita M Dol, CSE Dept

Literal Table Pointer	Literal	Address
1	= '5'	106
2		

Literal Table

POOL_Table Pointer	Literal_Table Pointer
1	1
2	2

POOL Table

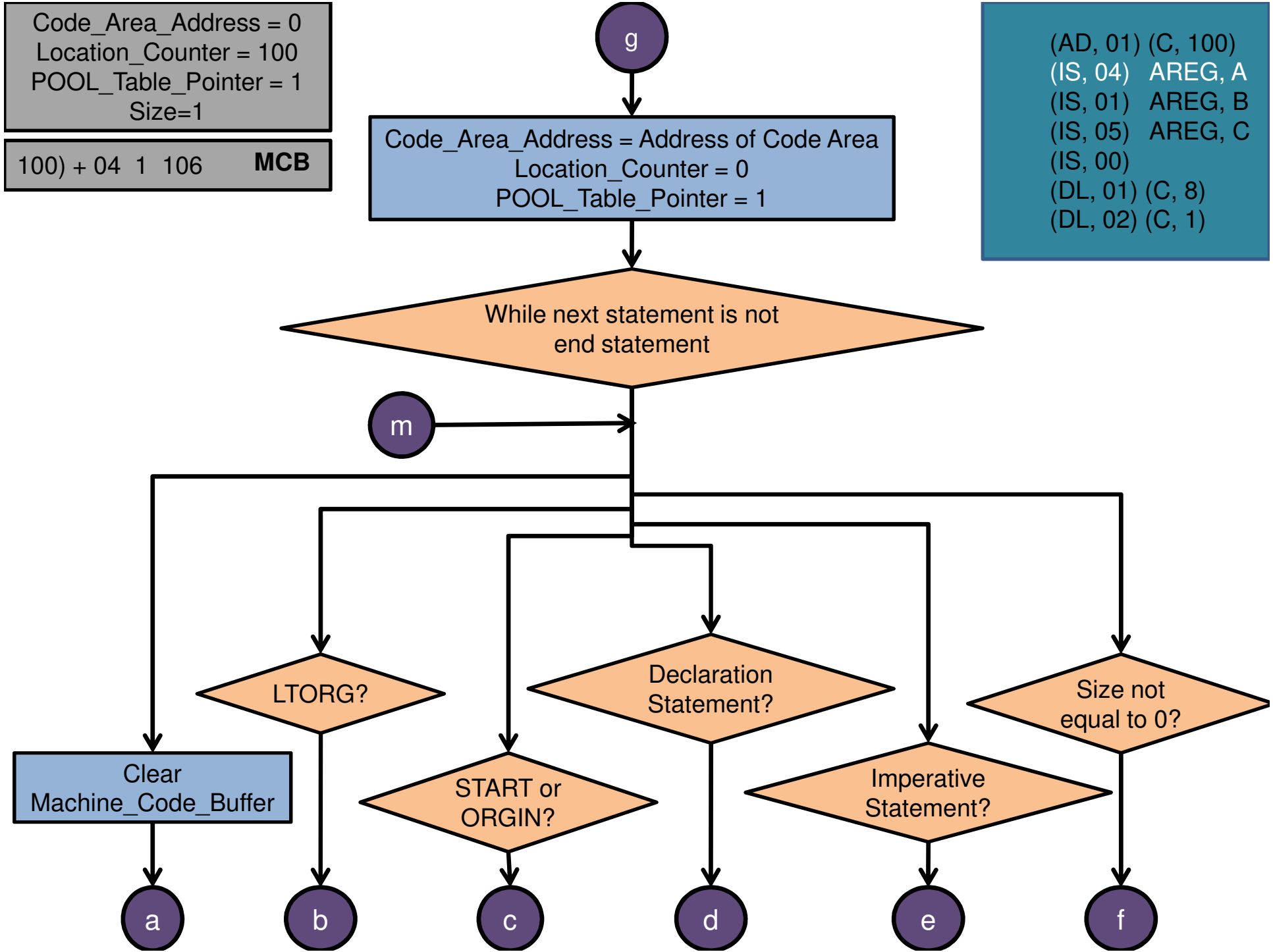
200

Code_Area_Address = 0
Location_Counter = 100
POOL_Table_Pointer = 1
Size=1

100) + 04 1 106 **MCB**

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

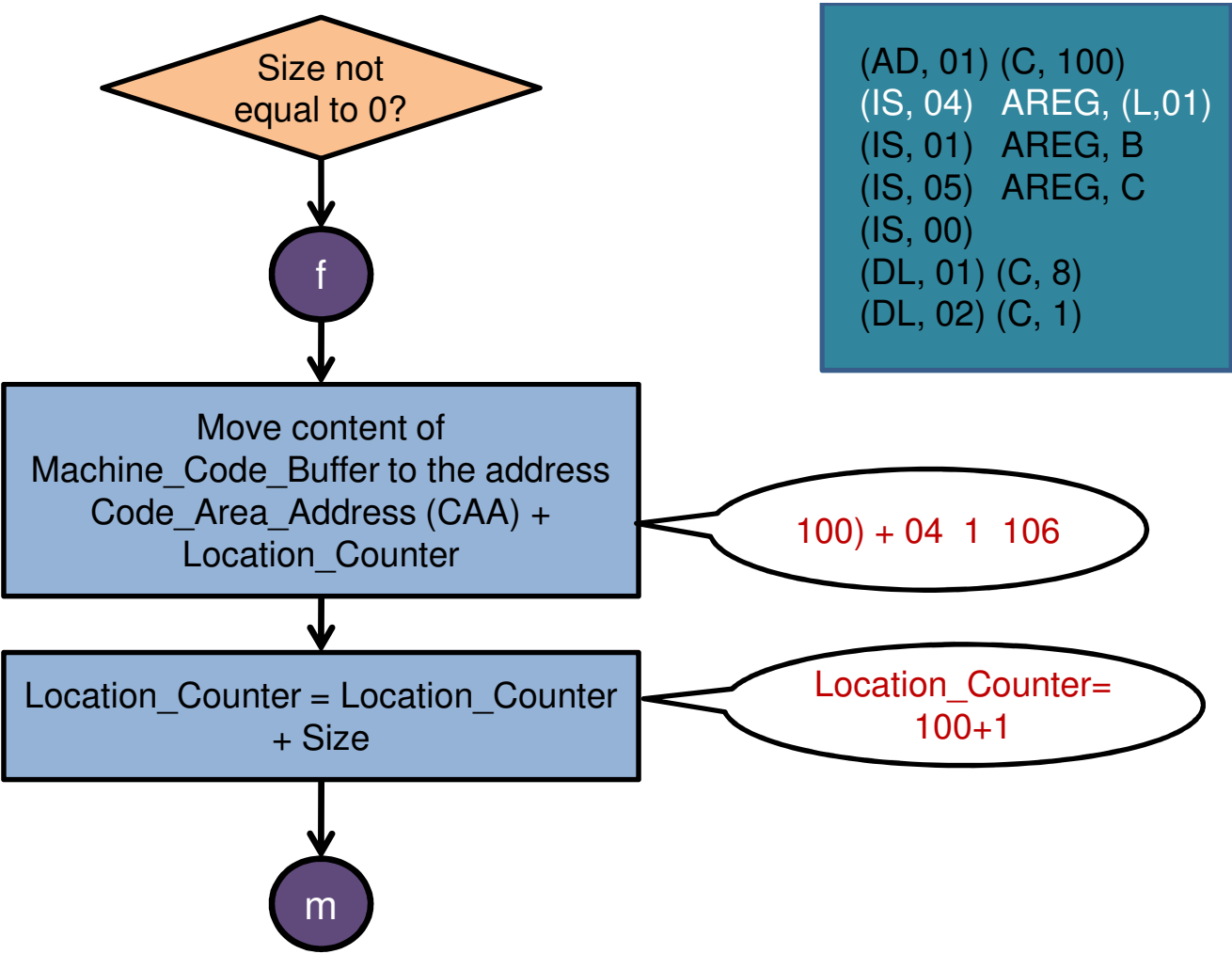
(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 100	
POOL_Table_Pointer = 1	
Size=1	

100) + 04 1 106	MCB
-----------------	------------

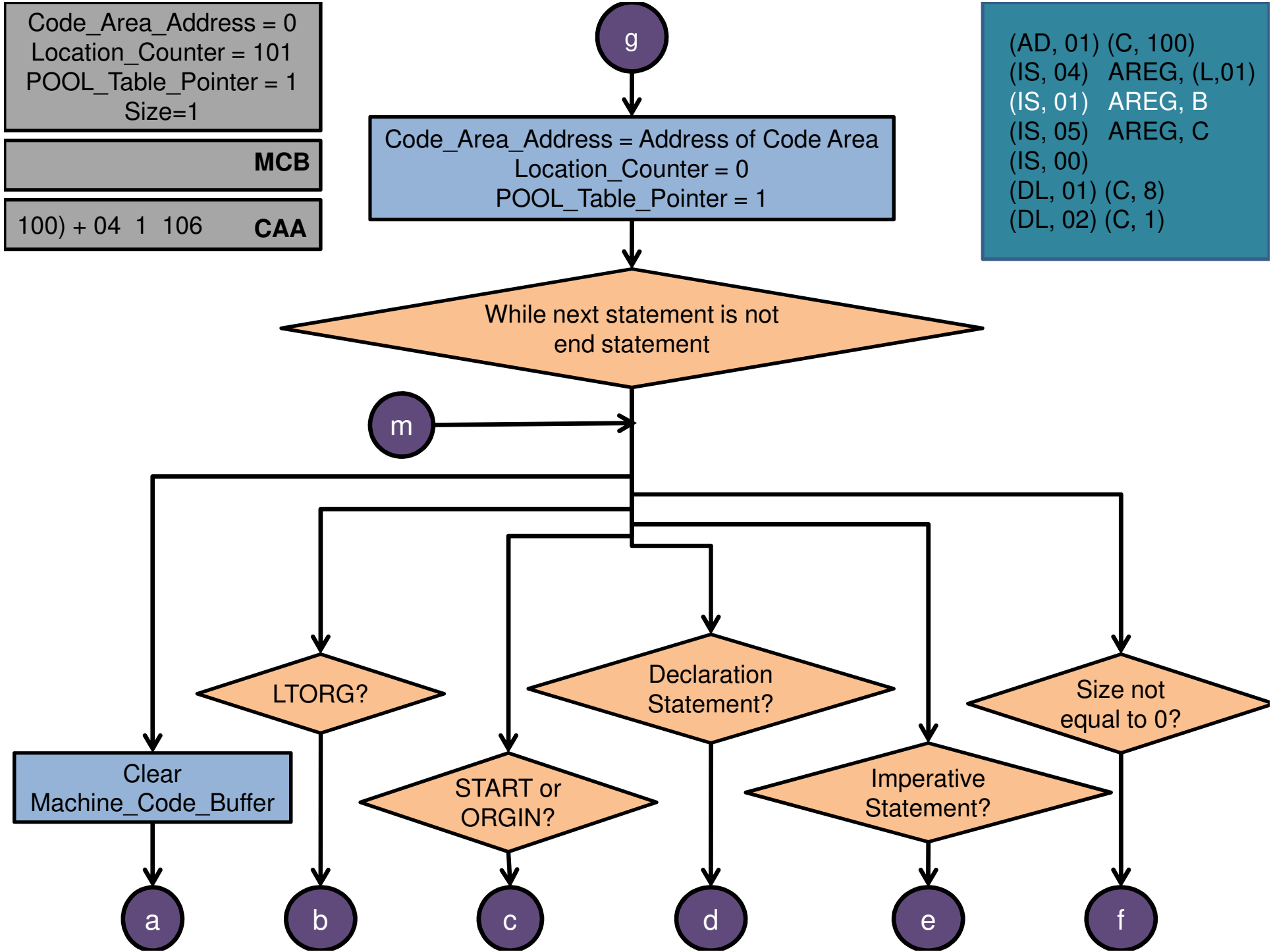
100) + 04 1 106	CAA
-----------------	------------



Code_Area_Address = 0 Location_Counter = 101 POOL_Table_Pointer = 1 Size=1	
	MCB
100) + 04 1 106	CAA

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
 Location_Counter = 101
 POOL_Table_Pointer = 1
 Size=1

101) + 01 1 104 **MCB**

100) + 04 1 106 **CAA**

Imperative
Statement?

e

Get the operand address from Symbol
Table or LITERAL TABLE

Address of B =
105

Assemble the instruction in Machine_Code_Buffer

101) + 01 1 104

Size = 1

Size = size of instruction

m

(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Symbol	Address	Length
B	104	1
C	105	1

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

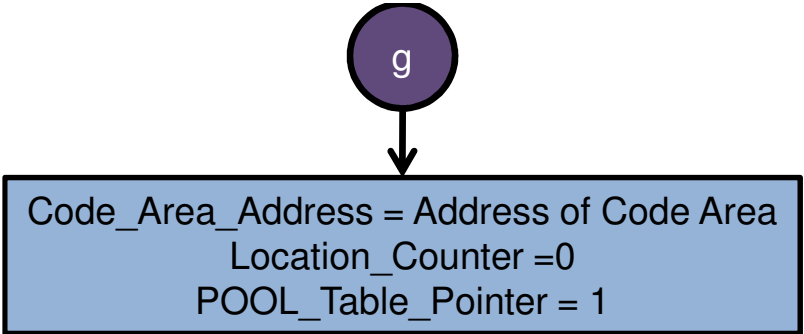
OP Table

Register	Code
AREG	1
BREG	2
CREG	3
DREG	4

Mrs. Sunita M Dol, CSE Dept

Symbol Table

Code_Area_Address = 0	
Location_Counter = 101	
POOL_Table_Pointer = 1	
Size=1	
101) + 01 1 104	MCB
100) + 04 1 106	CAA

