

Introduction

Unit- I

Components of System Programming

- Interpreter
- Assembler
- Compiler
- Macros and Microprocessors
- Formal systems
- Debugger
- Linkers
- Operating system



By

Prof. S. S. Wagh

Course Outcome

Unit- I



To **analyze** & **synthesize** various system software & understand the design of two pass assemblers.

Outline

Unit- I

Introduction



Assemblers



Software

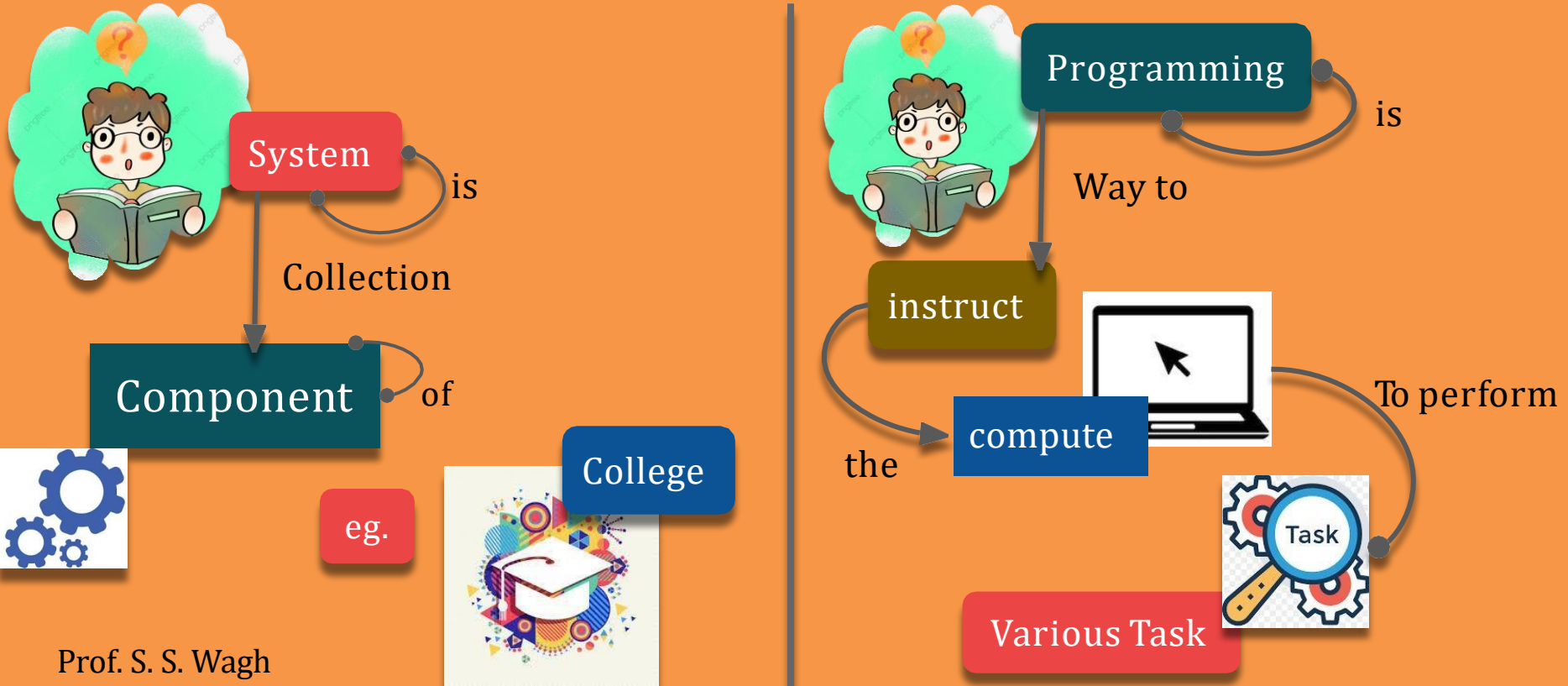


System introduction

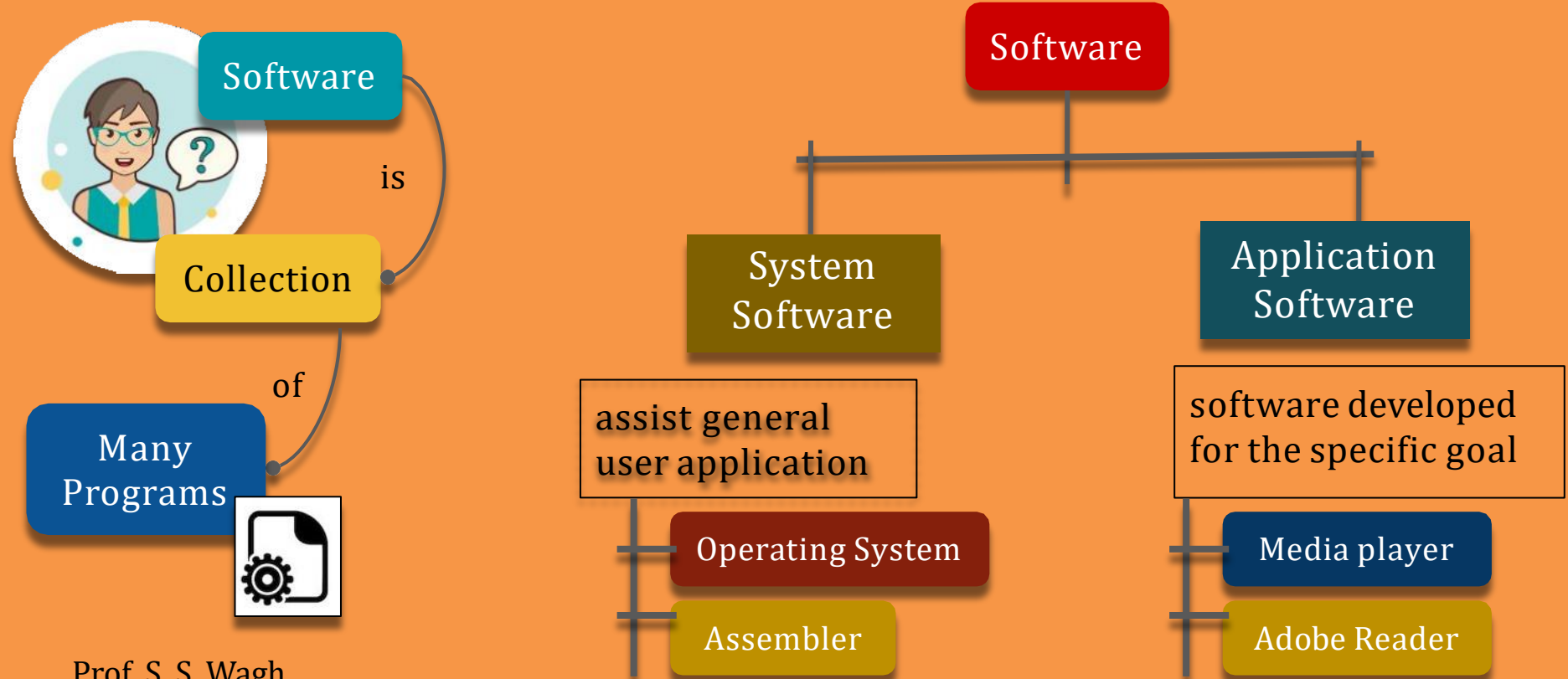
SPOS

Unit- I

system programming is an art of designing and implementing system Programs.



System introduction



System introduction

System Program

are

Required

for

Effective Execution

of

General user Programs

on

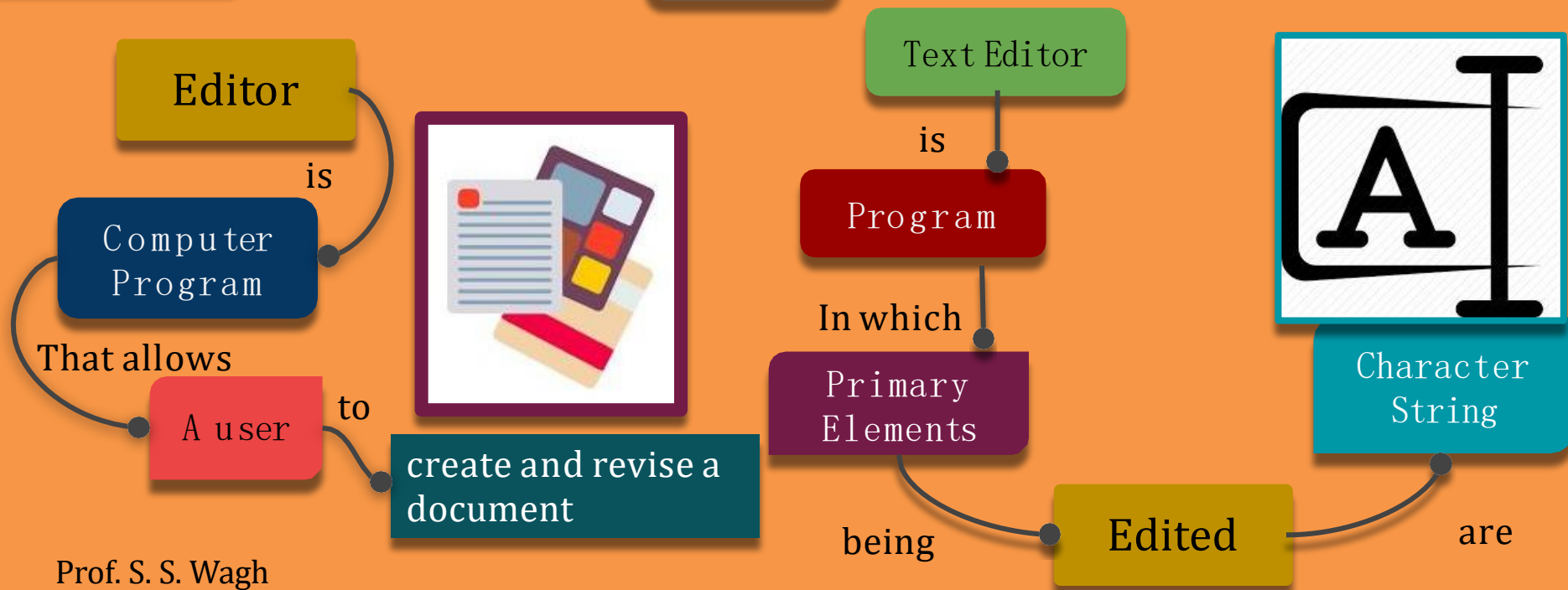
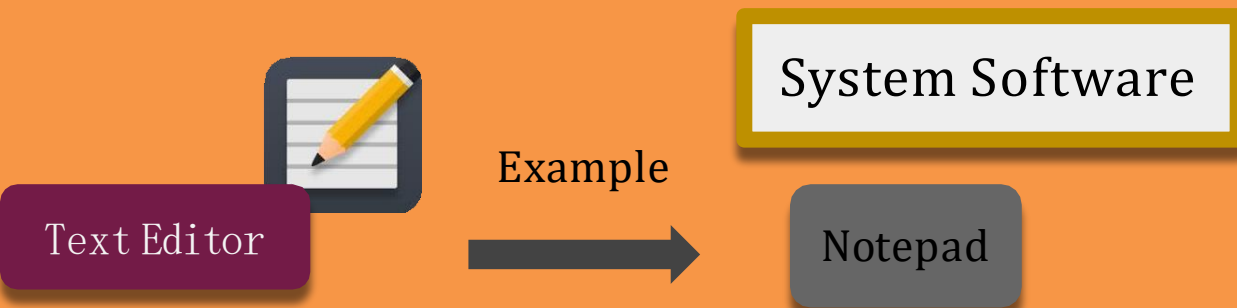


