System Programming and Compiler Construction

CSC 602

Subject Incharge

Varsha Shrivastava

Assistant Professor

email: varshashrivastava@sfit.ac.in

Room No: 407

CSC 602 System Programming and Compiler Construction

Module 2

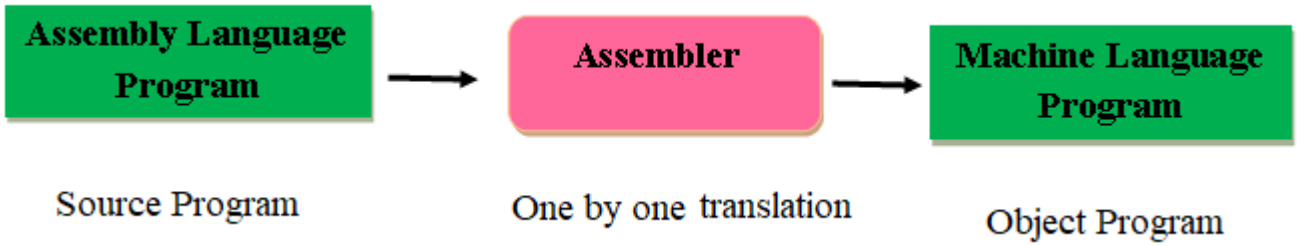
Assembler

Content as per syllabus

- Introduction
- Elements of Assembly Language programming, Assembly scheme
- Pass structure of assembler
- Assembler Design:
  - Two pass assembler Design and
  - Single pass Assembler Design for Hypothetical / X86 family processor
  - Data structures used

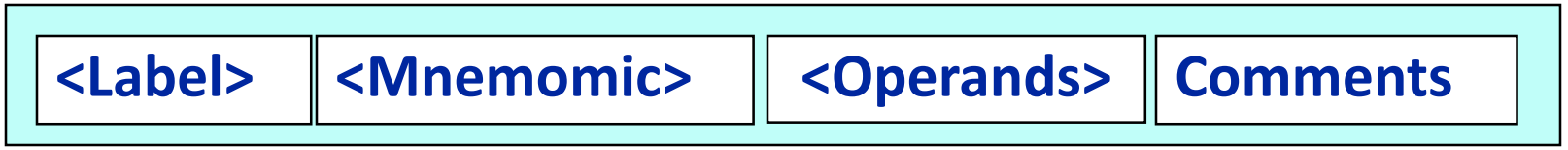
Introduction

- Assembler is system software which converts an assembly language
  - The input to the assembler is a source code written in assembly language (using mnemonics) and the output is an object code.
- Basic Assembler functions:
  - Translating mnemonic language to its equivalent object code.
  - Assigning machine addresses to symbolic labels.