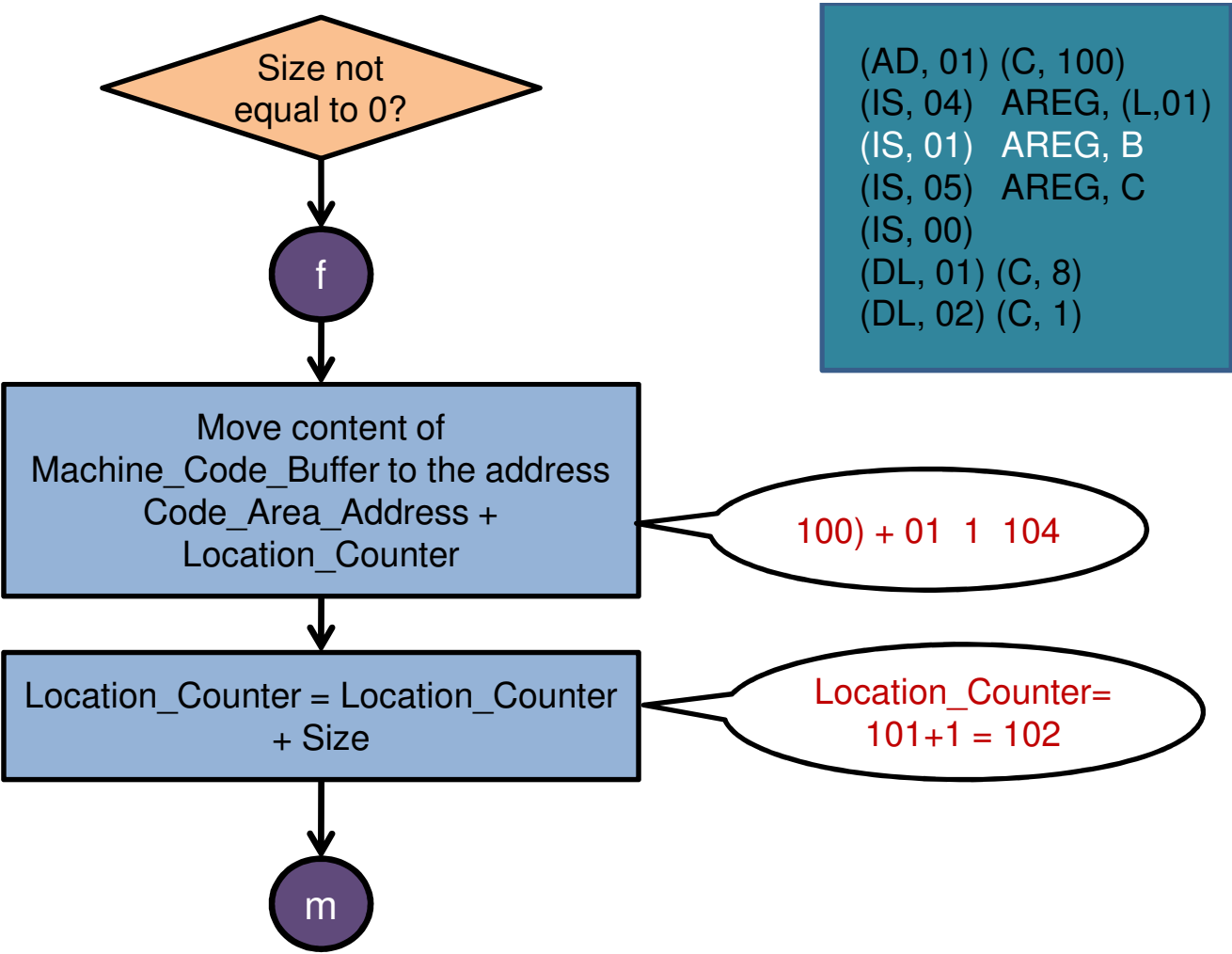


(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

Code_Area_Address = 0
Location_Counter = 101
POOL_Table_Pointer = 1
Size=1

101) + 01 1 104	MCB
-----------------	------------

100) + 04 1 106	CAA
101) + 01 1 104	

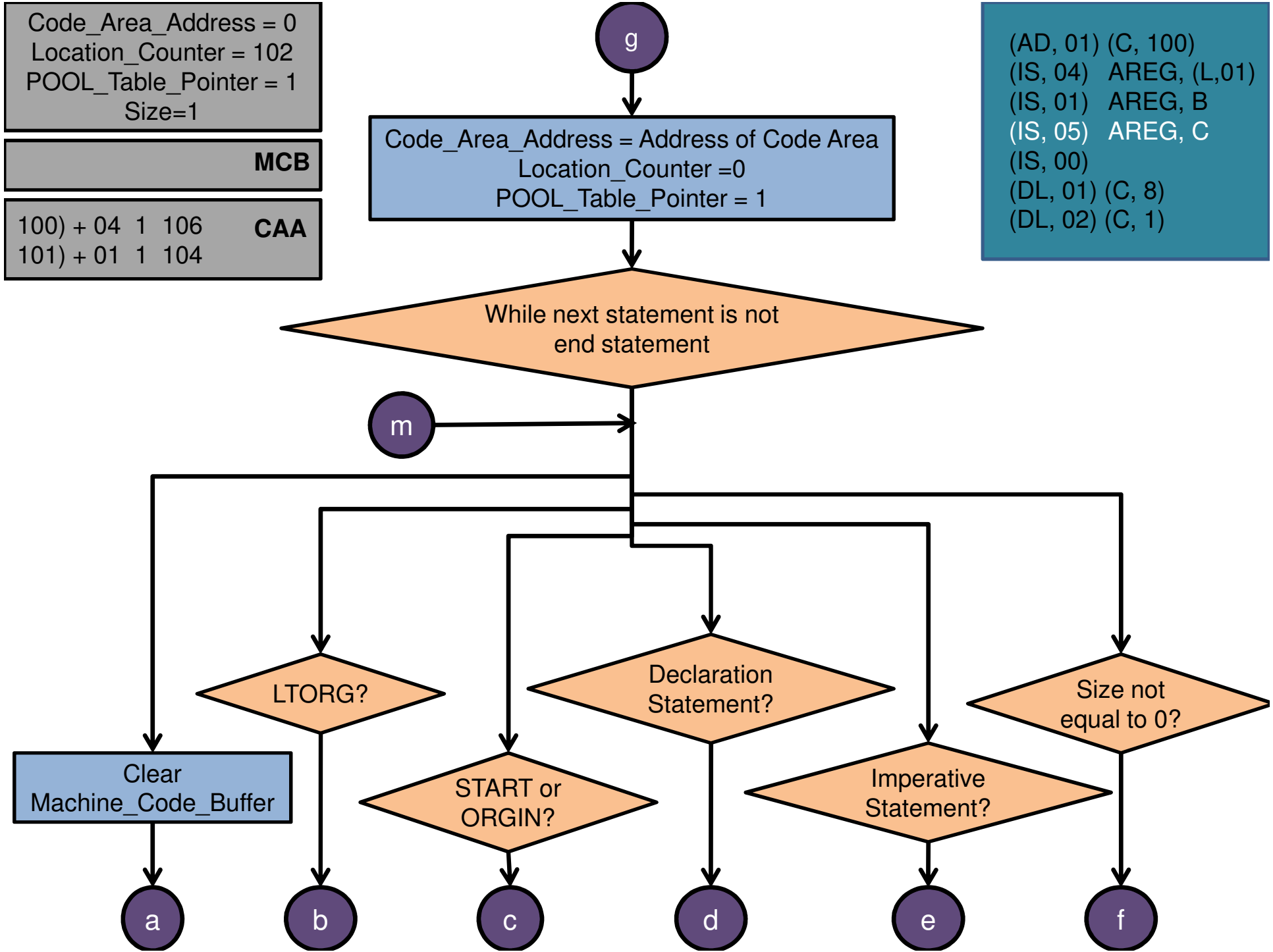


(AD, 01)	(C, 100)
(IS, 04)	AREG, (L,01)
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0 Location_Counter = 102 POOL_Table_Pointer = 1 Size=1	
	MCB
100) + 04 1 106 101) + 01 1 104	CAA

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0
 Location_Counter = 102
 POOL_Table_Pointer = 1
 Size=1

102) + 05 1 105 **MCB**

100) + 04 1 106 **CAA**
 101) + 01 1 104

Imperative
Statement?

e

Get the operand address from Symbol
Table or LITERAL TABLE

Address of B =
105

Assemble the instruction in Machine_Code_Buffer

102) + 05 1 105

Size = 1

Size = size of instruction

m

(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Symbol	Address	Length
B	104	1
C	105	1

Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

8/13/2015

OP Table

Register	Code
AREG	1
BREG	2
CREG	3
DREG	4

Mrs. Sunita M Dol, CSE Dept

Symbol Table

Code_Area_Address = 0	
Location_Counter = 102	
POOL_Table_Pointer = 1	
Size=1	

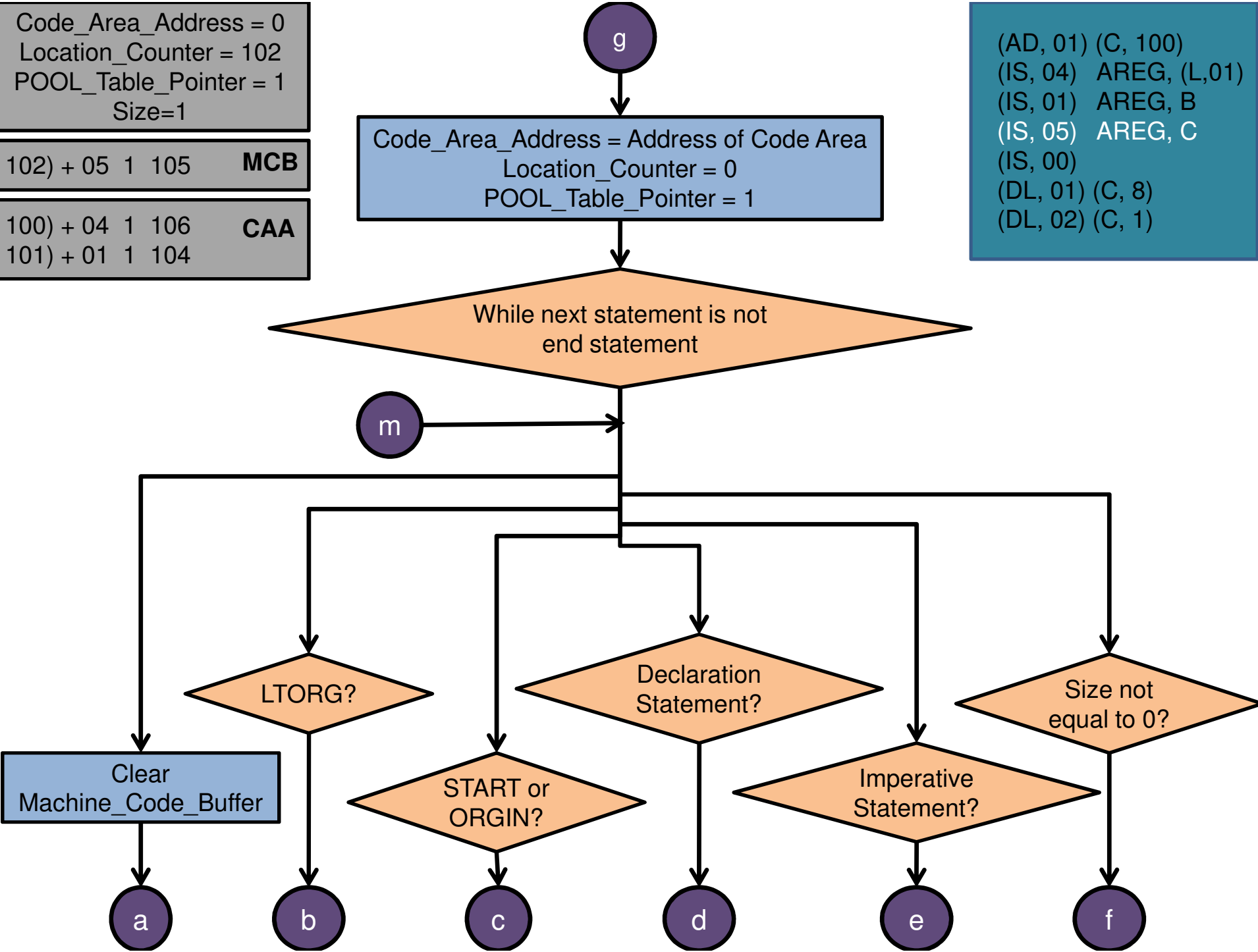
102) + 05 1 105	MCB
-----------------	------------

100) + 04 1 106	CAA
101) + 01 1 104	

g

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

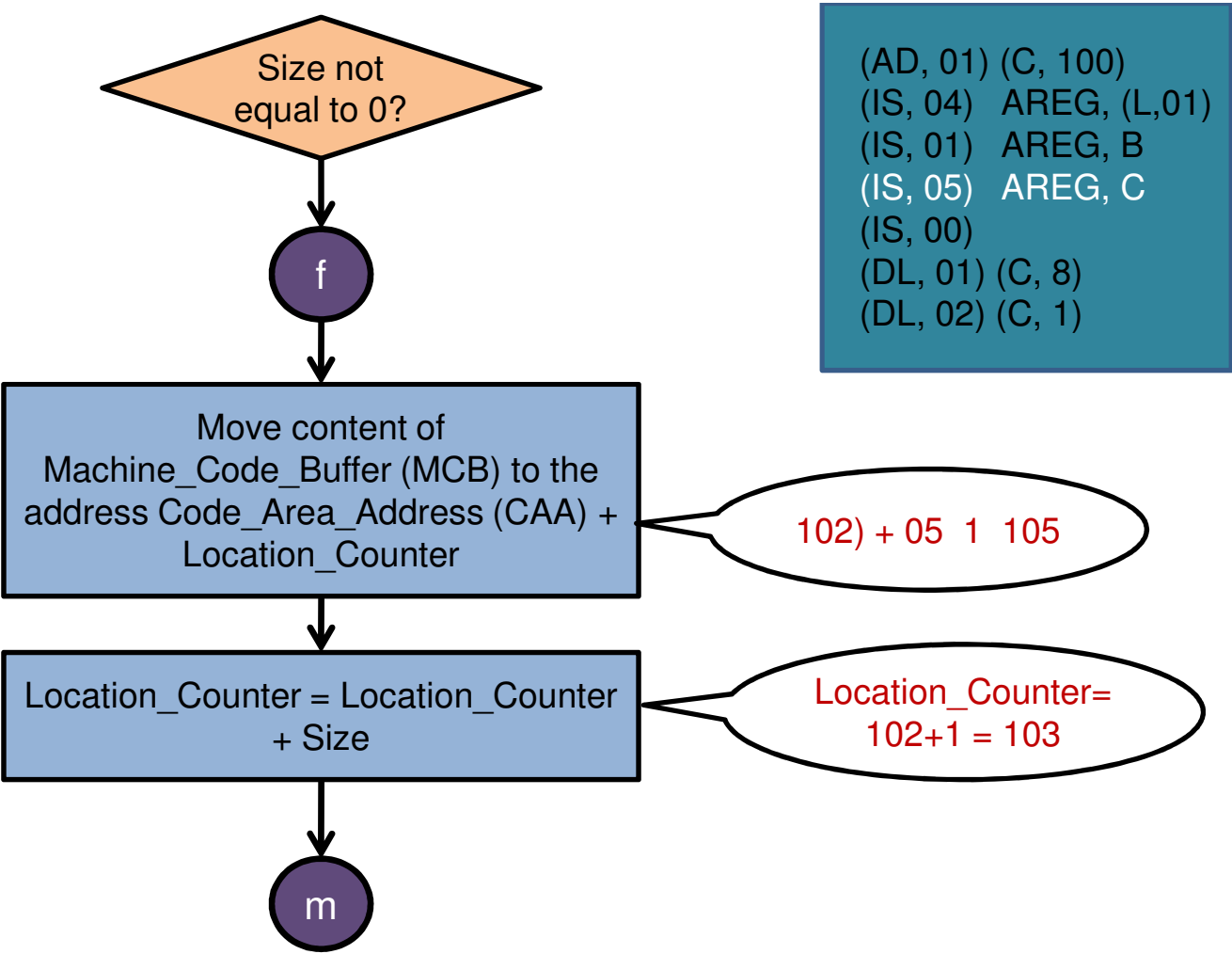
(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 102	
POOL_Table_Pointer = 1	
Size=1	