Computer System

System Programming

Is an art of

designing and implementing
system programs



System Software



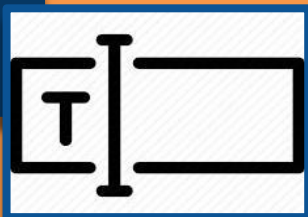
Text Editor

is

program

Used for

Editing plain
text files



With the help



of

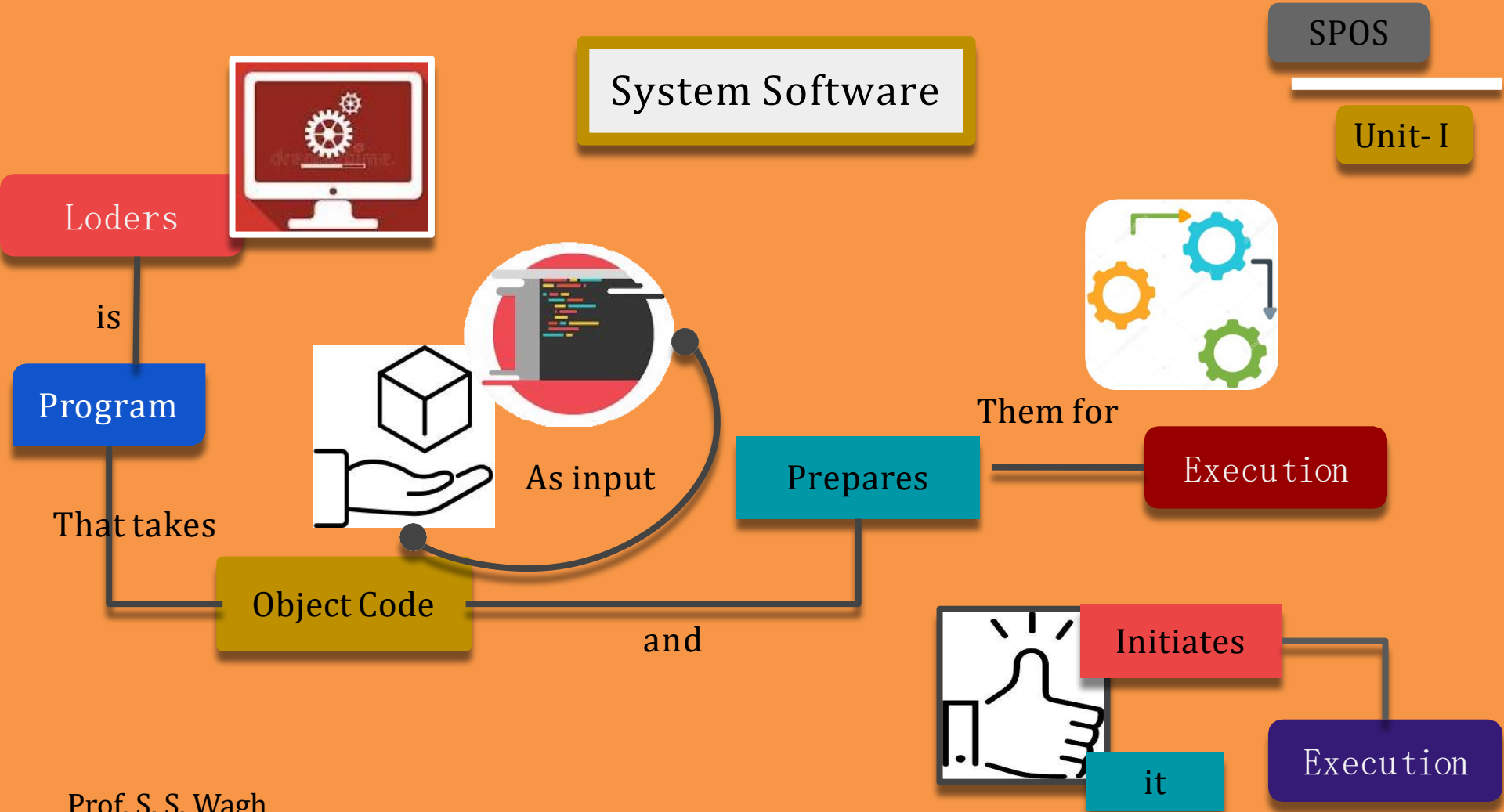
Text Editor

You can

Write Your Program

Example

C | Java
Prog.



System Software

Loders



Functions

Allocation



Linking



Relocation



Loading



System Software

Loaders



Functions

Allocation



Loader allocates space for programs in main memory.

System Software

Loaders



Functions

Relocation



- Adjusting all address dependent location.
- E.g. If we have two Programs Program A and Program B.
- Program A is saved at location 100.
- And user wants to save Program B on same location. That is physically not possible.
- So loader relocates program B to some another free location

System Software

Loaders



Functions

Linking



- If we have different modules of our program.
- Loader links object modules with each other.

System Software

Loders



Functions

Loading



Physically loading the machine instructions and data into main memory.

Outline

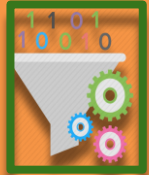
Unit- I



Review of previous session



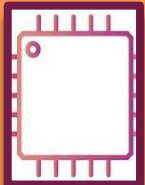
Compiler



Assembler



Debugger



Macro Processor



Assembly Language



System Software

Loders



Functions

Allocation



Linking



Relocation



Loading





Assembler

The diagram shows a computer monitor with a red border. On the screen, there are two interlocking gears, one red and one green, with curved arrows indicating they are rotating. Below the monitor is a small white base.

Assembly Lang.
Program



Assembler

The diagram shows a white square with a red border. Inside, there are two interlocking gears, one red and one green, with curved arrows indicating they are rotating.

Machine Lang.

Translate



```
graph LR; A[Assembly Lang. Program] --> B[Assembler]; B --> C[Machine Lang.]; B -- Translate --> B;
```

The flowchart illustrates the assembly process. It starts with a blue box labeled 'Assembly Lang. Program'. A blue arrow points from this box to a red box labeled 'Assembler'. Above the 'Assembler' box is a square icon containing two interlocking gears (one red, one green) with curved arrows indicating rotation. A red arrow points from the 'Assembler' box to a grey box labeled 'Machine Lang.'. A curved blue arrow loops back from the bottom of the 'Assembler' box to its input, with the word 'Translate' written below it.