Elements of Assembly Language programming

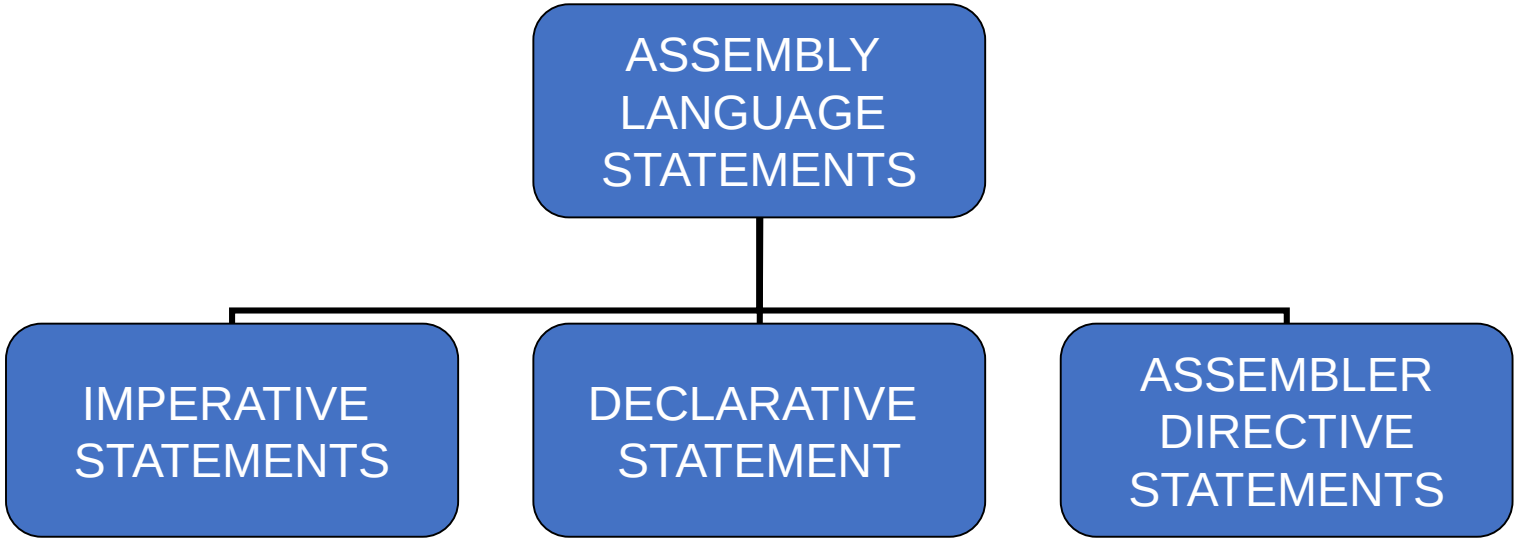
Assembler languages-structure



- Label
  - symbolic labeling of an assembler address (command address at Machine level)
- Mnemomic
  - Symbolic description of an operation
- Operands
  - Contains of variables or address if necessary
- Comments
  - ignored by assembler
  - used by humans to document/understand programs

Elements of Assembly Language programming

Types of Assembly Language Statements



Types of Assembly Language Statements

Imperative statements

An imperative statement in assembly language indicates the action to be performed during execution of assembly statement

Ex:- A 1,FOUR

Types of Assembly Language Statements

Assembler Directives

- These statements instructs the assembler to perform certain action during the assembly of a program.
- Ex START 100

  USING *, 15

  USING indicates to the assembler which general register to use as a base and what its content will be.

Types of Assembly Language Statements

Declarative Statement

- These statements declares the storage area or declares the constant in program.
- EX: A DS 1
  ONE DC "1"

[Label] DS <constant>

The DS statement reserves areas of memory and associates names with them.

[Label] DC '<value>'

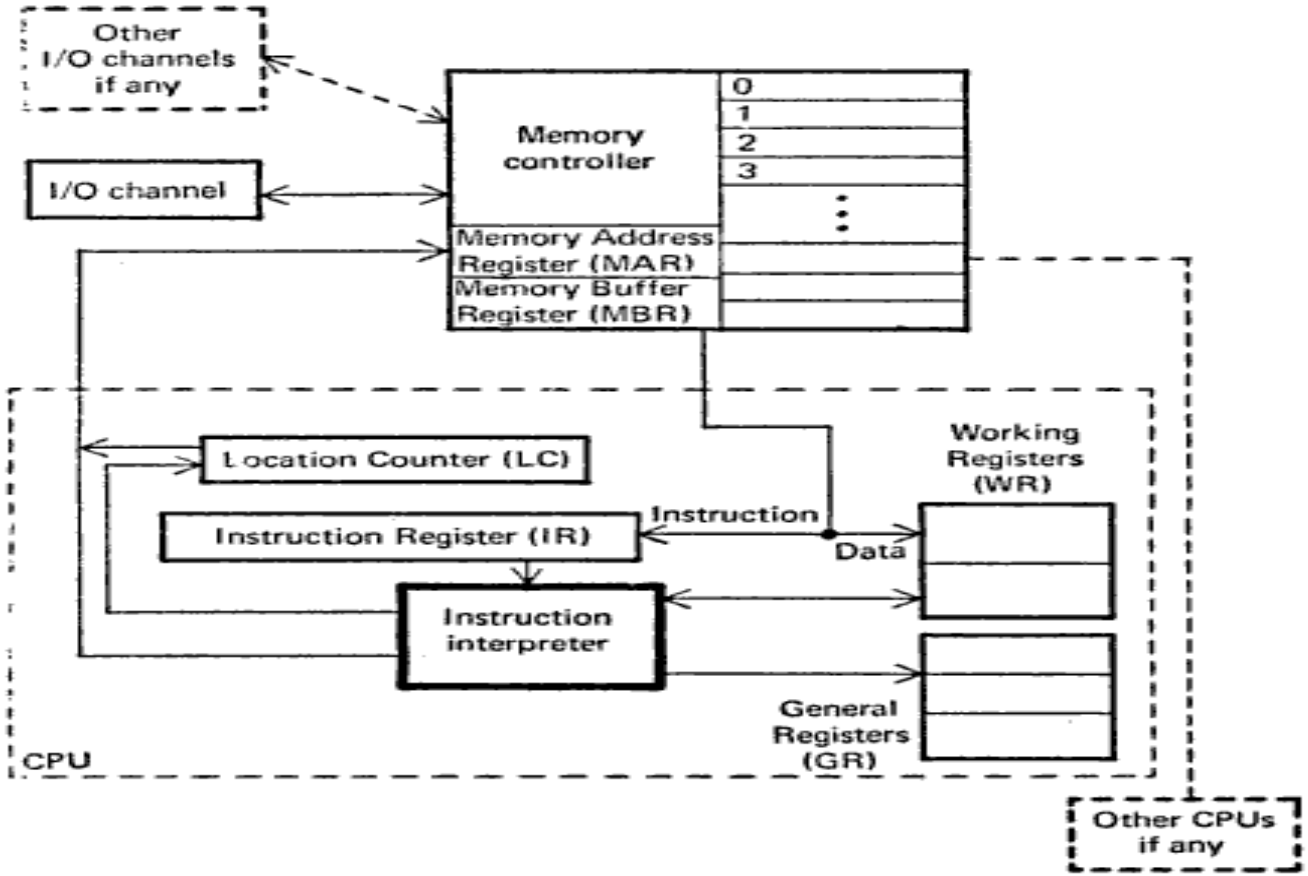
The DC statement constructs memory words containing constants.