102) + 05 1 105	MCB
-----------------	------------

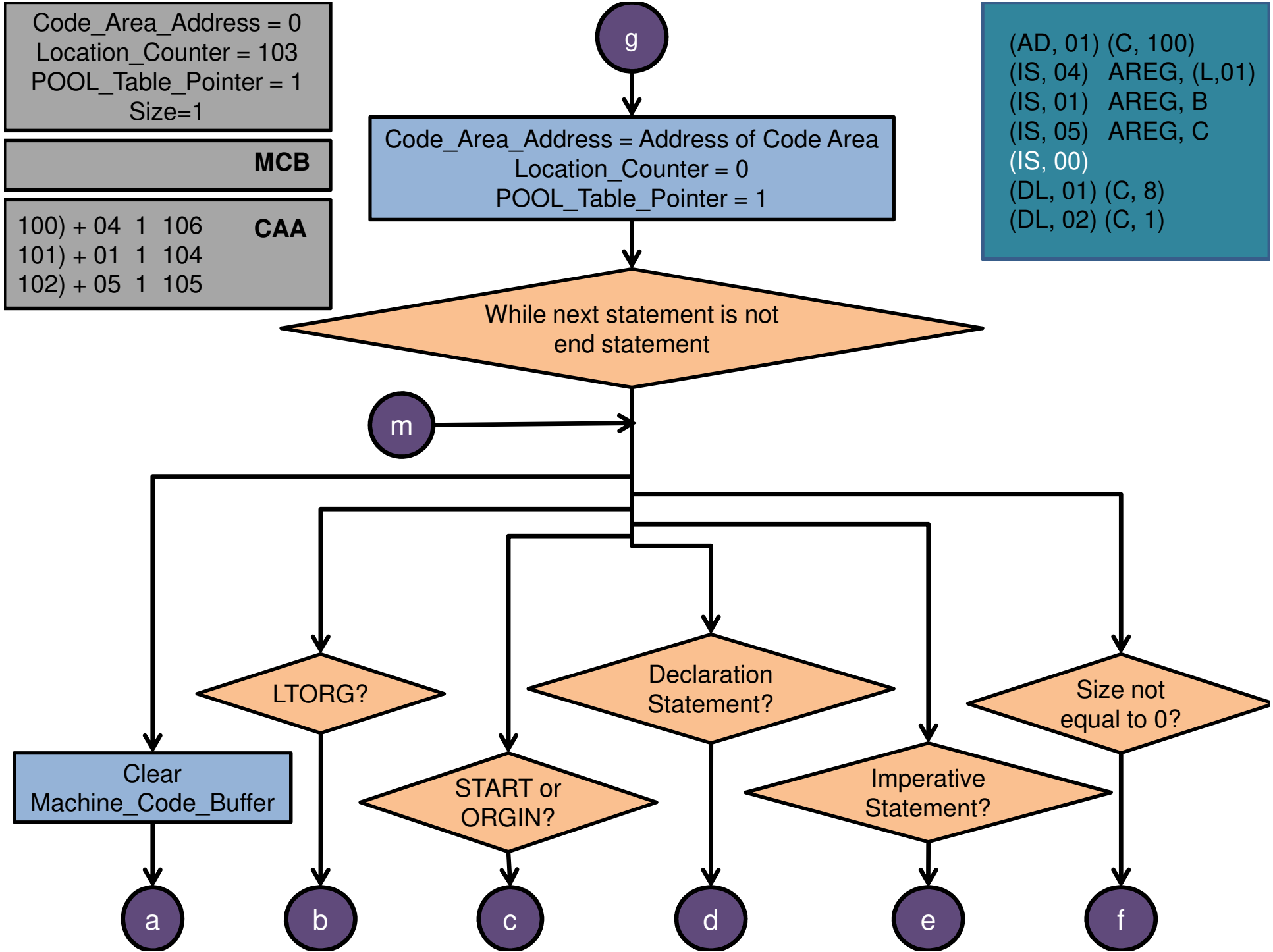
100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	



Code_Area_Address = 0 Location_Counter = 103 POOL_Table_Pointer = 1 Size=1	
	MCB
100) + 04 1 106 101) + 01 1 104 102) + 05 1 105	CAA

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

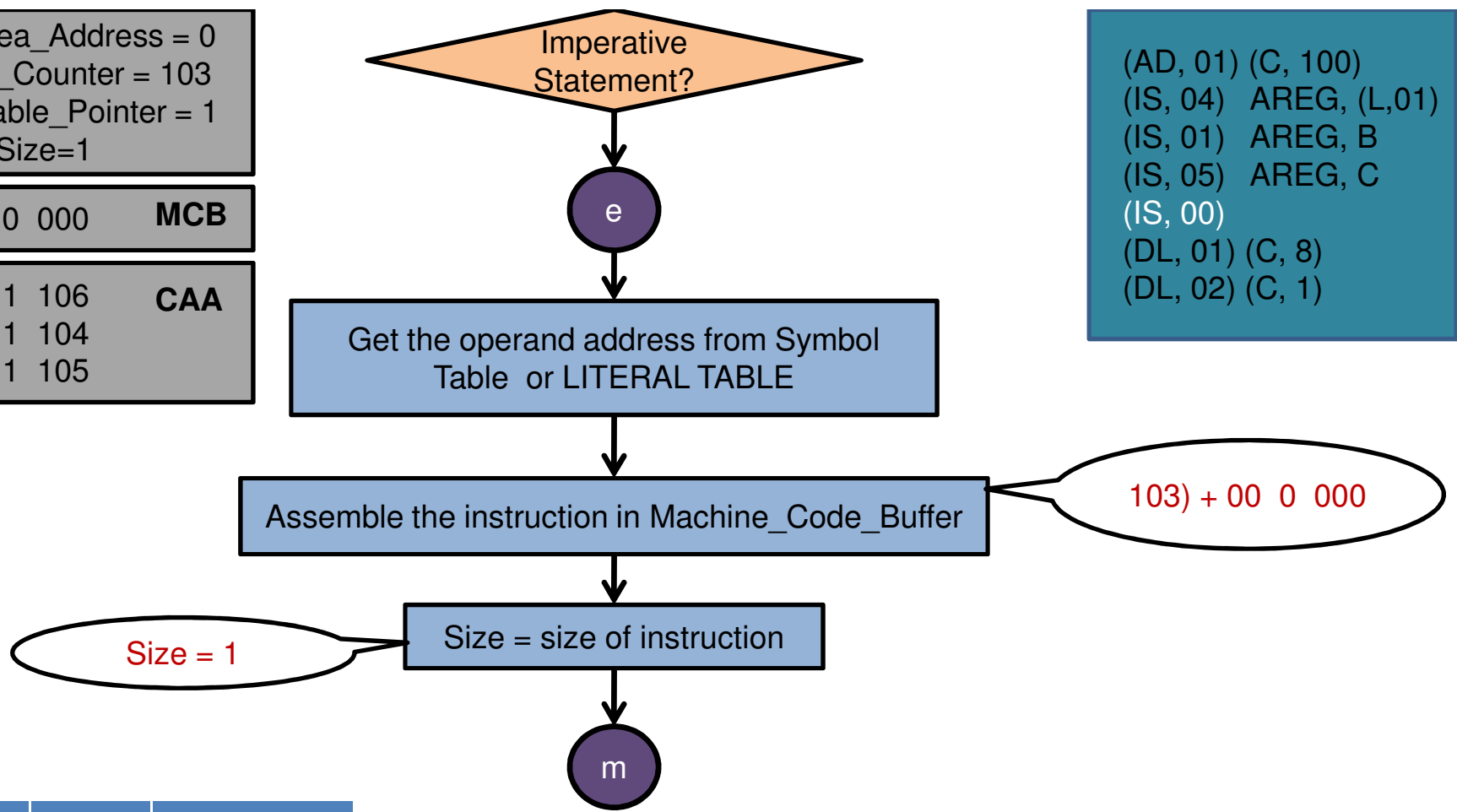


Code_Area_Address = 0
 Location_Counter = 103
 POOL_Table_Pointer = 1
 Size=1

103) + 00 0 000 **MCB**

100) + 04 1 106 **CAA**
 101) + 01 1 104
 102) + 05 1 105

(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)



Mnemonic Opcode	Class	Mnemonic Info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	.	
	.	

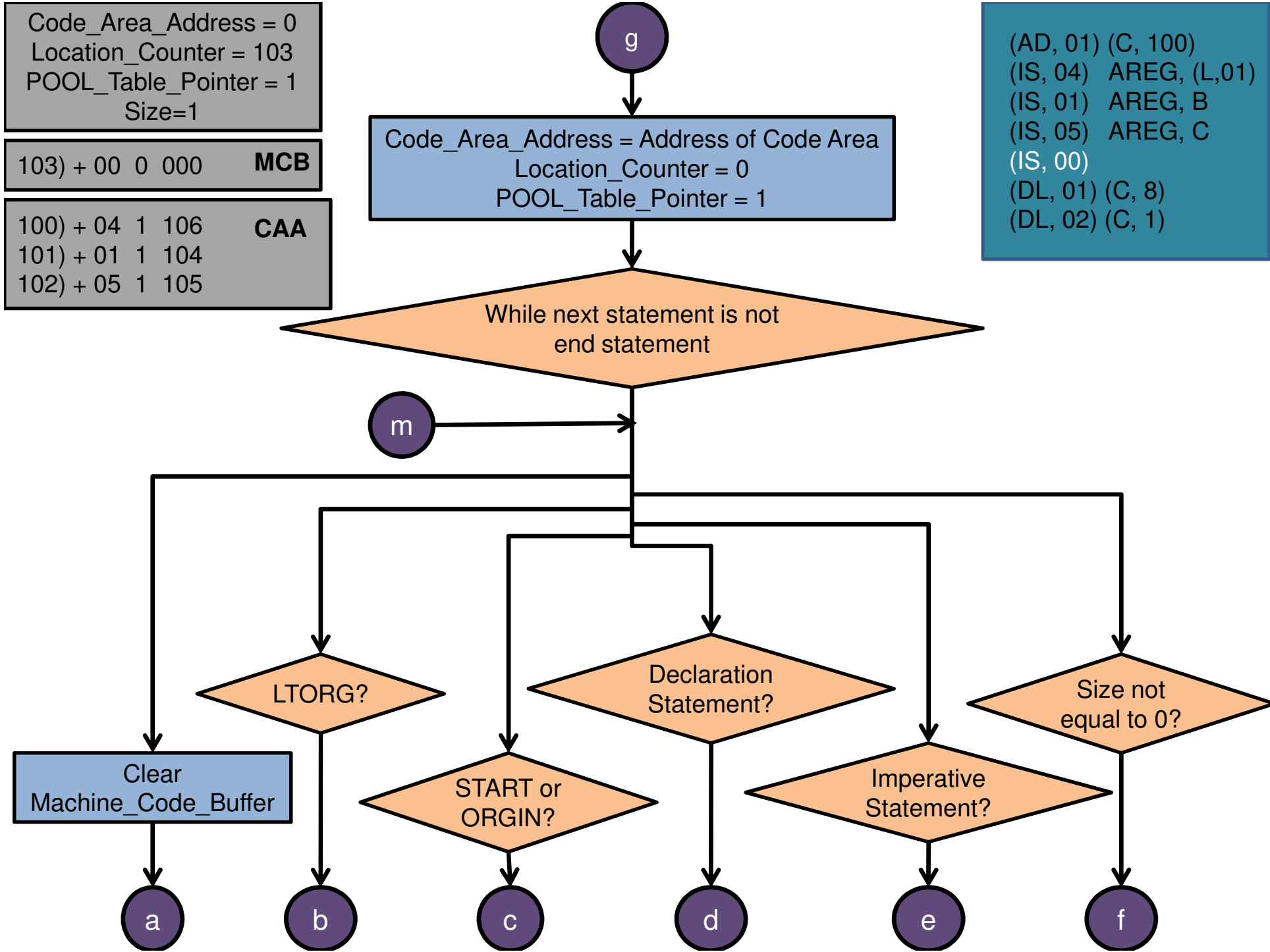
Code_Area_Address = 0
Location_Counter = 103
POOL_Table_Pointer = 1
Size=1

103) + 00 0 000 **MCB**

100) + 04 1 106 **CAA**
101) + 01 1 104
102) + 05 1 105

Code_Area_Address = Address of Code Area
Location_Counter = 0
POOL_Table_Pointer = 1

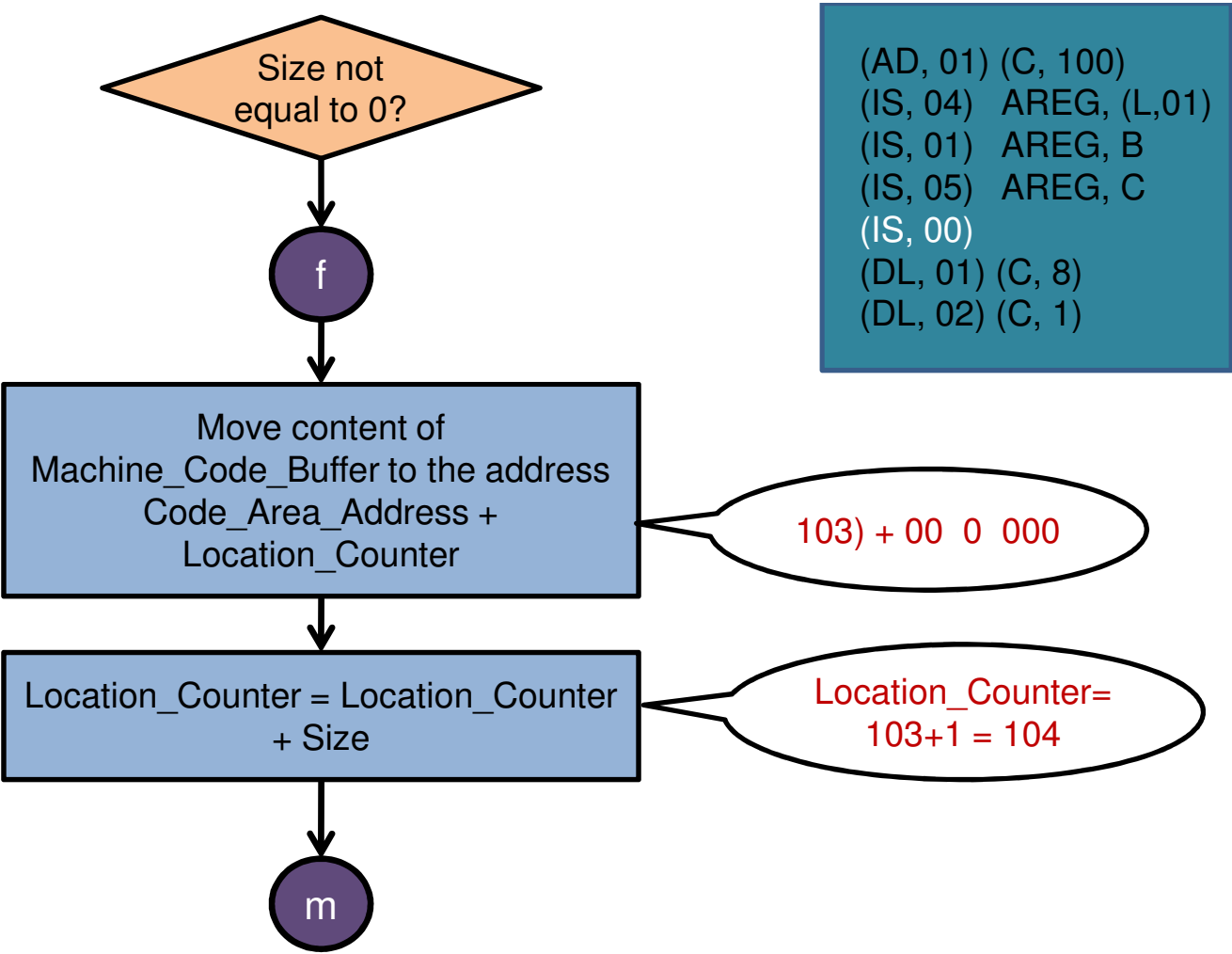
(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 103	
POOL_Table_Ponter = 1	
Size=1	