Macro Processor



Macro

Allows



Sequence

of

To be

Source Lang. Code

Defined @ once



Referred many times



Macro Processor



SPOS

Unit- I

Syntax

Macro Macro name [set of parameters]

// macro body

Mend



A macro processor takes a source with macro definition and macro calls and replaces each macro call with its body



The diagram illustrates the compilation process. At the top, a box labeled 'Compiler' is accompanied by an icon of a computer monitor displaying gears. Below this, a central flow shows a red box 'High Level Lang.' with a red arrow pointing to a yellow box 'Compiler'. Above the yellow box is an icon of two interlocking gears (one red, one green). A blue curved arrow labeled 'Converts' loops from the yellow box to a dark blue box 'Low Level Lang.', which is reached via a straight blue arrow.

Compiler

High Level Lang.

Compiler

Low Level Lang.

Converts

Compiler



SPOS

Unit- I

Benefits of writing a program in a high level language

Increases productivity

It is very easy to write a program in a high level language

Machine Independence

A program written in a high level language is machine independent.

Debugger



SPOS

Unit- I

Debugging tool helps programmer for testing and debugging programs

It provides some facilities:

- Setting breakpoints.
- Displaying values of variables.

Assembly Language



SPOS

Unit- I

- Assembly language is middle level language.
- An assembly language is machine dependent.
- It differs from computer to computer.
- Writing programs in assembly language is very easy as compared to machine(binary) language

Assembly Language



SPOS

Unit- I

Assembly language programming Terms

Location Counter

(LC)



points to the next instruction

Literals



Constant Values

Assembly Language



SPOS

Unit- I

Assembly language programming Terms

Symbols



Name of variables and labels

Procedures



Methods | Function

Assembly Language

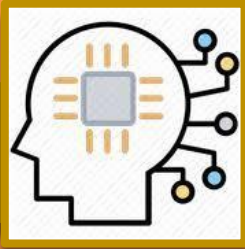


SPOS

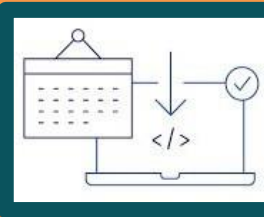
Unit- I

Assembly language Statements:

Imperative
Statements



Declarative/Declaration
Statements



Assembler Directive



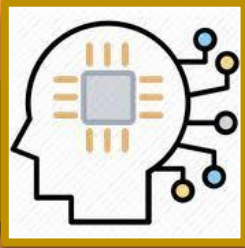
Assembly Language



SPOS

Unit- I

Imperative Statements

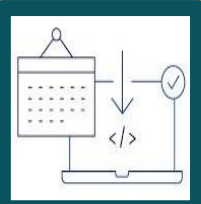


- ❑ Imperative means mnemonics
- ❑ These are executable statements.
- ❑ Each imperative statement indicates an action to be taken during execution of the program

```
E.g.  MOVER BREG, X  
      STOP  
      READ X  
      ADD AREG, Z
```

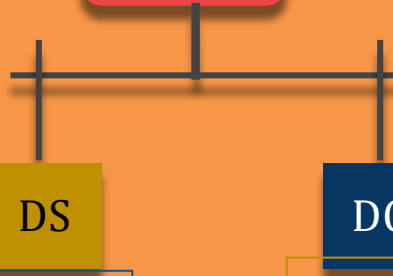


Declarative/Declaration Statements



- ❑ Declaration statements are for reserving memory for variables.
- ❑ We can specify the initial value of a variable.

Types



Declare Storage

Declare Constant

Assembly Language



SPOS

Unit- I

Declare Storage

Syntax

[Label]

DS

< Constraint Specifying size >

Example

X

DS

1

Assembly Language



SPOS

Unit- I

Declare Constant

Syntax

[Label]

DC

< Constraint Specifying values >

Example

X

DC

' 5 '

Assembly Language



SPOS

Unit- I

Assembler Directive



Some assembler directive are:

- ❑ Assembler directive instruct the assembler to perform certain actions during assembly of a program

START

< Address Constant >

END

Assembly Language



SPOS

Unit- I

Advance Assembler Directive



Origin

LTORG

EQU

DROP

USING

Outline

Unit- I



Review of previous session



Identify Statement



Machine Structure



Pass 1 Assembler

Assembly Language



SPOS

Unit- I

Sample Assembly language Code



```
1.  START 100
2.  MOVER A REG, X
3.  MOVER B REG, Y
4.  ADD A REG, Y
5.  MOVEM A REG, X
6.  X DC '10'
7.  Y DS 1
8.  END
```

Assembly Language



SPOS

Code

Unit- I

Identify types of statement

Sr. No

IS

DS

AD

1.

2.

3.

4.



1. START 100

2. MOVER B REG, Y

3. MOVER B REG, Y

4. ADD A REG, Y

Assembly Language



SPOS

Code

Unit- I

Identify types of statement

Sr. No

IS

DS

AD

5.



6.



7.



8.