Terms to remember

- A pseudo-op is an assembly language instruction that specifies an operation of the assembler i.e
  about the base register & its contents e.g. USING, DC, DS instruction.
- On the other hand, a machine-op instruction. That represents a machine instruction to the assembler e.g. BR instruction is a machine-op instruction

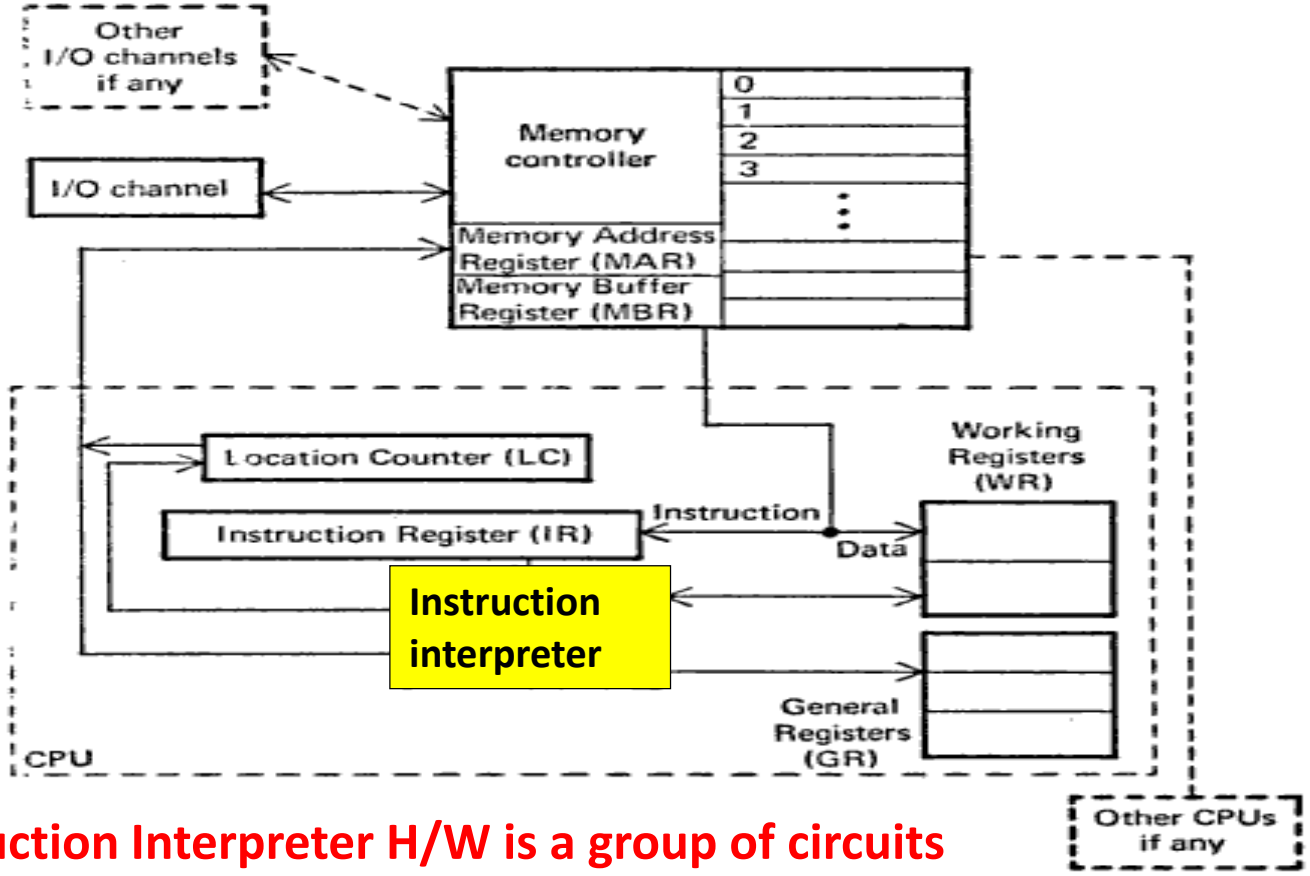
General Machine Architecture

IBM 360 Architecture



General Machine Architecture

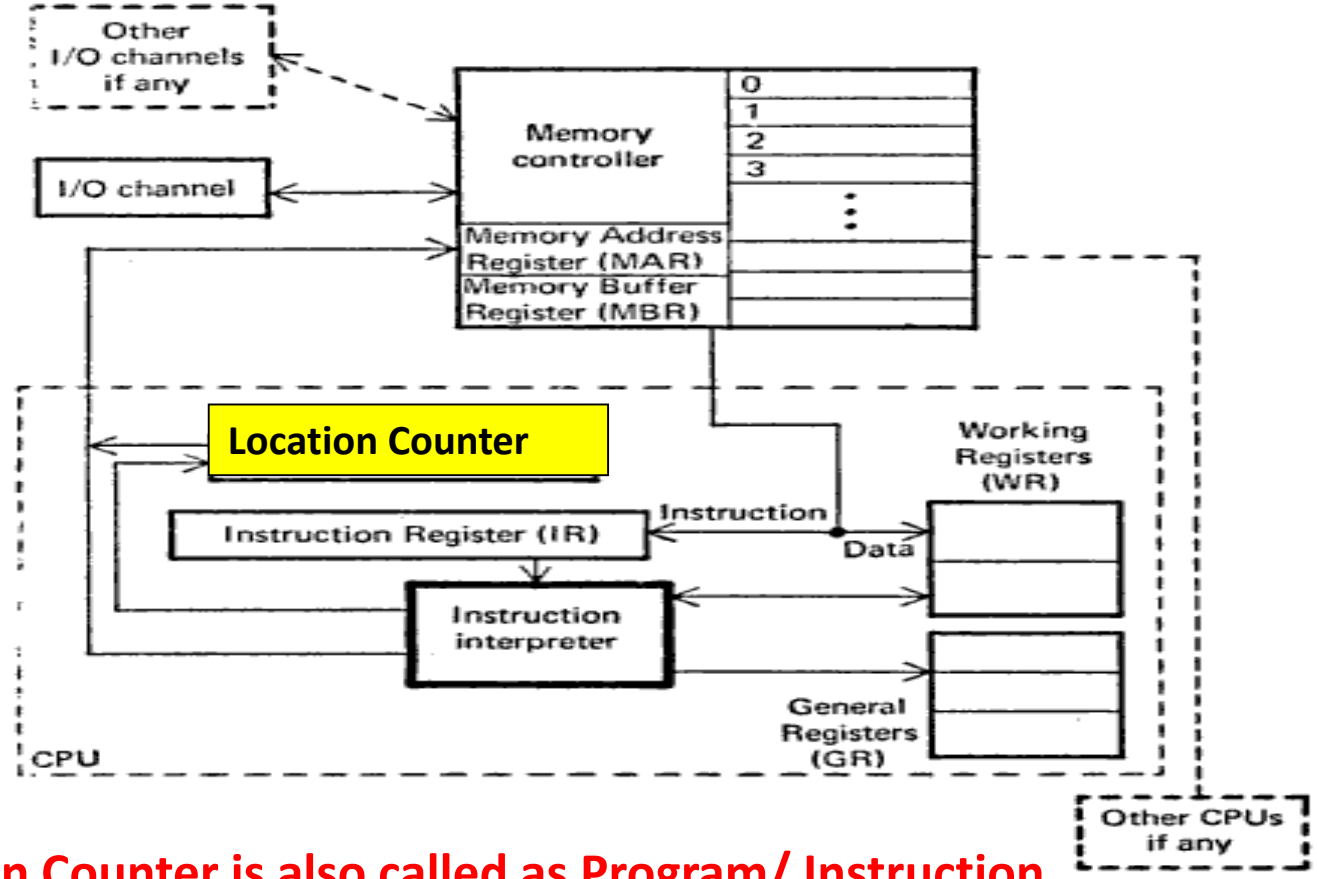
IBM 360 Architecture



Instruction Interpreter H/W is a group of circuits that perform the operation specified by the instruction fetched from the memory