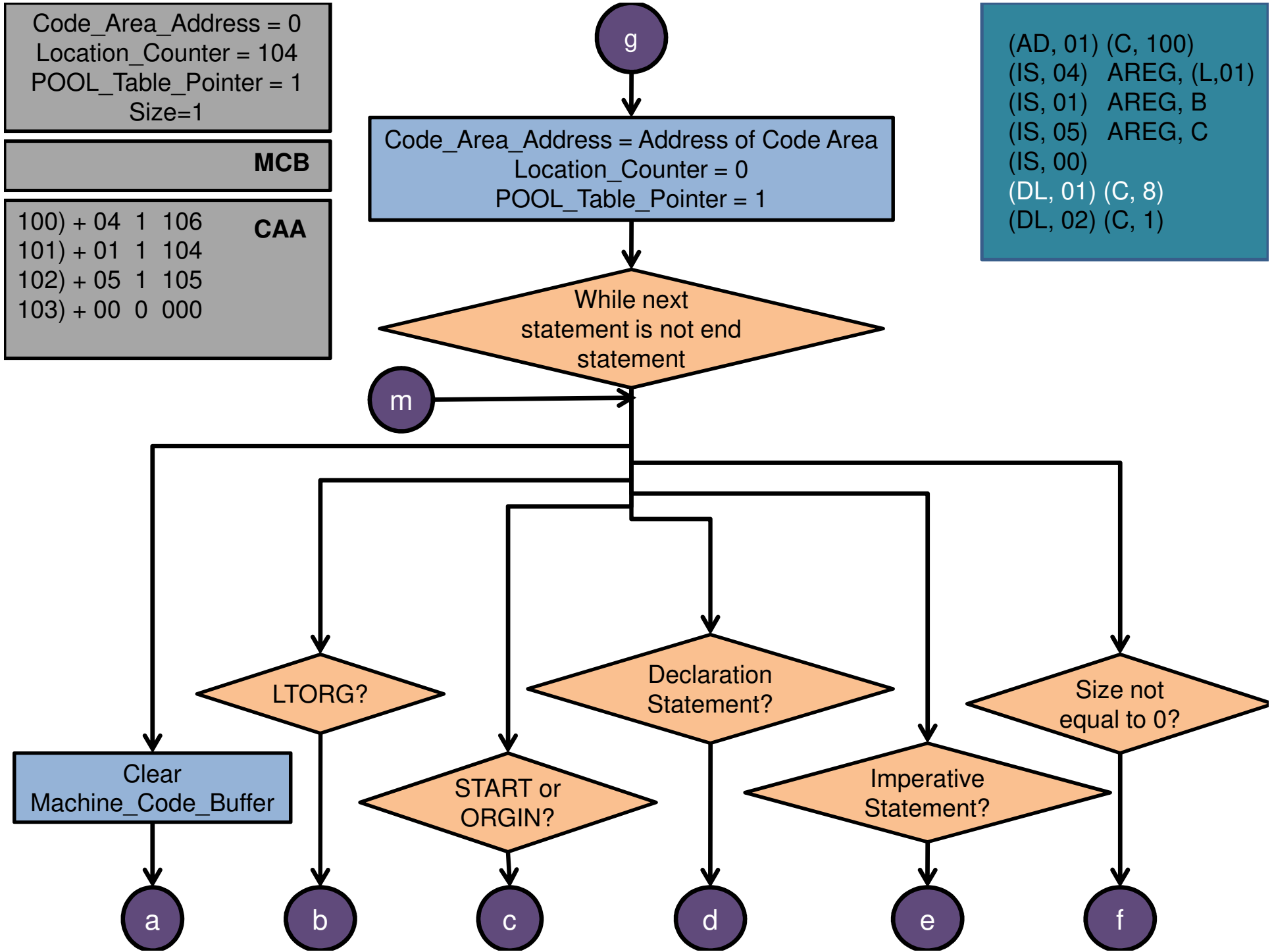
103) + 00 0 000	MCB
-----------------	------------

100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	
103) + 00 0 000	



Code_Area_Address = 0 Location_Counter = 104 POOL_Table_Pointer = 1 Size=1	
MCB	
100) + 04 1 106 101) + 01 1 104 102) + 05 1 105 103) + 00 0 000	CAA

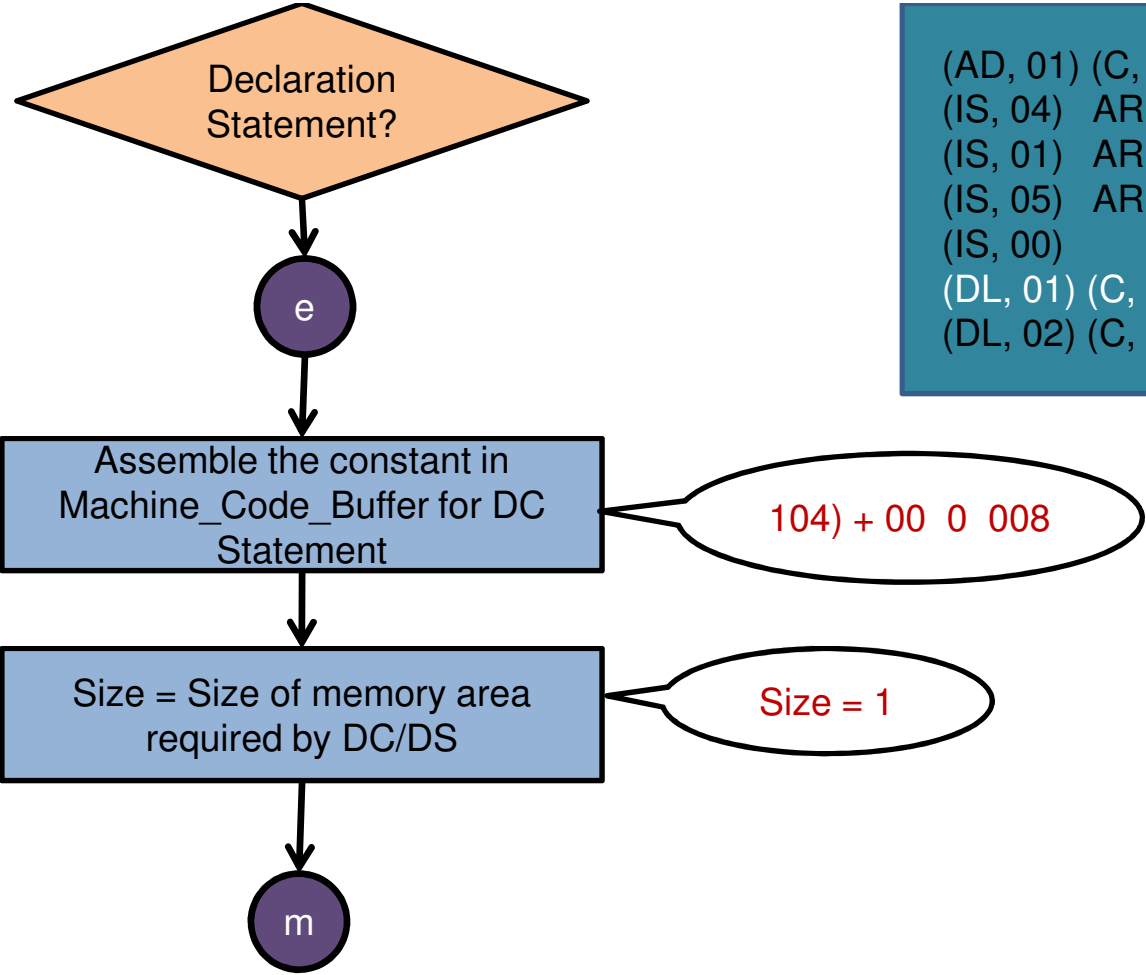
(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 104	
POOL_Table_Pointer = 1	
Size=1	

104) + 00 0 008	MCB
-----------------	------------

100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	
103) + 00 0 000	



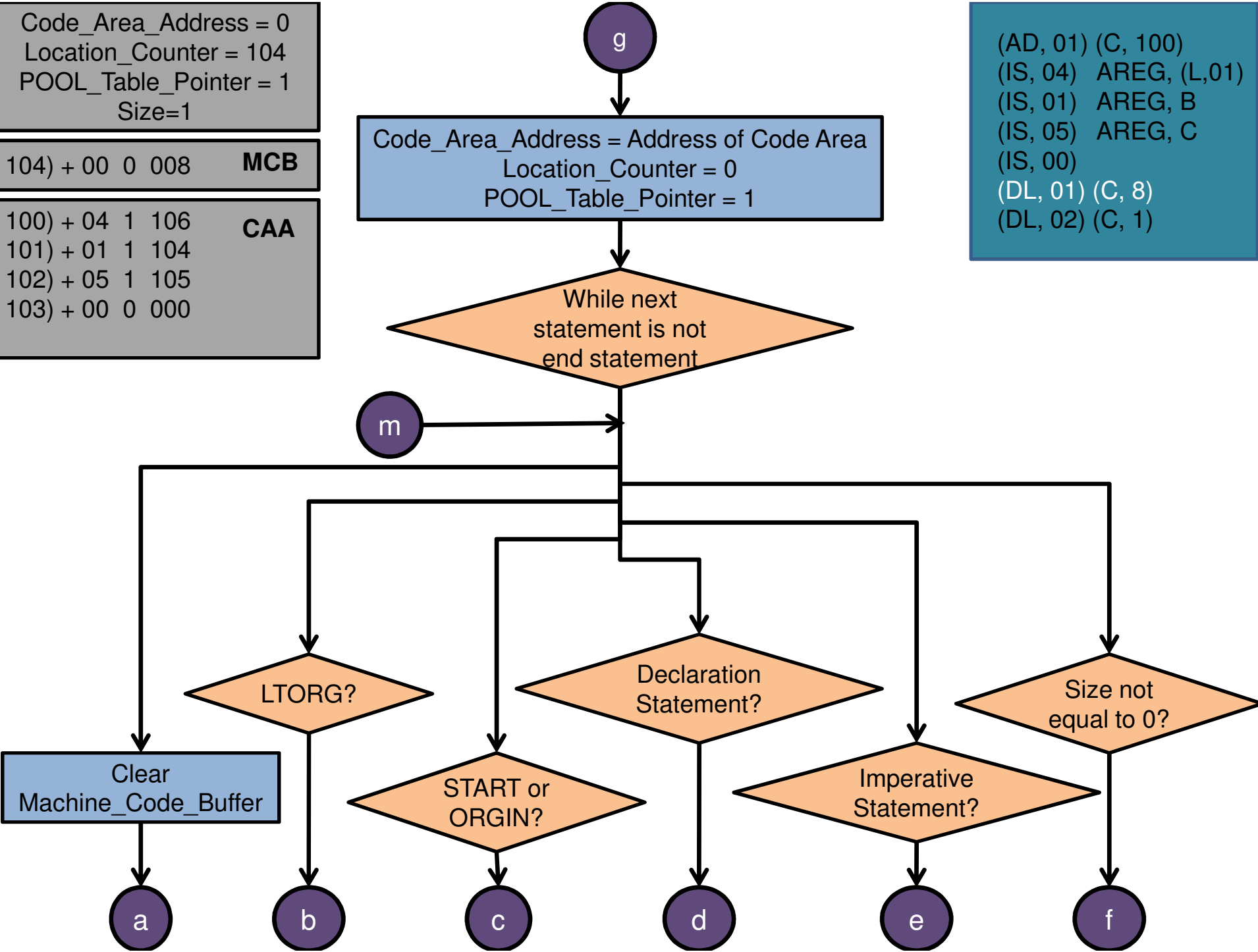
(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)

Code_Area_Address = 0	
Location_Counter = 104	
POOL_Table_Pointer = 1	
Size=1	

104) + 00 0 008	MCB
-----------------	------------

100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	
103) + 00 0 000	

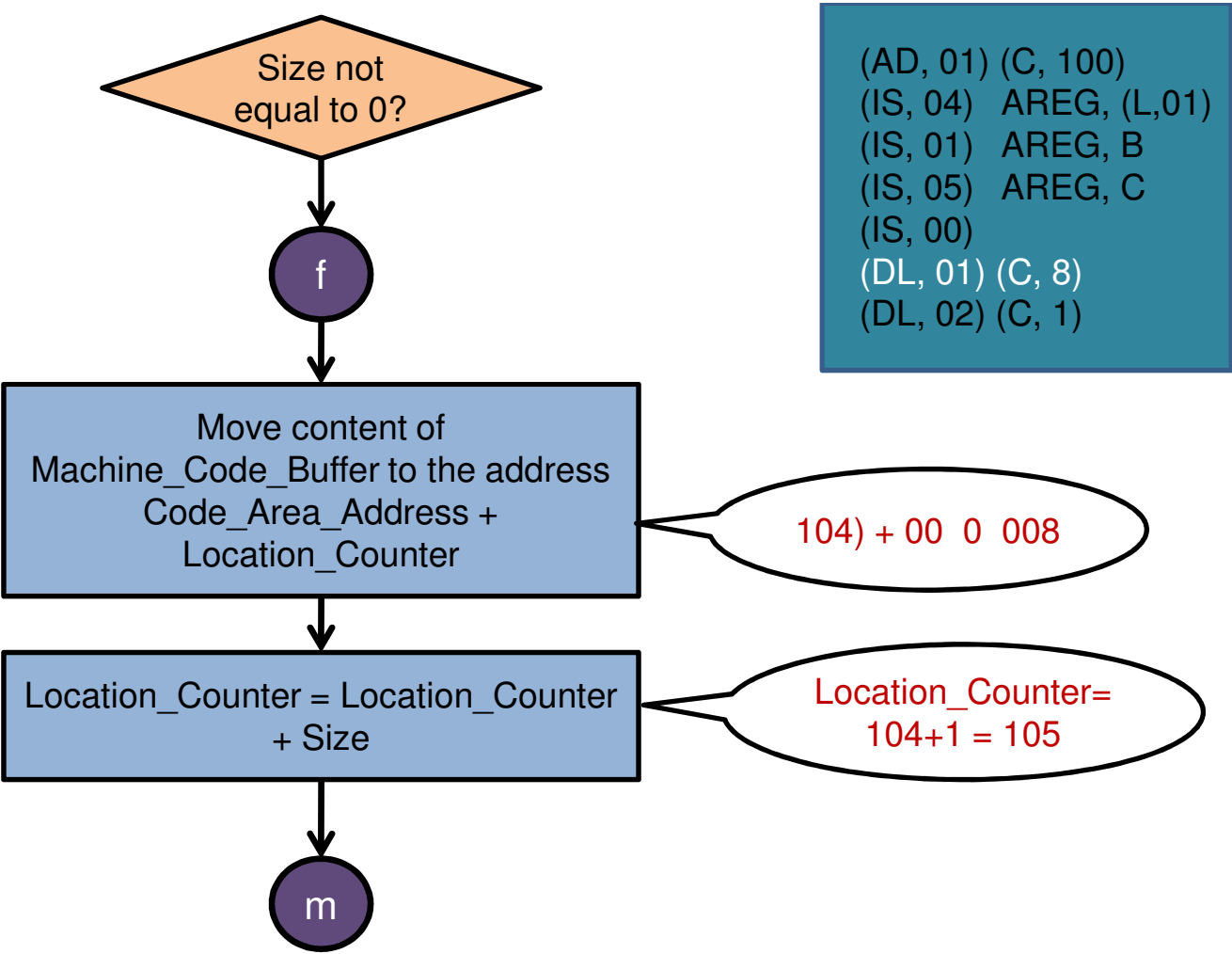
(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