5. MOVEM A REG, X

6. X DC '10'

7. YDS 1

8. End

Assembly Language



SPOS

Unit- I

Some Definitions

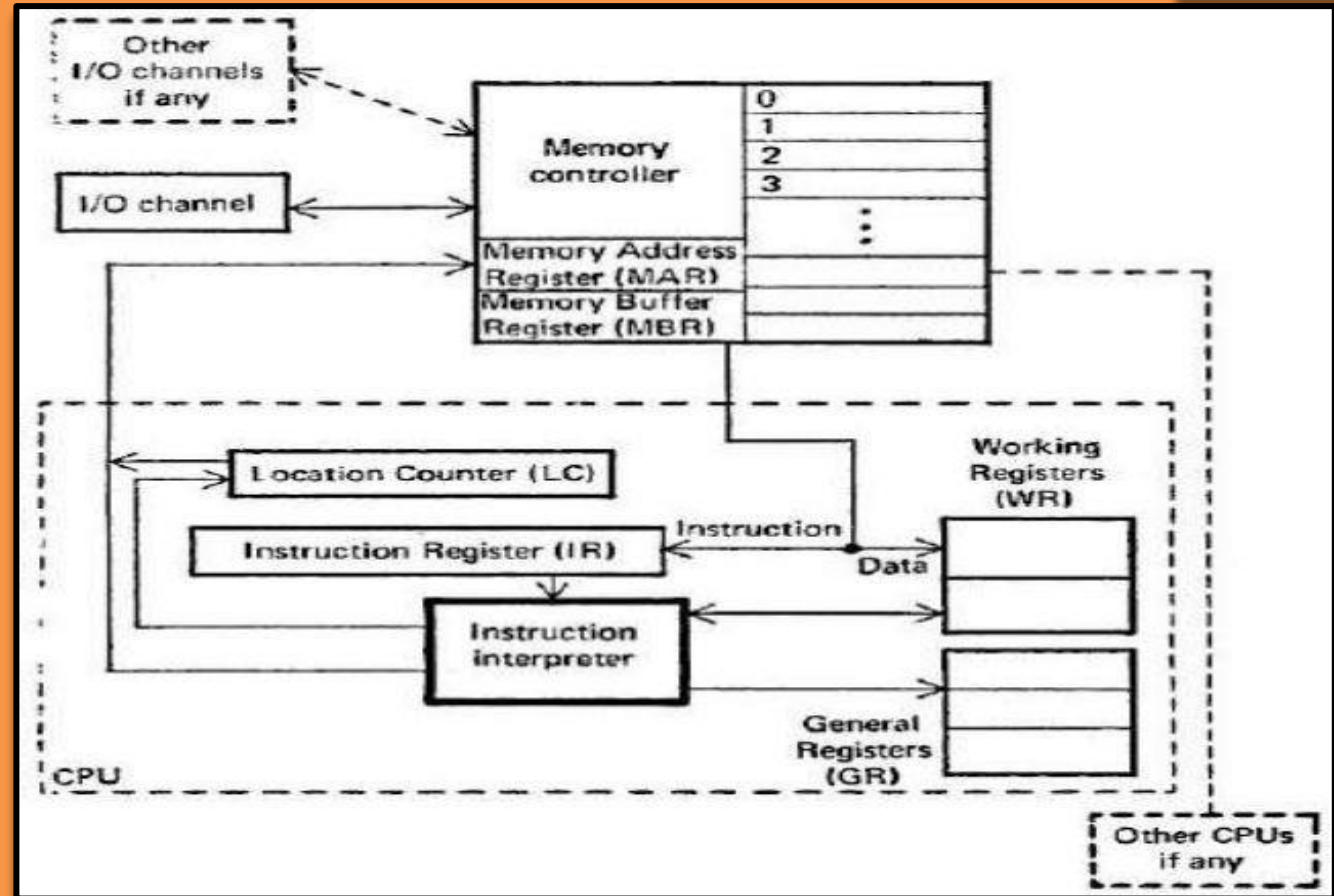


LC

Procedures

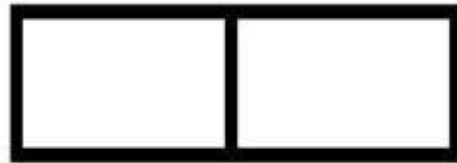
Symbol

Literals





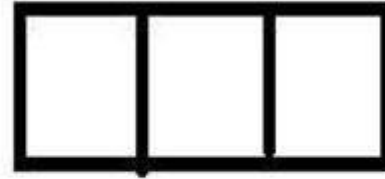
Machine Instruction Format



opcode



register operand



memory operand

Assembler



SPOS

Unit-I

Assembler

Pass 1
Assembler

Pass 2
Assembler

Labels

Mnemonic Opcode

Operand Fields

Separate

Determine Storage
Requirement

Build the Symbol
Table

Generate the
machine code

Pass 1 Assembler



SPOS

Unit-I

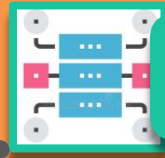
How pass 1 assembler works?



Pass 1
Assembler



Uses

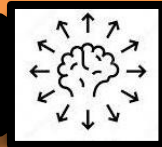


Data Structure



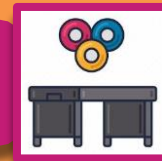
Machine opcode Table

Literal Table



Symbol Table

Pool Table





Observe code

```

L1      START 200
        MOVER AREG, ='5'
        MOVEM AREG, X
        MOVER BREG, ='2'
        ORIGIN L1+3
        LTORG

NEXT    ADD AREG, ='1'
        SUB BREG, ='2'
        BC LT, BACK
        LTORG

        BACK EQU L1
        ORIGIN NEXT+5
        MULT CREG, ='4'
        STOP
        X DS 1
        END
    
```

Apply LC

```

START 200
        MOVER AREG, ='5'      200
        MOVEM AREG, X        201
L1      MOVER BREG, ='2'      202
        ORIGIN L1+3
        LTORG
                ='5'          205
                ='2'          206

NEXT    ADD AREG, ='1'        207
        SUB BREG, ='2'        208
        BC LT, BACK          209
        LTORG
                ='1'          210
                ='2'          211

BACK    EQU    L1
        ORIGIN NEXT+5
        MULT CREG, ='4'      212
        STOP                  213
        X DS 1                214
        END
                ='4'          215
    
```