General Machine Architecture

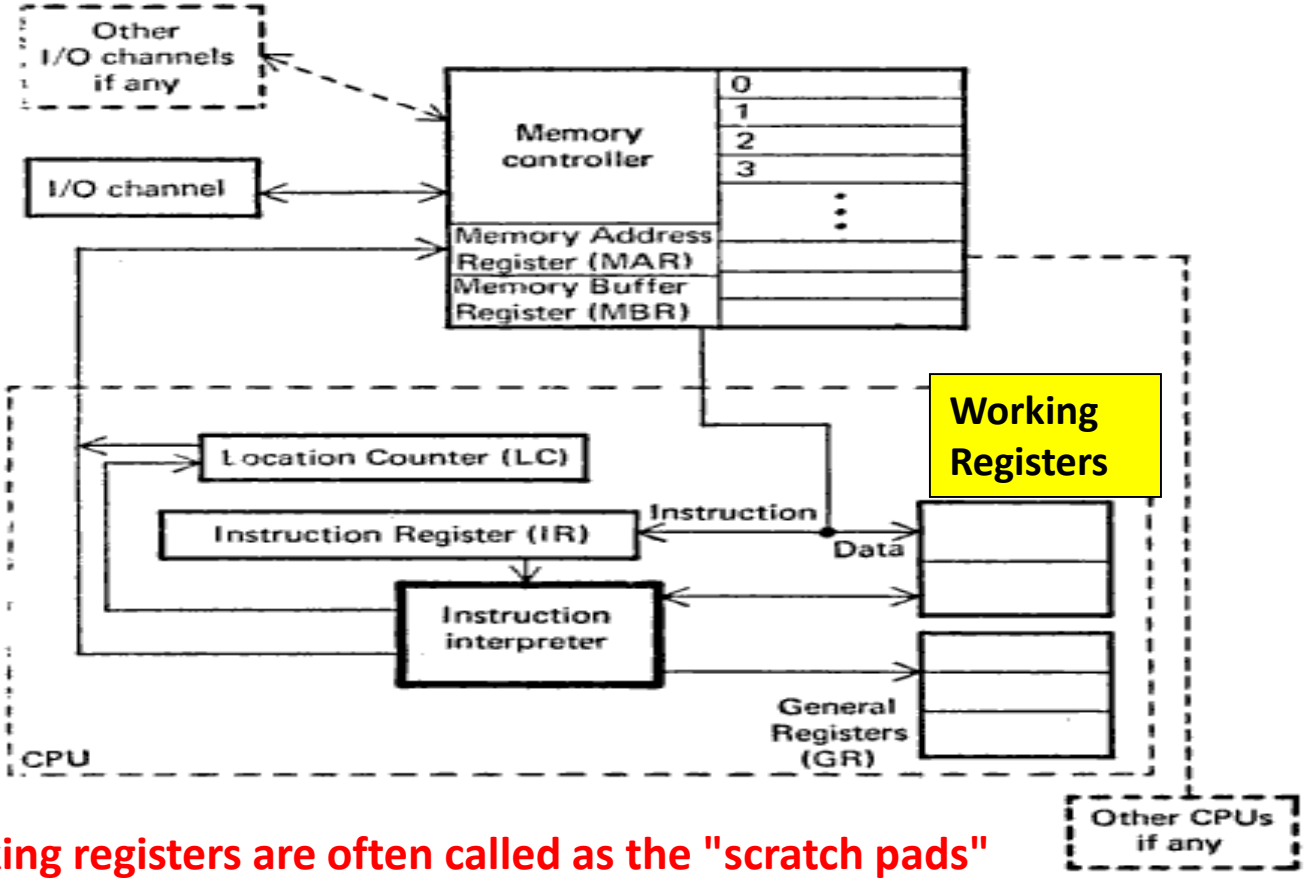
IBM 360 Architecture



Location Counter is also called as Program/ Instruction Counter, it points to the current instruction being executed.