Code_Area_Address = 0	
Location_Counter = 104	
POOL_Table_Ponter = 1	
Size=1	

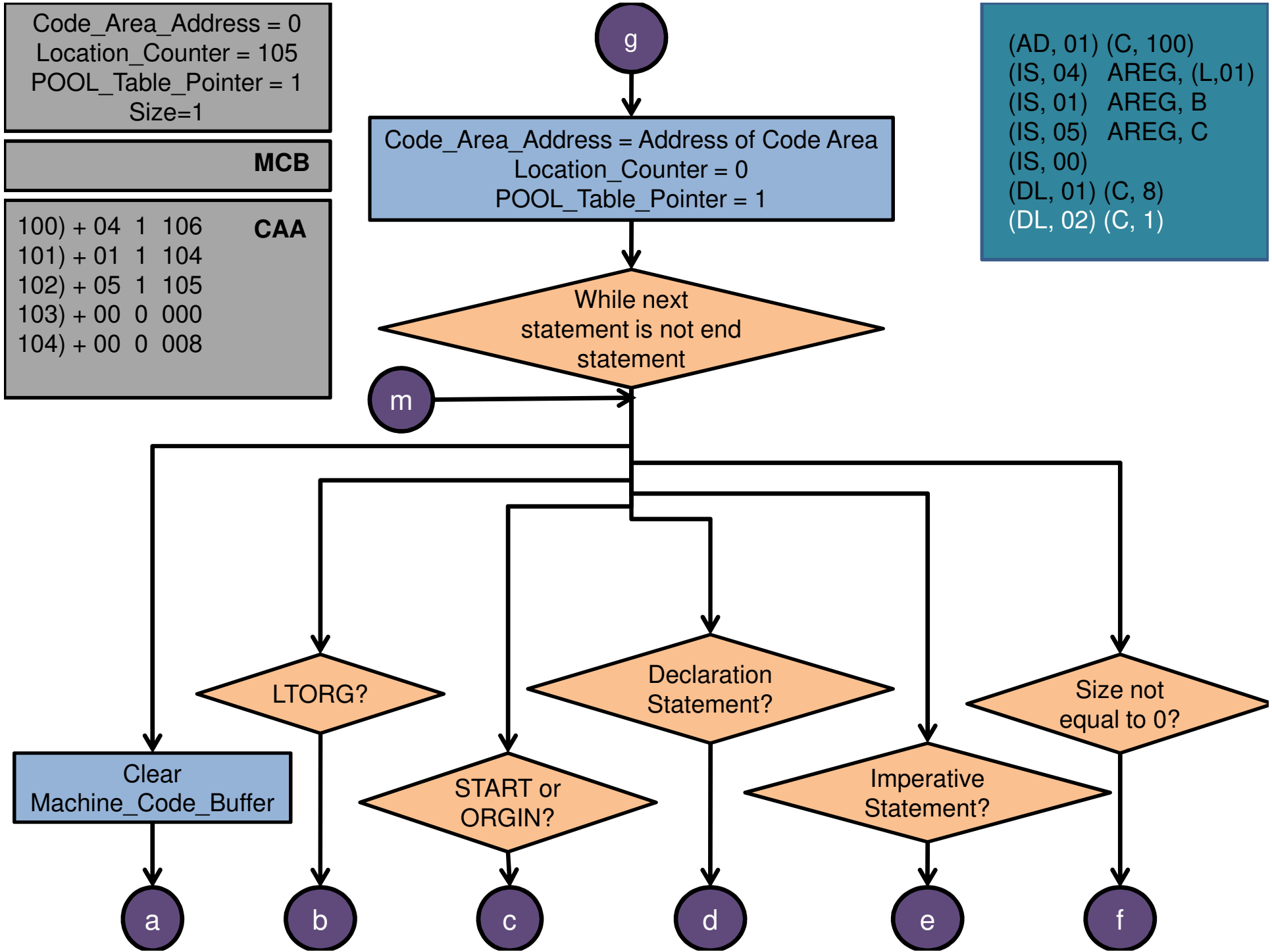
104) + 00 0 008	MCB
-----------------	------------

100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	
103) + 00 0 000	
104) + 00 0 008	

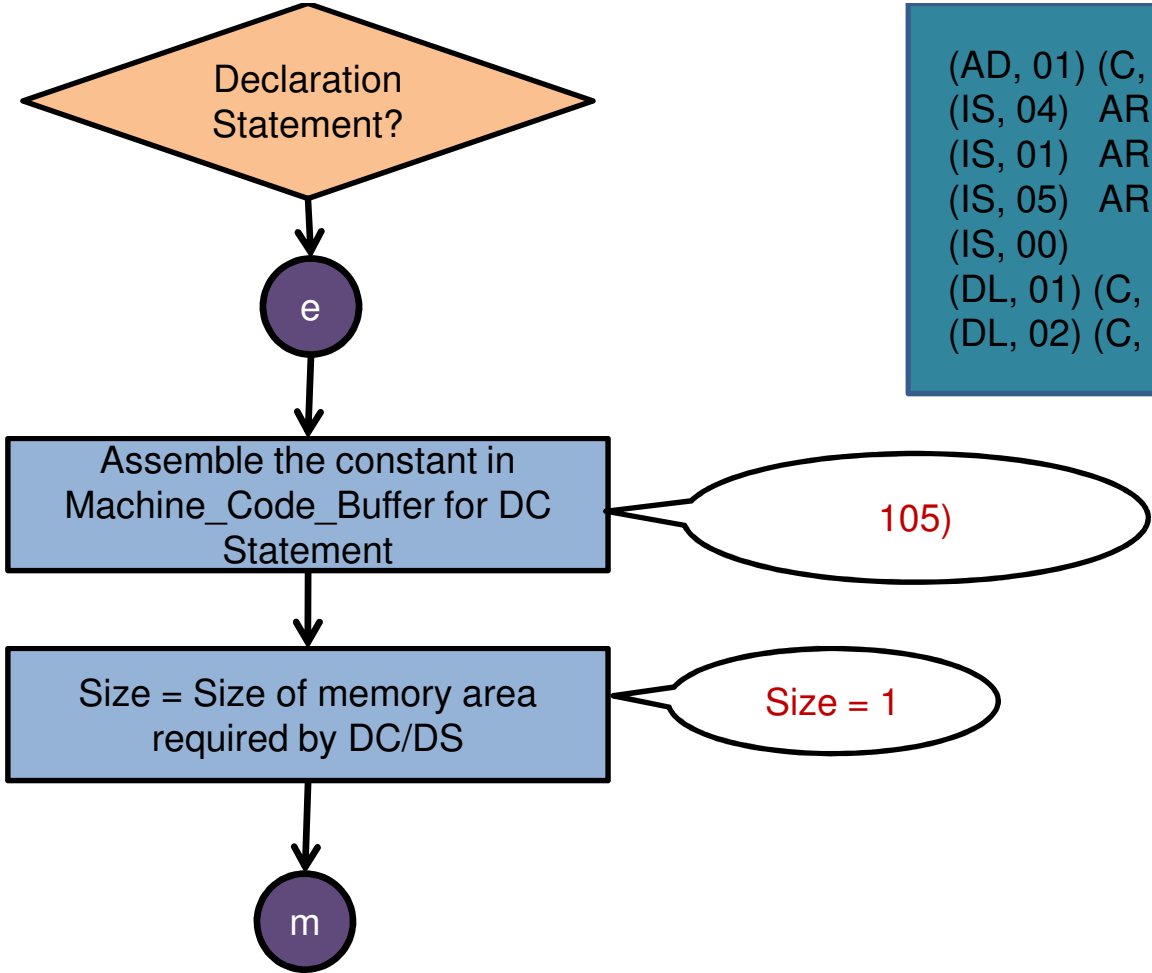


Code_Area_Address = 0 Location_Counter = 105 POOL_Table_Pointer = 1 Size=1	
MCB	
100) + 04 1 106 101) + 01 1 104 102) + 05 1 105 103) + 00 0 000 104) + 00 0 008	CAA

(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



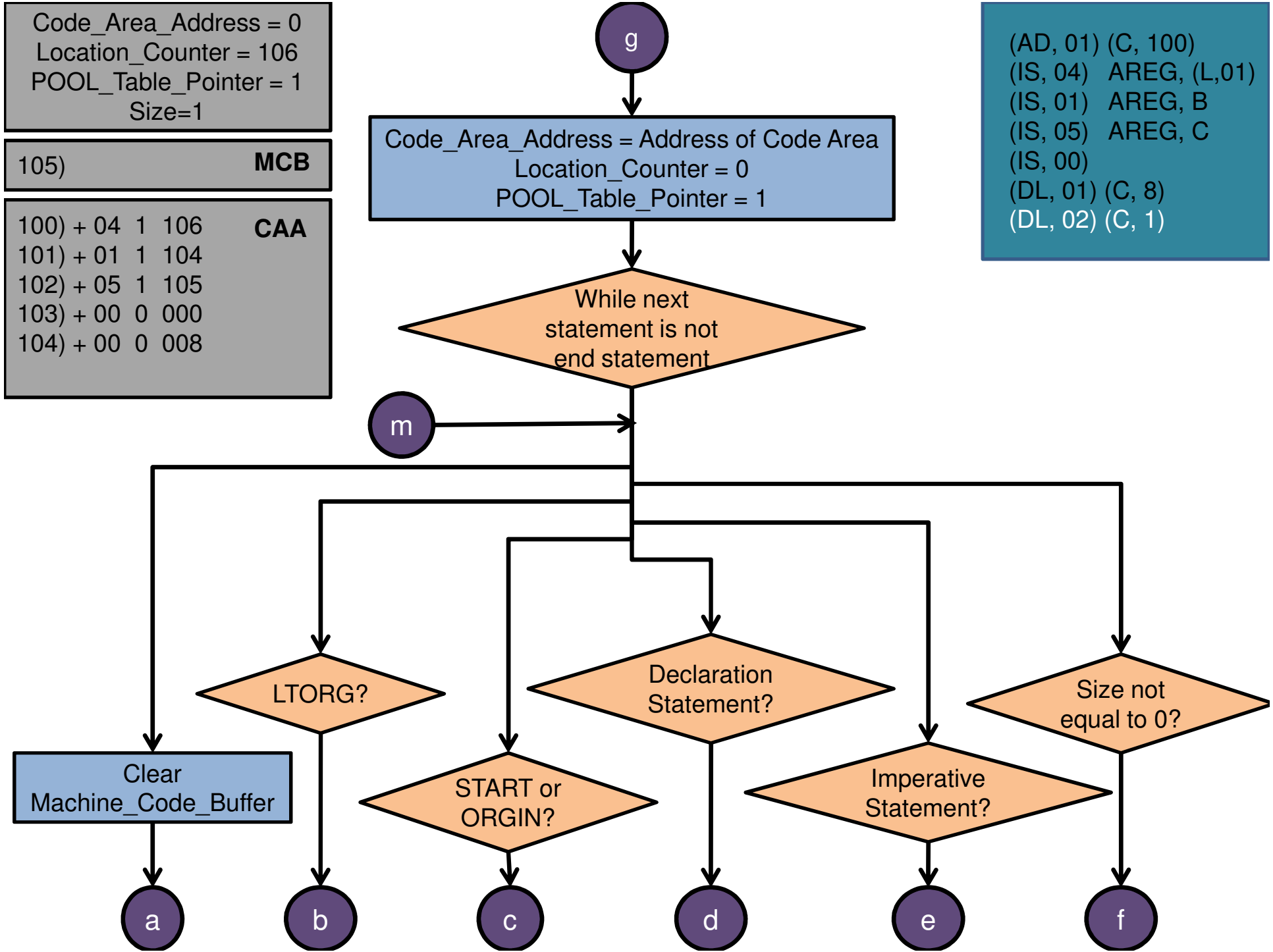
Code_Area_Address = 0		
Location_Counter = 105		
POOL_Table_Pointer = 1		
Size=1		
105)	MCB	
100) + 04 1 104	CAA	
101) + 01 1 105		
102) + 05 1 106		
103) + 00 0 000		
104) + 00 0 008		



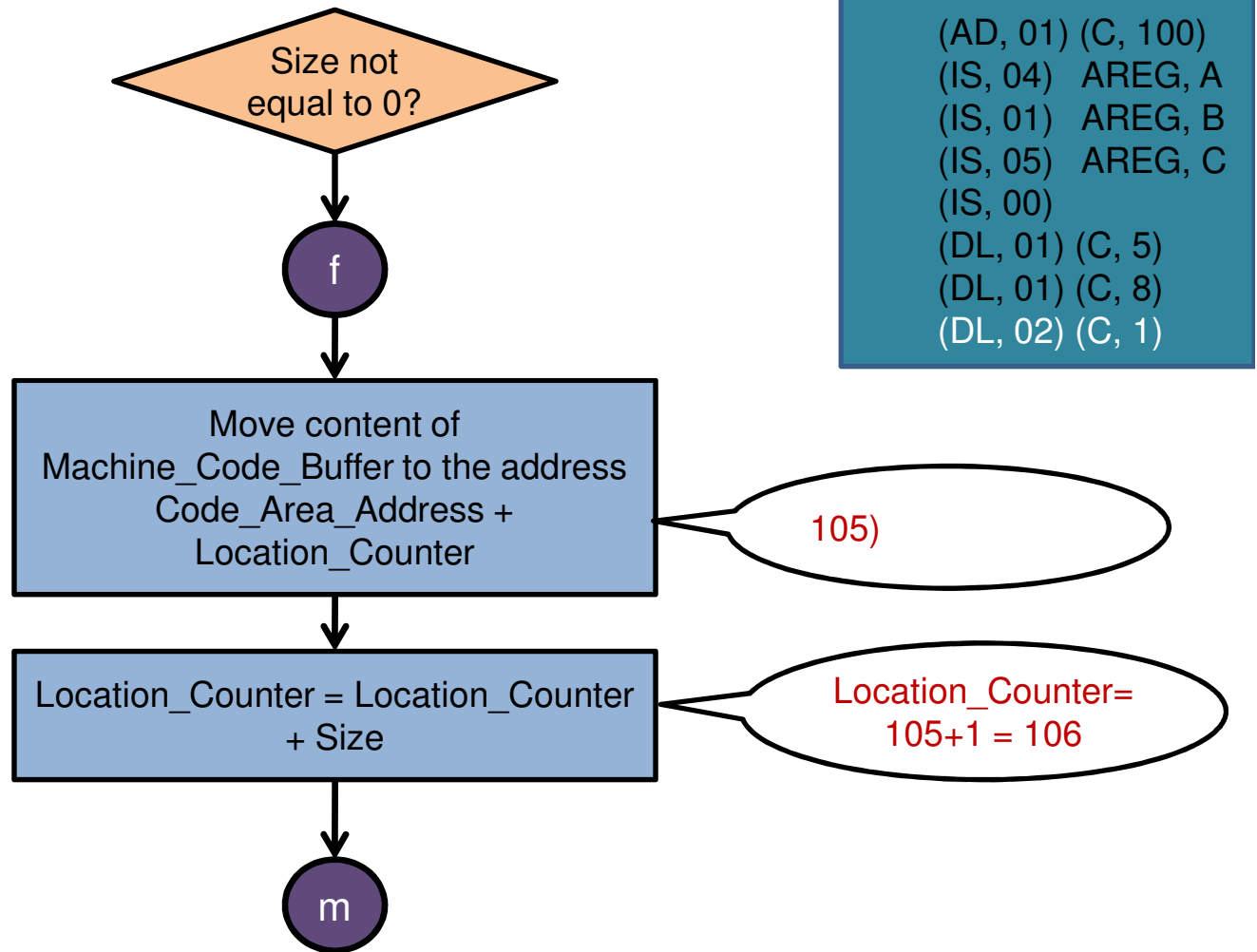
(AD, 01)	(C, 100)
(IS, 04)	AREG, (L,01)
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)