Construct

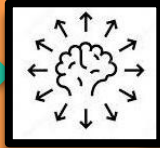


Symbol Table

index	Symbol Name	Address
0	X	214
1	L1	202
2	NEXT	207
3	BACK	202



Construct



Literal Table

index	Literal	Address
0	5	205
1	2	206
2	1	210
3	2	211
4	4	215



Pool Table



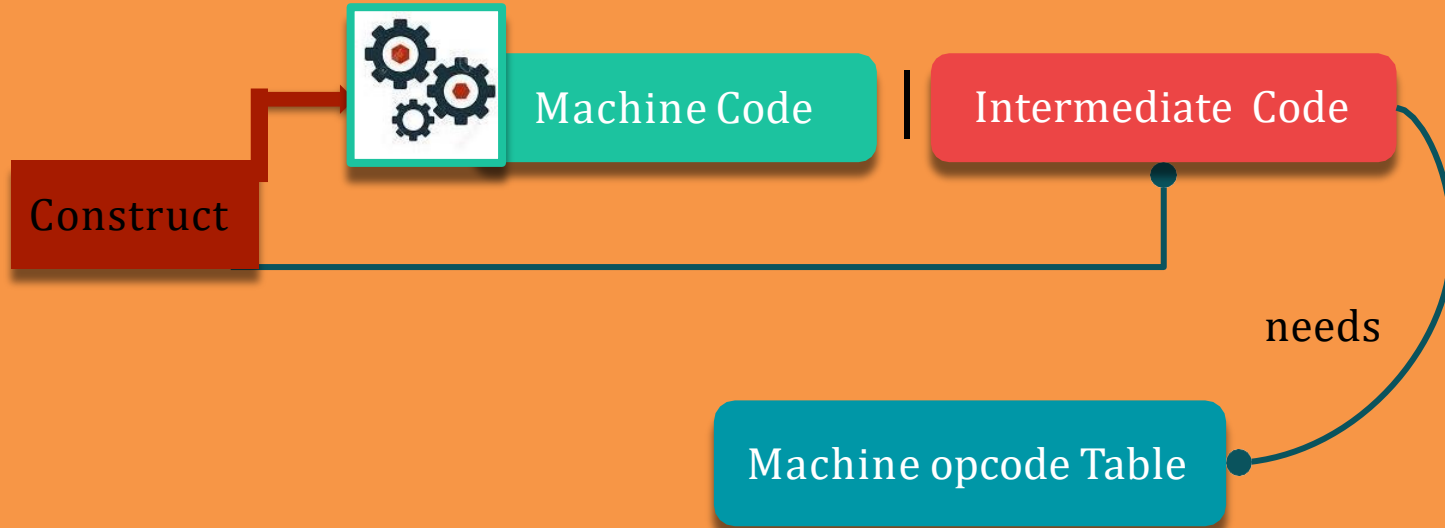
Pool table contains starting literal(index) of each pool.

Literal number

0

2

4





Enhanced Machine Opcode Table

Mnemonic opcode	Class	Opcode	Length
STOP	IS	00	1
ADD	IS	01	1
SUB	IS	02	1
MULT	IS	03	1
MOVER	IS	04	1
MOVEM	IS	05	1
COMP	IS	06	1
BC	IS	07	1
DIV	IS	08	1
READ	IS	09	1
PRINT	IS	10	1
START	AD	01	-
END	AD	02	-
ORIGIN	AD	03	-
EQU	AD	04	-
LTORG	AD	05	-
DS	DL	01	-
DC	DL	02	1
AREG	RG	01	-
BREG	RG	02	-
CREG	RG	03	-
EQ	CC	01	-



Enhanced Machine Opcode Table

Mnemonic opcode	Class	Opcode	Length
LT	CC	02	-
GT	CC	03	-
LE	CC	04	-
GE	CC	05	-
NE	CC	06	-
ANY	CC	07	-



Intermediate code

- ❑ For every line of assembly statement, one line of intermediate code is generated

- ❑ Each mnemonic field is represented as

(Statement Class , Machine code)



Intermediate code

Statement Class

Can be

IS

DS | DC

AD

MOVER

AREG, X

Mnemonic Field

Operand Field

IC for mnemonic field of above line is ,

(Statement Class , Machine code)

(IS,04)

From MOT



Intermediate code

S

Symbol

L

Literal

Operand Field

RG

Register

C

Constant

CC

Condition Codes

Operand Field

represented

(operand Class , reference)

For a symbol or literal the reference field contains the index of the operands entry in symbol table or literal table.

MOVER AREG, X

Mnemonic
code

(IS, 04) (RG, 01) (S, 0)

E.g.

START 200

(AD, 01) (C, 200)

Intermediate code

START	200	
	MOVER AREG, ='5'	200
	MOVEM AREG, X	201
L1	MOVER BREG, ='2'	202
	ORIGIN L1+3	
	LTORG	
	= '5'	205
	= '2'	206
NEXT	ADD AREG, ='1'	207
	SUB BREG, ='2'	208
	BC LT, BACK	209
	LTORG	
	= '1'	210
	= '2'	211
BACK	EQU L1	
	ORIGIN NEXT+5	
	MULT CREG, ='4'	212
	STOP	213
	X DS 1	214
	END	
	= '4'	215

(AD, 01) (C, 200)
200 (IS, 04) (RG,01) (L, 0)
201 (IS, 05) (RG,01) (S,0)
202 (IS, 04) (RG,02) (L,1)
203 (AD, 03) (C, 205)
205 (DL, 02) (C,5)
206 (DL, 02) (C, 2)
207 (IS,01) (RG, 01) (L, 2)
208 (IS, 02) (RG, 02) (L,3)
209 (IS, 07) (CC, 02) (S, 3)
210 (DL,02) (C,1)
211 (DL,02) (C,2)
212 (AD, 04) (C, 202)
212 (AD, 03) (C, 212)
212 (IS, 03) (RG, 03)(L, 4)
213 (IS, 00)
214 (DL, 01, C, 1)
215 (AD, 02)

Example 2

Assignment

SPOS

Unit- I

```
START 205
MOVER AREG, ='6'
MOVEM AREG, A
LOOP  MOVER AREG, A
      MOVER CREG, B
      ADD CREG, ='2'
      BC ANY , NEXT
      LTORG
      ADD BREG, B
NEXT  SUB AREG, ='1'
      BC LT, BACK
LAST  STOP
      ORIGIN LOOP+2
      MULT CREG, B
      ORIGIN LAST+1
A     DS      1
BACK  EQU     LOOP
B     DS      1
END
```