General Machine Architecture

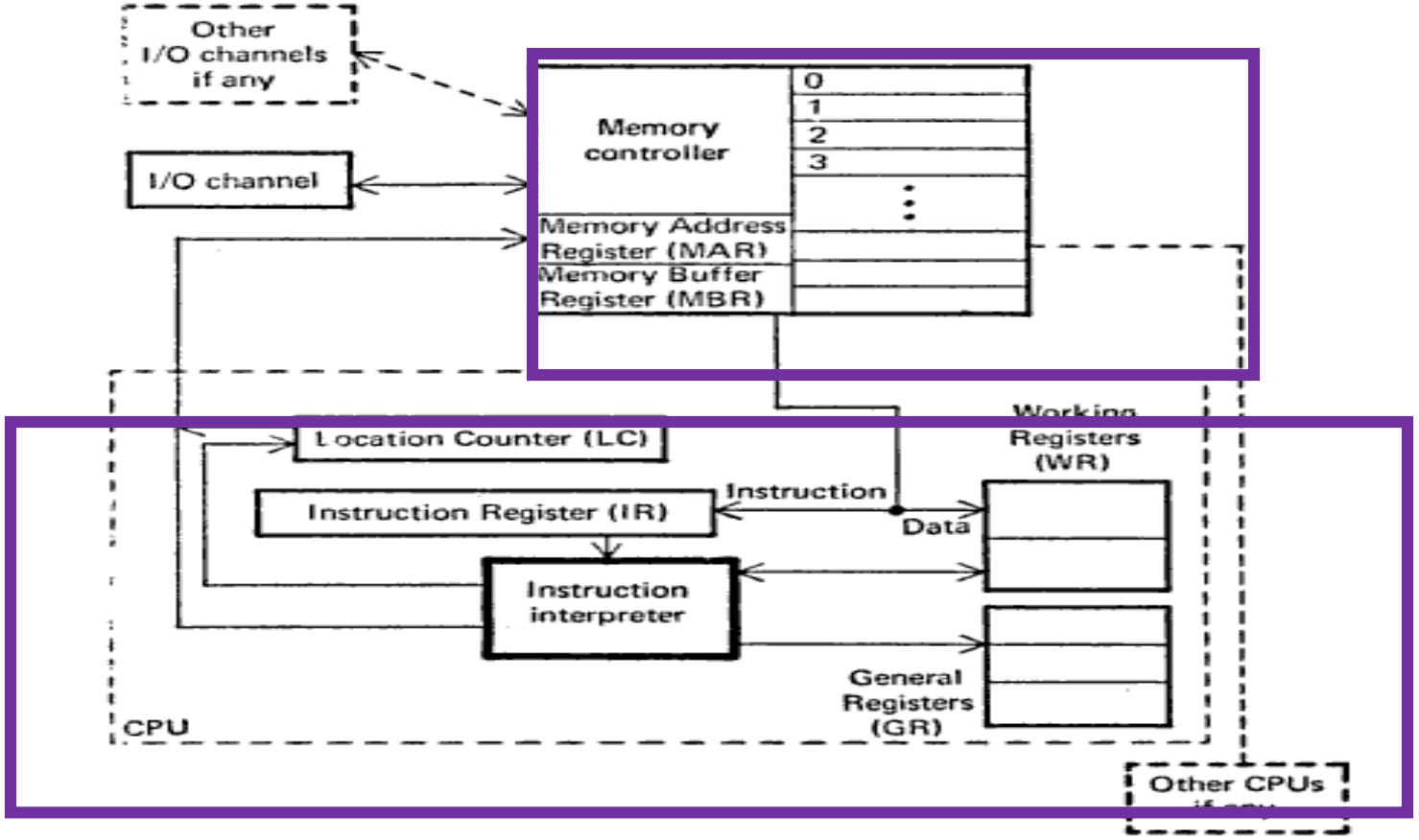
IBM 360 Architecture



The working registers are often called as the "scratch pads" because they are used to store temporary values while calculation is in progress.