Code_Area_Address = 0 Location_Counter = 106 POOL_Table_Pointer = 1 Size=1	
105)	MCB
100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	
103) + 00 0 000	
104) + 00 0 008	

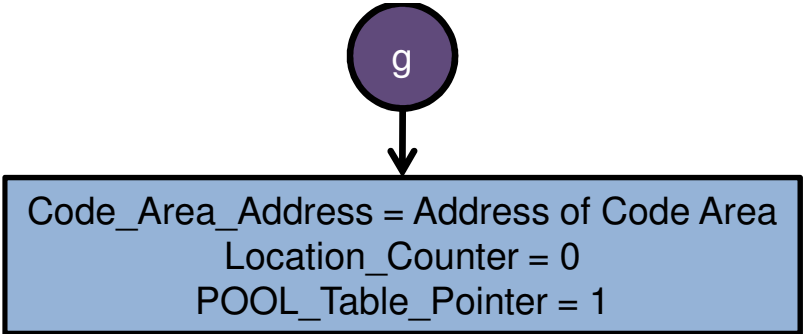
(AD, 01) (C, 100)
(IS, 04) AREG, (L,01)
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 8)
(DL, 02) (C, 1)



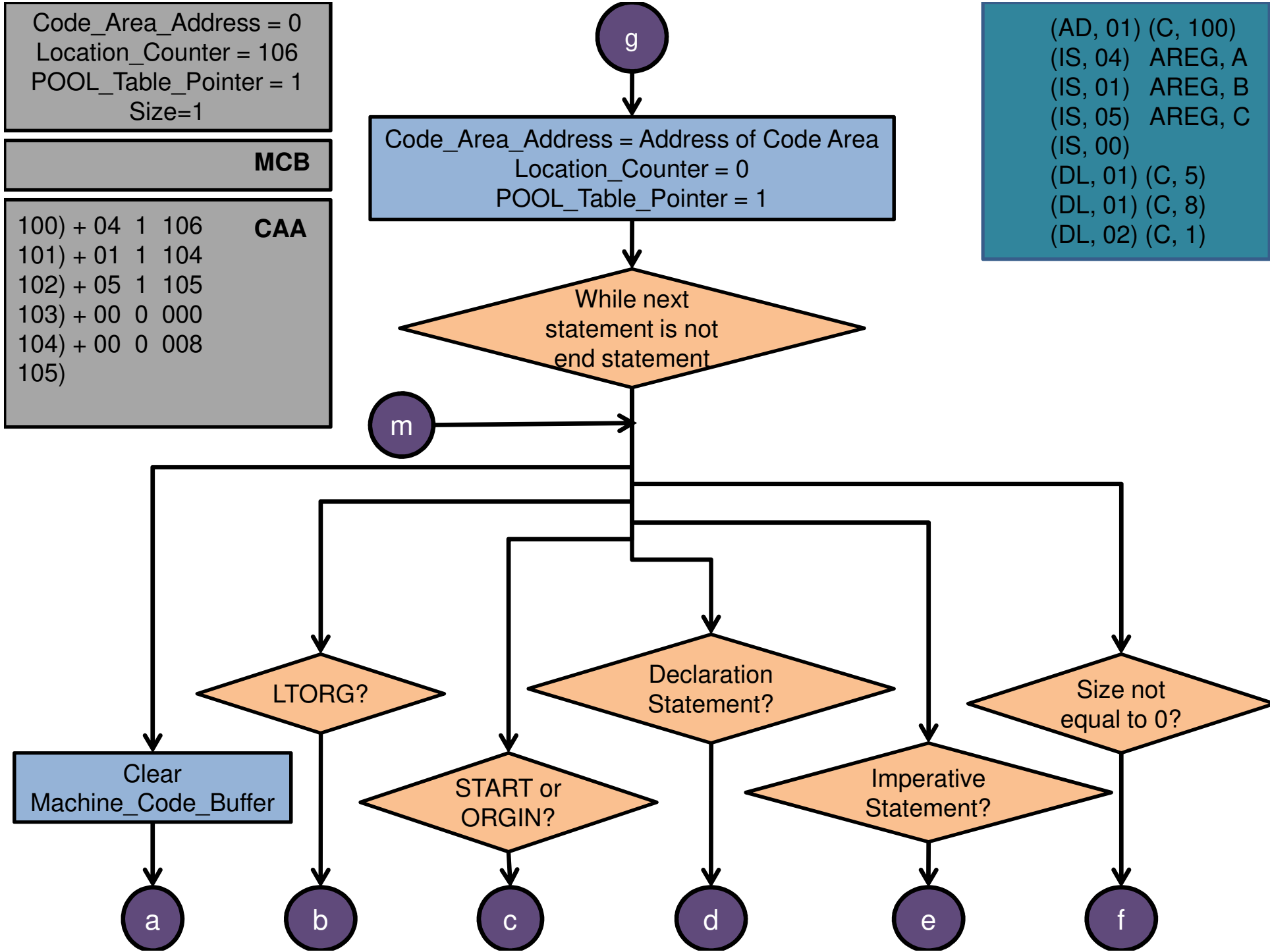
Code_Area_Address = 0 Location_Counter = 105 POOL_Table_Pointer = 1 Size=1	
105)	MCB
100) + 04 1 106 101) + 01 1 104 102) + 05 1 105 103) + 00 0 000 104) + 00 0 008 105)	CAA

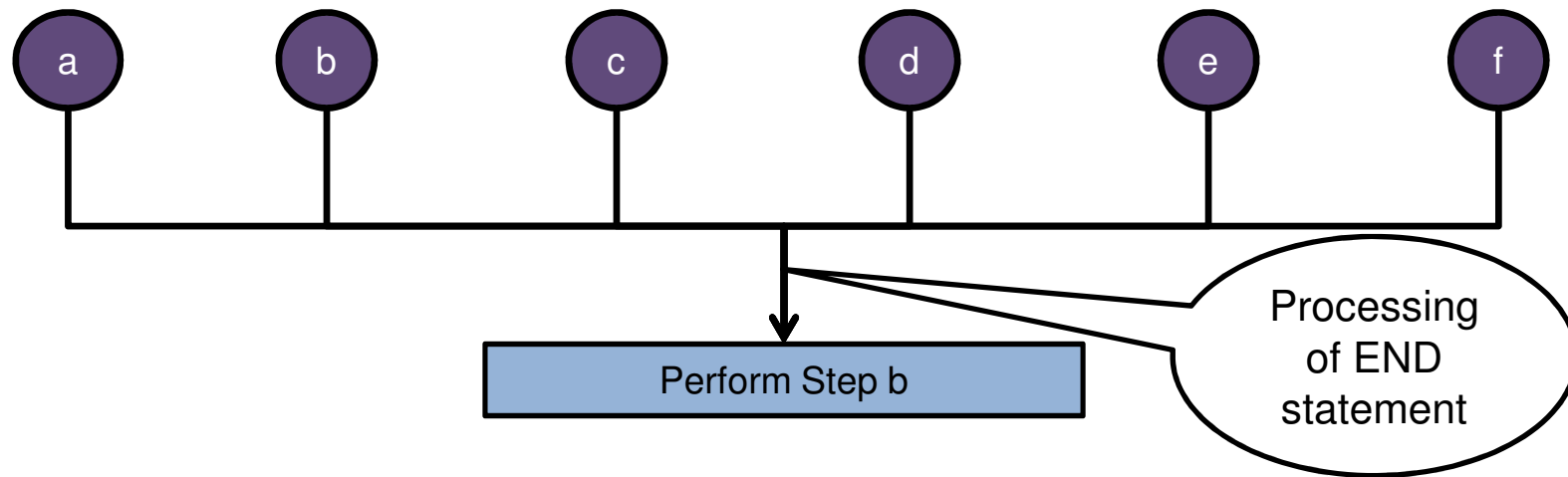


Code_Area_Address = 0	
Location_Counter = 106	
POOL_Table_Pointer = 1	
Size=1	
MCB	
100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	
103) + 00 0 000	
104) + 00 0 008	
105)	



(AD, 01) (C, 100)
(IS, 04) AREG, A
(IS, 01) AREG, B
(IS, 05) AREG, C
(IS, 00)
(DL, 01) (C, 5)
(DL, 01) (C, 8)
(DL, 02) (C, 1)





Code_Area_Address = 0
 Location_Counter = 106
 POOL_Table_Pointer = 1
 Size=1

MCB

CAA

100) + 04 1 106
 101) + 01 1 104
 102) + 05 1 105
 103) + 00 0 000
 104) + 00 0 008
 105) 8/13/2015

(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Code_Area_Address = 0
 Location_Counter = 106
 POOL_Table_Pointer = 1
 Size=1

106) + 00 0 005 **MCB**

CAA
 100) + 04 1 106
 101) + 01 1 104
 102) + 05 1 105
 103) + 00 0 000
 104) + 00 0 008
 105)



(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Process literals
 LITERAL_TABLE[POOL_TABLE[POOL_Table_Pointer]]
 ...LITERAL_TABLE[Literal_Table_Pointer - 1]] similar to
 processing of constants in a DC statement i.e.
 assemble the literals' in Machine_Code_Buffer