Outline

Unit- I



Review of previous session



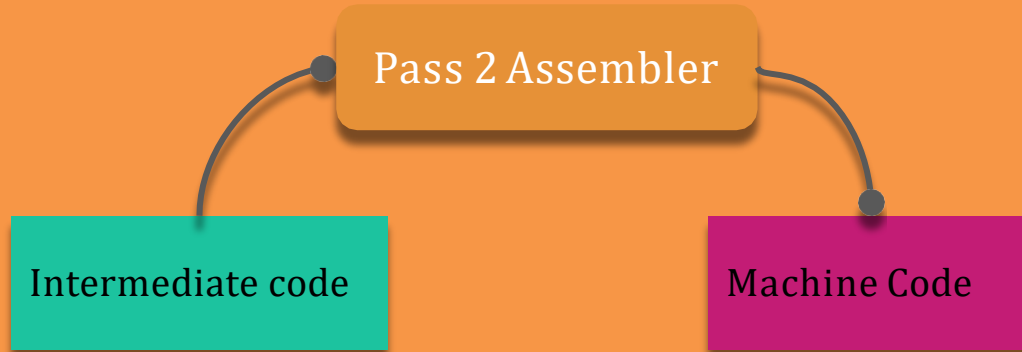
Pass 2 Assembler



Pass 2 Assembler

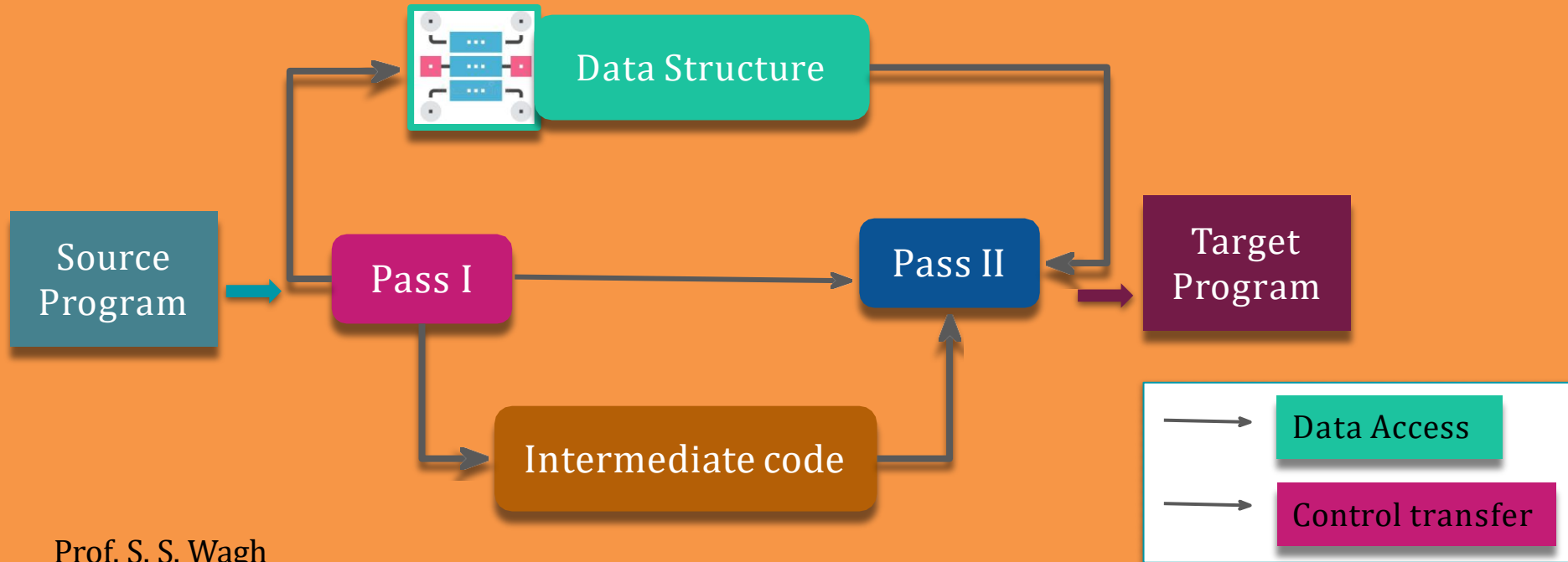


Processes the intermediate representation (IR) to synthesize the target program.