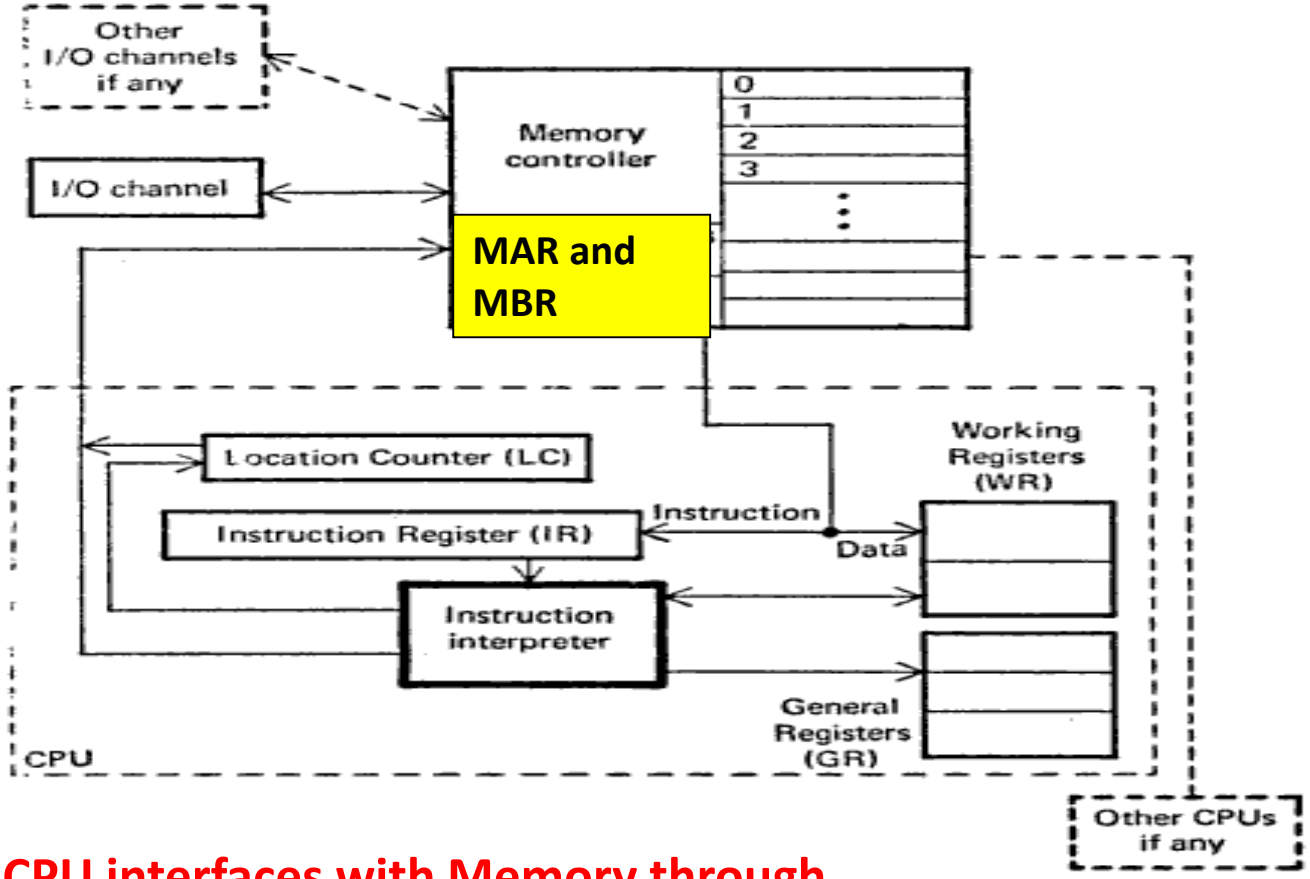
General Machine Architecture

IBM 360 Architecture



General Machine Architecture

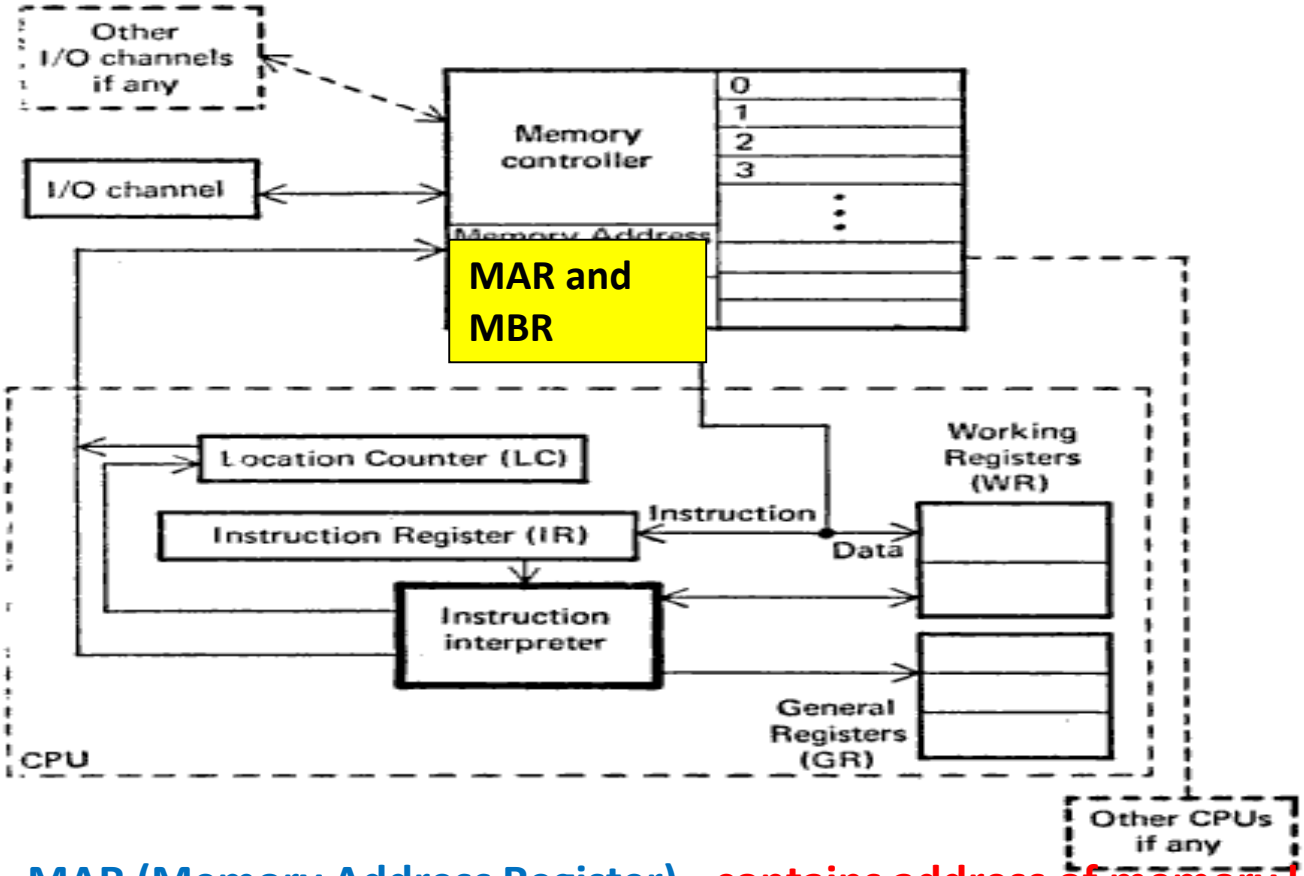
IBM 360 Architecture



This CPU interfaces with Memory through MAR & MBR

General Machine Architecture