106) + 00 0
 005

Size = 1

Size = size of memory area
 required for literals

Literal_ Table_P ointer	Literal	Address
1	= '5'	106
2		

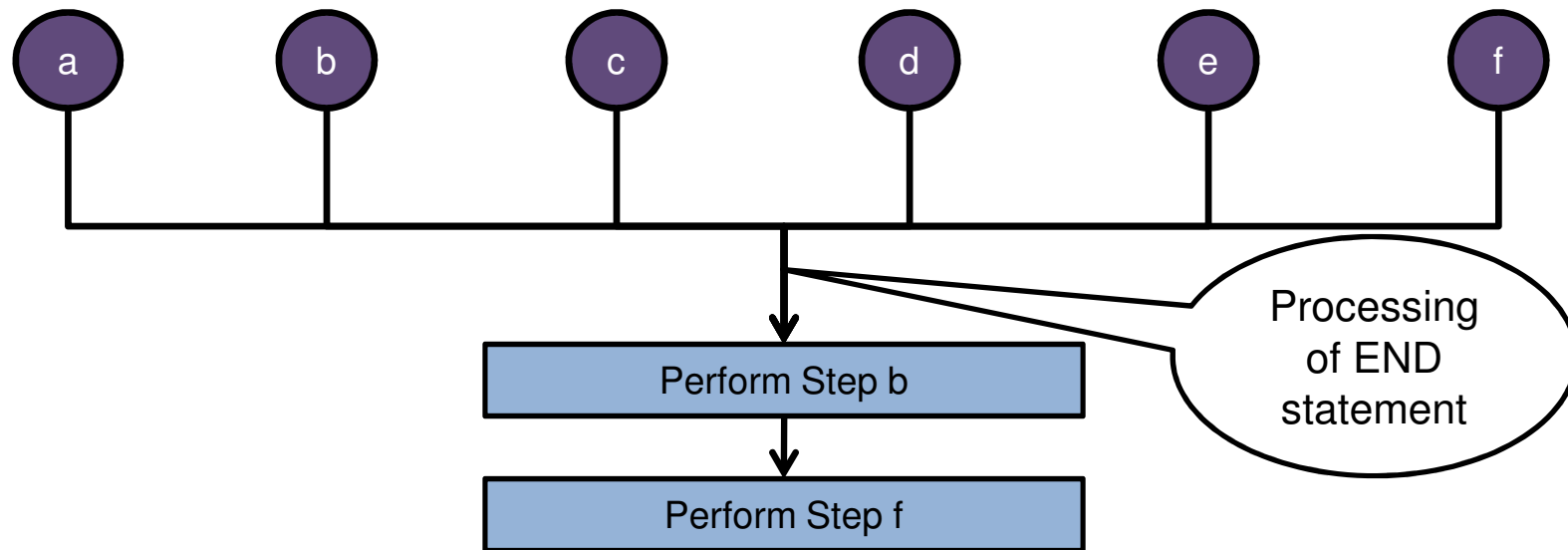
Literal Table

POOL_Table_Pointer
 = 1+1 =2

POOL_Table_Pointer =
 POOL_Table_Pointer + 1

POOL_Table_ Pointer	Literal_Table_ Pointer
1	1
2	2

POOL Table



Code_Area_Address = 0
 Location_Counter = 106
 POOL_Table_Pooter = 2
 Size=1

106) + 00 0 005 **MCB**

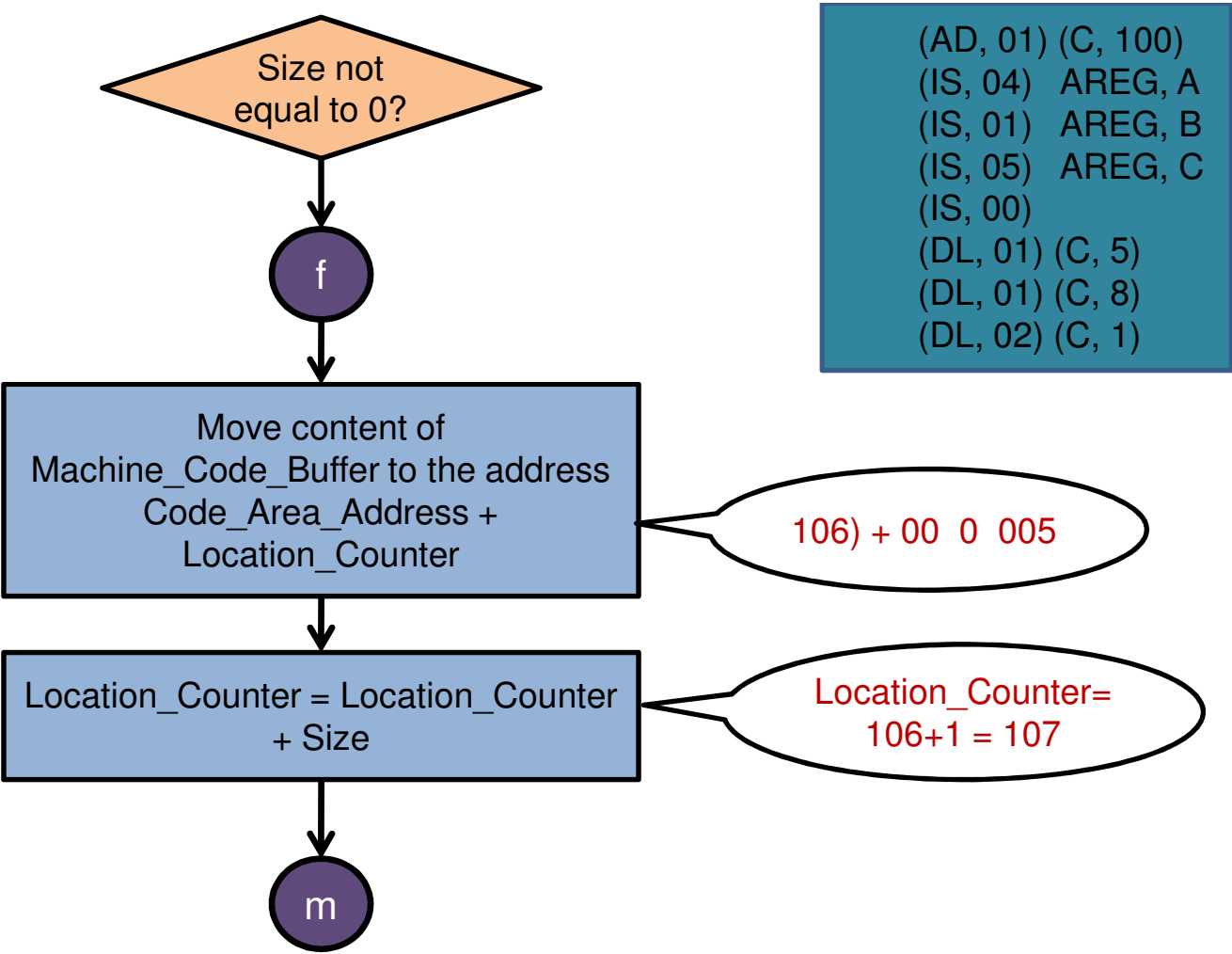
CAA
 100) + 04 1 106
 101) + 01 1 104
 102) + 05 1 105
 103) + 00 0 000
 104) + 00 0 008
 105) 8/13/2015

(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

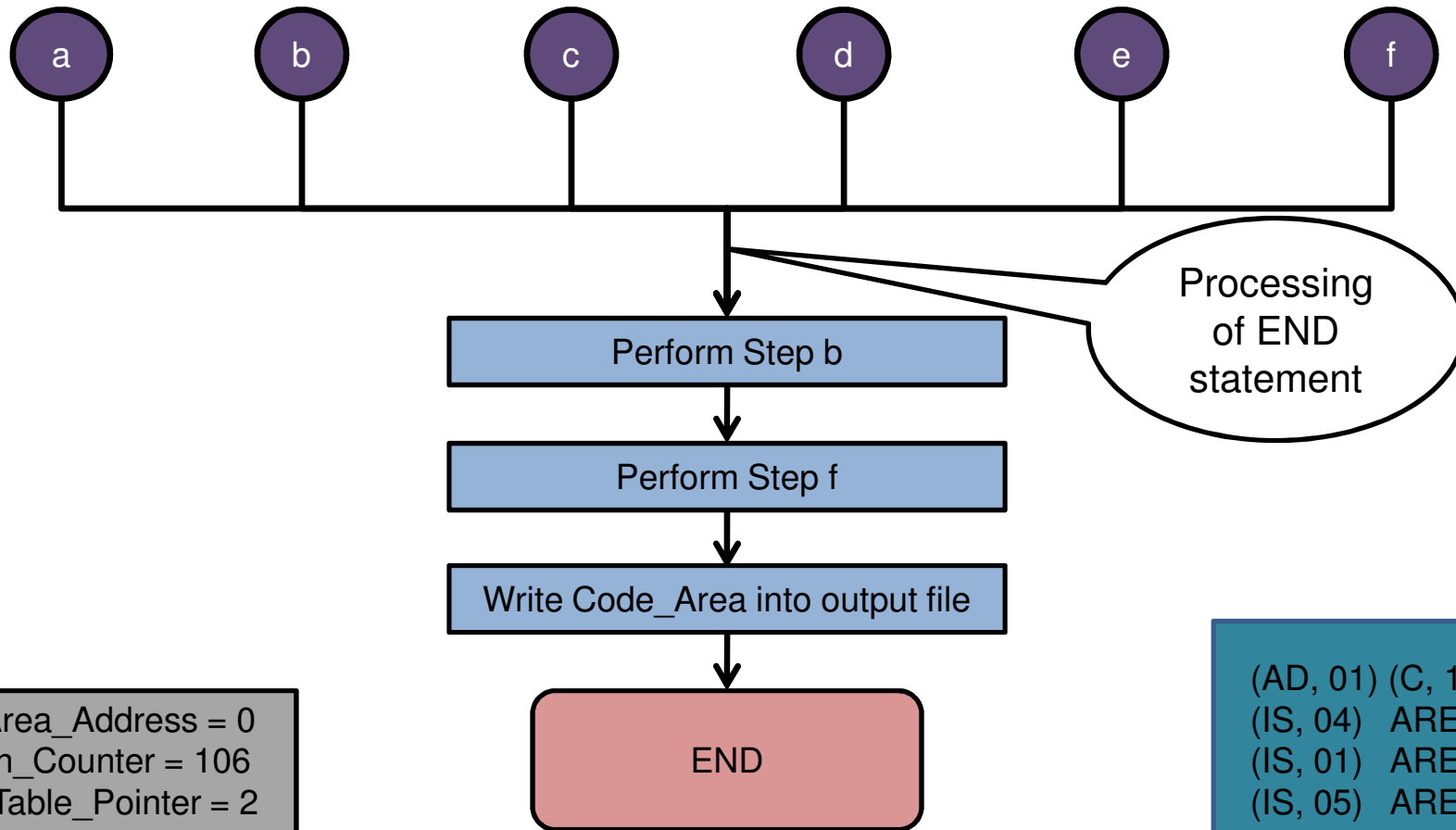
Code_Area_Address = 0			
Location_Counter = 105			
POOL_Table_Pointer = 1			
Size=1			

106) + 00 0 005	MCB
-----------------	------------

100) + 04 1 106	CAA
101) + 01 1 104	
102) + 05 1 105	
103) + 00 0 000	
104) + 00 0 008	
105)	
106) + 00 0 005	



(AD, 01)	(C, 100)
(IS, 04)	AREG, A
(IS, 01)	AREG, B
(IS, 05)	AREG, C
(IS, 00)	
(DL, 01)	(C, 5)
(DL, 01)	(C, 8)
(DL, 02)	(C, 1)



Code_Area_Address = 0
 Location_Counter = 106
 POOL_Table_Pointer = 2
 Size=1

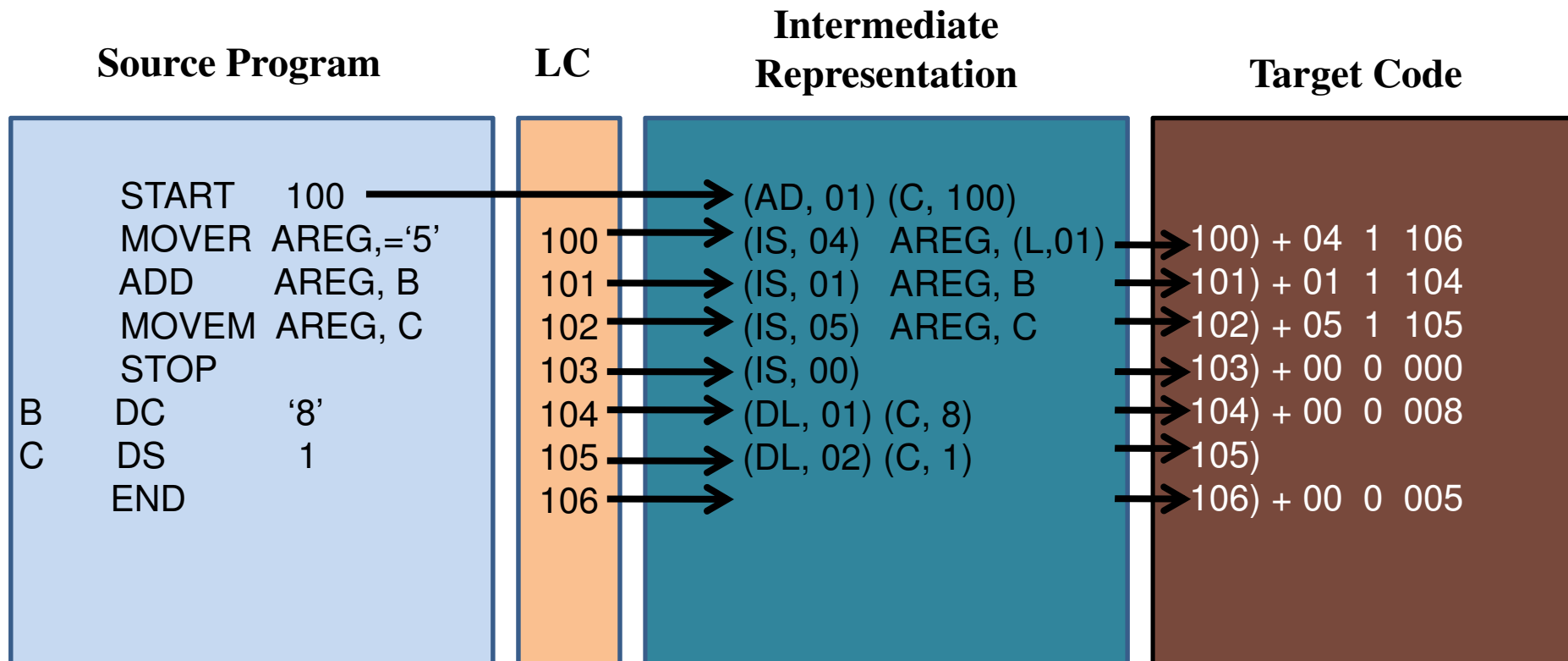
MCB

100) + 04 1 106 **CAA**
 101) + 01 1 104
 102) + 05 1 105
 103) + 00 0 000
 104) + 00 0 008
 105)
 106) + 00 0 005

100) + 04 1 104
 101) + 01 1 105
 102) + 05 1 106
 103) + 00 0 000
 104) + 00 0 005
 105) + 00 0 008
 106)
 107) + 00 0 005

(AD, 01) (C, 100)
 (IS, 04) AREG, (L,01)
 (IS, 01) AREG, B
 (IS, 05) AREG, C
 (IS, 00)
 (DL, 01) (C, 8)
 (DL, 02) (C, 1)

Output File



Literal_Table_P ointer	Literal	Address
1	= '5'	106
2		

Literal Table

POOL_Table_ Pointer	Literal_Table_ Pointer
1	1
2	2

POOL Table

Symbol	Address	Length
B	104	1
C	105	1

Symbol Table

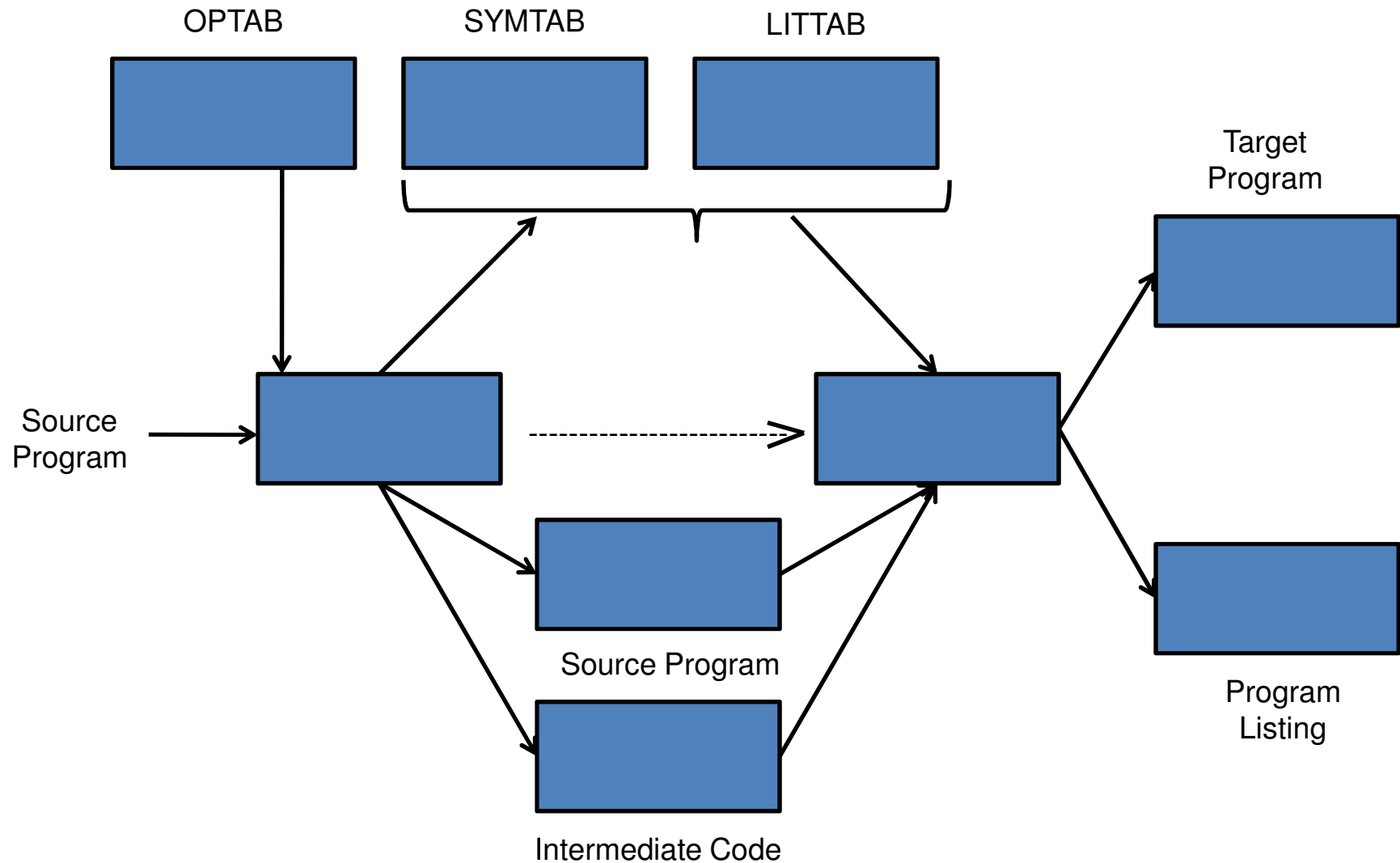
Listing and Error Reporting

Sr. No.	Statement			Address
1	START	200		
2	MOVER	AREG,A		200
		:		
3				
9	MVER	BREG,A		207
	ERRORINVALID OPCODE			
10	ADD	BREG,B		208
14	A	DS	1	209
		:		
15				
21	A	DC	'5'	227
	ERRORDUPLICATE DEFINATION OF SYMBOL A			
22		:		
35	END			
	ERRORUNDEFINED SYMBOL B IN STATEMENT 10			