Pass 2 Assembler

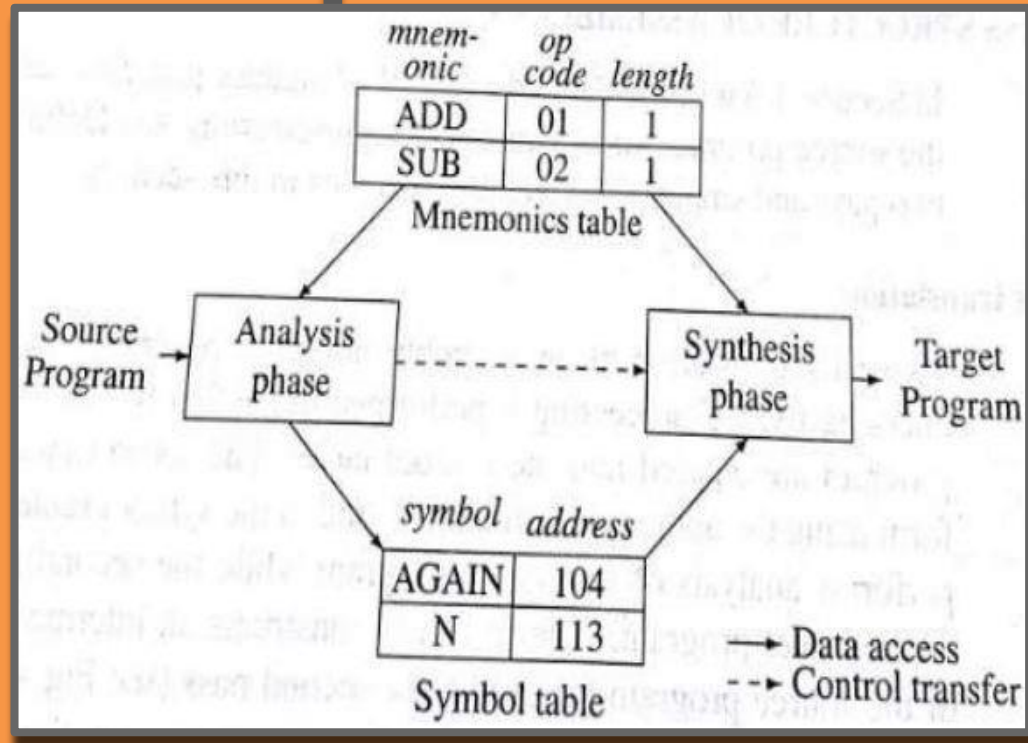




Analysis Phase

Vs

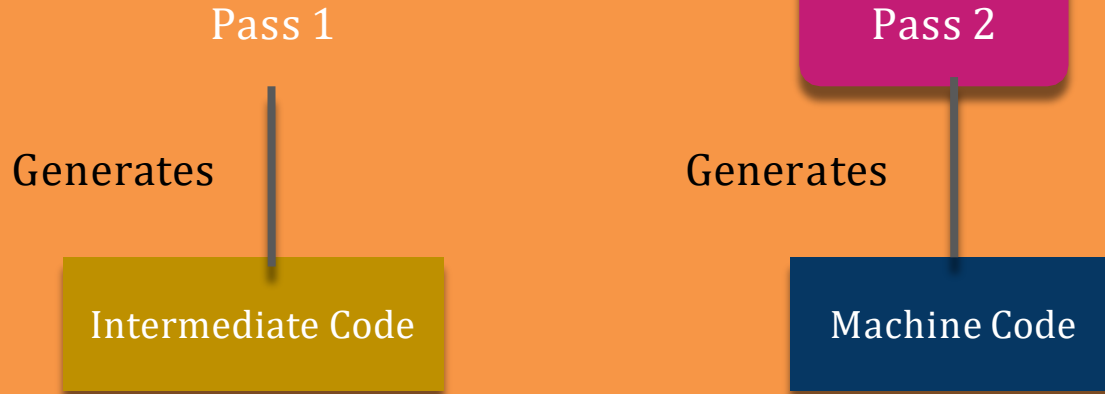
Synthesis Phase





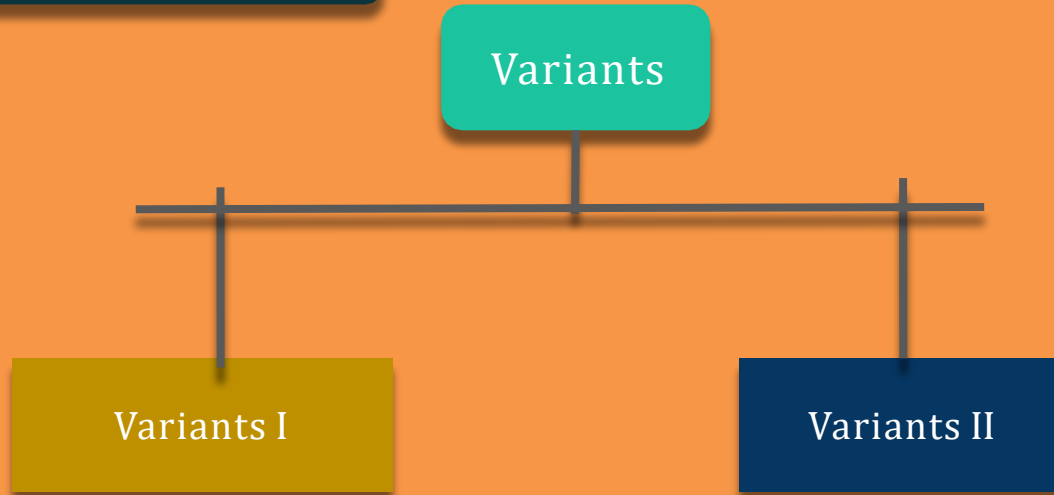
Sr. No	Pass 1	Pass 2
01	It requires only one scan to generate machine code	It requires two scan to generate machine code.
02	It has forward reference problem.	It don't have forward reference problem.
03	It performs analysis of source program and synthesis of the intermediate code.	It process the IC to synthesize the target program.
04	It is faster than pass 2.	It is slow as compared to pass 1.

Output of Pass 1 | Pass 2 Assembler





Variants of Intermediate Code





Variants of Intermediate Code

Variants I

In Variant I, each operand is represented by a pair of the form (operand class, code).

The operand class is one of:

1. S for symbol
2. L for literal
3. C for constant
4. RG for register.



Variants of Intermediate Code

Variants I

	START	100	(AD, 01) (C, 100)
L1	READ	A	(IS, 09) (S, 1)
	SUB	AREG, = '5'	(IS, 02) (RG, 01) (L, 0)
	BC	GT, L1	(IS, 07) (CC, 03) (S, 0)
	STOP		(IS, 00)
A	DS	1	(DL, 01) (C, 1)
	-		-
	-		-
	-		-



Variants of Intermediate Code

Variants II

In variant II, operands are processed selectively.

Constants and literals are processed. Symbols, condition codes and CPU registers are not processed.



Variants of Intermediate Code

Variants II

A sample intermediate code

Fig. 1.10.6.

	START	100	(AD, 01) (C, 100)
L1	READ	A	(IS, 09) A
	SUB	AREG, = '5'	(SI, 02) AREG, (L, 0)
	BC	GT, L1	(IS, 07) GT, L1
	STOP		(SI, 00)
A	DS	1	(DL, 01) (C, 1)
	—		—
	—		—
	—		—

Fig. 1.10.6 : Intermediate code using variant II