Listing and Error Reporting

Sr. No.	Statement	Address	INSTRUCTION
1	START 200		
2	MOVER AREG,A	200	+04 1 209
3	:		
9	MVER BREG,A	207	+ -- 2 209
	ERRORINVALID OP CODE		
10	ADD BREG,B	208	+01 2 ---
	ERRORUNDEFINED SYMBOL B IN OPERAND FIELD		
14	A DS 1	209	
15	:		
21	A DC '5'	227	+00 0 005
	ERRORDUPLICATE DEFINATION OF SYMBOL A		
22	:		
35	END		

Some Organizational Issues



2. Assemblers

- Elements of Assembly Language Programming
- A Simple Assembly Scheme
- Pass Structure of Assemblers
- Design of a Two Pass Assembler
- A Single Pass Assembler for IBM PC

Architecture of Intel 8088

CPU contains following features:

- Data Registers AX, BX, CX and DX
- Index Registers SI and DI
- Stack Pointers Registers SP and BP
- Segment registers Code Stack, Data and Extra

Architecture of Intel 8088

Data Register

AH	AL
BH	BL
CH	CL
DH	DL

Base Register

SP
BP

Index Register

SI
DI

Segment Register

CODE
STACK
DATA
EXTRA

Addressing Modes

Addressing Modes	Examples	Remarks
Immediate	MOV SUM, 1234H	Data = 1234H
Register	MOV SUM, AX	AX contains data
Direct	MOV SUM, [1234H]	Data Displacement = 1234H
Indirect	MOV SUM, [BX]	Data Displacement = (BX)
Register Indirect	MOV SUM, CS: [BX]	Segment Override: Segment Base = (CS) Data Displacement (BX)
Based	MOV SUM, 12H[BX]	Data Displacement = 12H + (BX)
Indexed	MOV SUM, 34H[SI]	Data Displacement = 34H + (SI)
Based and Indexed	MOV SUM, 56H[SI][BX]	Data Displacement = 56H + (SI) + (BX)

Assembly Language of Intel 8088

Statement Format:

[Label:] opcode operand(S) ; comment string

Assembly Language of Intel 8088

- Assembler Directives
 - ORG
 - EQU
 - END

Assembly Language of Intel 8088

- Declarations:
 - DB
(e.g. A DB 25 ; Reserve byte and initialize)
 - DW
(e.g. B DW ? ; Reserve byte and no initialization)
 - DD
(e.g. A DD 6DUP(0) ; 6 Double words, all 0's)
 - DQ
 - DT

Assembly Language of Intel 8088

- EQU and PURGE

e.g.

```
XYZ DB      ?
```

```
ABC EQU     XYZ ; ABC represent name XYZ
```

```
      PURGE ABC ; ABC no longer XYZ
```

```
ABC EQU     25  ; ABC stands for '25'
```


Assembly Language of Intel 8088

- SEGMENT, ENDS and ASSUME
 - SEGMENT and ENDS directives demarcate the segments in assembly language.
 - ASSUME tells the assembler that it can assume the address of indicated segment to be present in <register>
ASSUME <register> : <segment name>

Assembly Language of Intel 8088

```
SAMPLE_DATA  SEGMENT
ARRAY        DW      100 DUP ?
SUM          DW      0
SAMPLE_DATA  ENDS
SAMPLE_CODE  SEGMENT
              ASSUME  DS: SAMPLE_DATA
HERE         MOV     AX, SAMPLE_DATA
              MOV     DS, AX
              MOV     AX, SUM
              -----
SAMPLE_CODE  ENDS
              END     HERE
```

Assembly Language of Intel 8088

- PROC, ENDP, NEAR and FAR

e.g.

```
SAMPLE_CODE    SEGMENT
CALCULATE      PROC      FAR
               -----
               RET
CALCULATE      ENDP
SAMPLE_CODE    ENDS
PGM            SEGMENT
               -----
               CALL
               -----
               ENDS
               END
```

Assembly Language of Intel 8088

- PUBLIC and EXTRN
 - PUBLIC : when a symbolic name declared in one assembly module is to be accessible in other module, it is specified in a PUBLIC statement
 - EXTRN : Another module wishing to use this name must specify in an EXTRN statement

Assembly Language of Intel 8088

- Analytic operator
 - SEG
 - OFFSET
 - TYPE
 - SIZE
 - LENGTH

Assembly Language of Intel 8088

- Synthetic operator
 - PTR creates new memory operand with the same segment and offset addresses as an existing operand but having a different type.
 - THIS performs the special function of creating a new memory operand with the same address as the next memory byte available for allocation

Assembly Language of Intel 8088

Example

```
XYZ            DW      312
NEW_NAME      EQU     BYTE PTR XYZ
LOOP:          CMP     AX, 234
               JMP     LOOP
FAR_LOOP      EQU     FAR PTR LOOP
               JMP     FAR_LOOP
```

Assembly Language of Intel 8088

Example

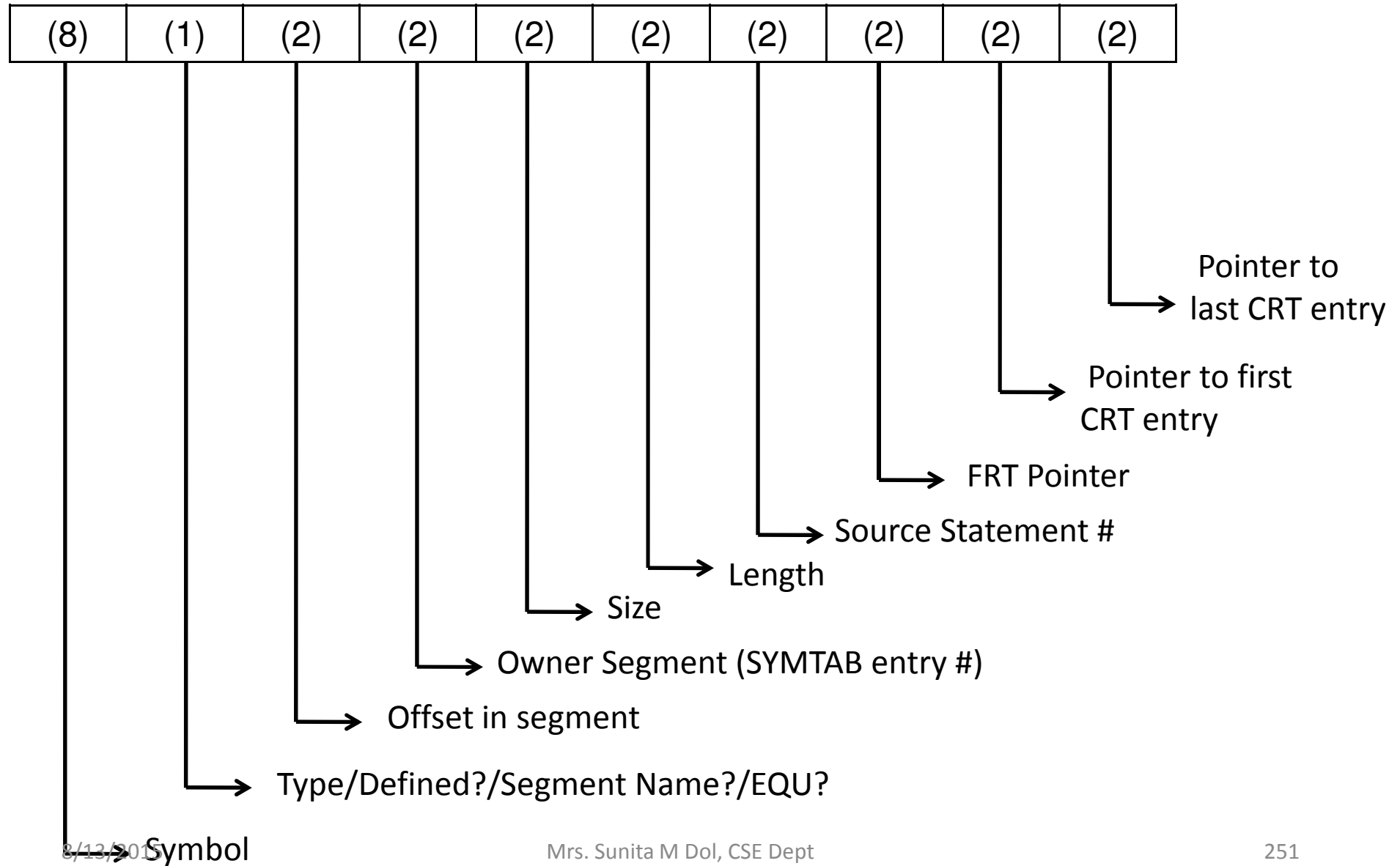
```
                                DW
NEW_NAME EQU THIS BYTE
XYZ      DW      312
FAR_LOOP EQU THIS FAR
LOOP     CMP     AX, 234
        JMP     LOOP
-----
        JMP     FAR_LOOP
```


<u>Sr.No</u>			<u>Statement</u>	<u>Offset</u>
1	CODE	SEGMENT		
2		ASSUME	CS:CODE,DS:DATA	
3		MOV	AX,DATA	0
4		MOV	DS,AX	3
5		MOV	CX,LENGTH STRING	5
6		MOV	COUNT,0000	8
7		MOV	SI,OFFSET STRING	11
8		ASSUME	ES:DATA,DS:NOTHING	
9		MOV	AX,DATA	14
10		MOV	ES,AX	17
11	COMP:	SEGMENT	[SI],'A'	19
12		JNE	NEXT	22
13		MOV	COUNT,1	24
14	NEXT:	INC	SI	27
15		DEC	CX	29
16		JNE	COMP	30
17	CODE	ENDS		
18	DATA	SEGMENT		
19		ORG	1	
20	COUNT	DB	?	1
21	STRING	DW	50 Dup(?)	2
22	DATA	ENDS		
23		END		

Mnemonics table(MOT)

Mnemonic Opcode (6)	Machine Opcode (2)	Alignment/ Format info (1)	Routine Id (4)
JNE	75H	00H	R2

Symbol Table (SYMTAB)



Segment register table array (SRTAB_ARRAY)

Segment register (1)	Segment name (2)		
00(ES)	23		SRTAB#1
:			SRTAB#2

Forward reference table(FRT)

Pointer (2)	SRTAB # (1)	Instruction Address (2)	Usage Code (1)	Source Stmt# (2)
----------------	----------------	-------------------------------	----------------------	------------------------

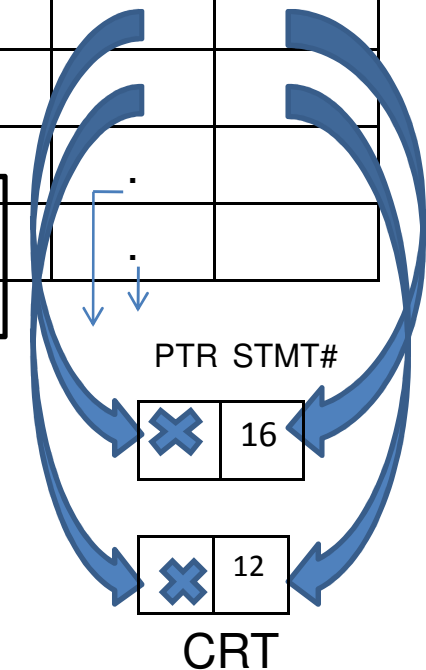
Cross Reference Table (CRT)

Pointer (2)	Source Stmt# (2)
----------------	---------------------

symbol	D ?	S ?	T	Offset	Owner segment	Length & size	FTR Pointer	CRT pointer	
								First	Last
CODE	Y	Y							
DATA	N	Y							
COMP	Y	N	-1	19	1		-		
NEXT	Y	N	-1	27	1		-		
COUNT	N	N					.		
STRING	N	N					.		

PTR	SRTAB	ADDR	CODE	STMT#
✕	#1	0008	D	0006
	#2	0024	D	0013

✕	#1	0005	L	0005
	#1	0011	F	0007



01(CS)	1	SRTAB#1
11(DS)	2	
00 (ES)	2	SRTAB#2
01(CS)	1	

Single Pass Assembler

1. **code_area_address := address of code_area;**
srtab_no := 1;
LC := 0;
stmt_no := 1;
SYMTAB_segment_entry := 0;
Clear ERRTAB , SRTAB_ARRAY
2. **While next statement is not an END statement**
 - a) **Clear machine_code_buffer.**
 - b) **If label is present then**
this_label := symbol in the label field;

c) If an EQU statement

- i) `this_address := value of operand expression;`
- ii) Make an entry for `this_label` in SYMTAB with
 - `offset := this_address;`
 - `defined := 'yes';`
 - `owner_segment := m owner_segment of operand symbol;`
 - `source_stmt# := stmt_no;`
- iii) Enter `stmt_no` in the CRT list of the label in the operand field.
- iv) Process forward references to `this_label`;
- v) `size := 0;`

d) If an ASSUME statement

- i) Copy the SRTAB in SRTAB_ARRAY[srtab_no] into SRTAB_ARRAY[srtab_no + 1];
- ii) srtab_no := srtab_no + 1;
- iii) this_register := register mentioned in the statement.
- iv) this_segment := entry number of SYMTAB entry of the segment appearing in the operand field.
- v) Make the entry (this_register ; this_segment) in the SRTAB_ARRAY[srtab_no]. (This overwrites an existing entry for this register.)
- vi) size := 0;

e) If a SEGMENT statement

- i) Make an entry for this_label in SYMTAB.
- ii) Set segment name ? := true;
- iii) SYMTAB_segment_entry := entry no in SYMTAB;
- iv) LC := 0;
- v) size := 0;

f) If an ENDS statement then

SYMTAB_segment_entry := 0;

g) If a declaration statement

- i) Align LC according to the specification in the operand field.
- ii) Assemble the constant(s), if any, in the machine_code_buffer
- iii) size := size of memory area required;

h) If an imperative statement

- i) If operand is a symbol symb then
enter stmt_no in the CRT list of symb.
- ii) If operand symbol is already defined then
Check its alignment & addressability.
Generate the address specification (segment register, offset) for the symbol using its SYMTAB entry and SRTAB_ARRAY[srtab_no].
else
Make an entry for symbol in SYMTAB.
Defined := 'no';
Enter (srtab_no, LC, usage_code, stmt_no) in FRT.
- iii) Assemble the instruction in machine_code_buffer.
- iv) size := size of the instruction;

- i) If size $\neq 0$ then
 - i) If label is present then
 - Make an entry for this_label in SYMTAB.
 - owner_segment := SYMTAB_segment_entry;
 - Defined := 'yes' ;
 - offset := LC;
 - source_stmt# := stmt_no;
 - ii) Move contents of machine_code_buffer to the address code_area_address;
 - iii) code_area_address := code_area_address + size;
 - iv) Process forward references to the symbol. Check for alignment & addressability errors. Enter errors in ERRTAB.
 - v) List the statement with errors contained in ERRTAB.
 - vi) Clear ERRTAB.

3) (Processing of END statement)

- a) Report undefined symbol from SYMTAB.**
- b) Produce cross reference listing.**
- c) Write code_area into output file.**