IBM 360 Architecture

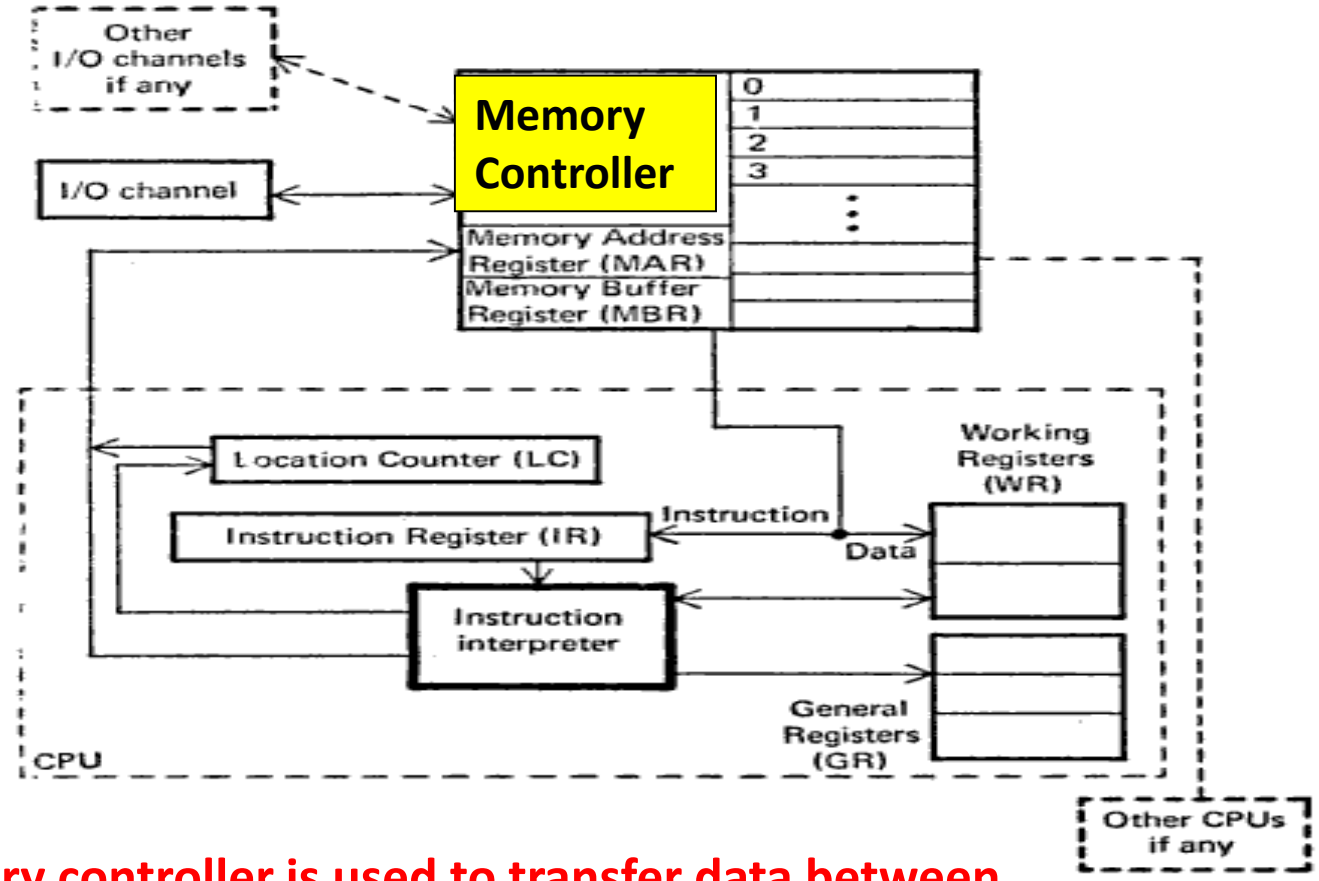


MAR (Memory Address Register) - contains address of memory location (to be read from or stored into)

MBR (Memory Buffer Register) - contains copy of address specified by MAR

General Machine Architecture

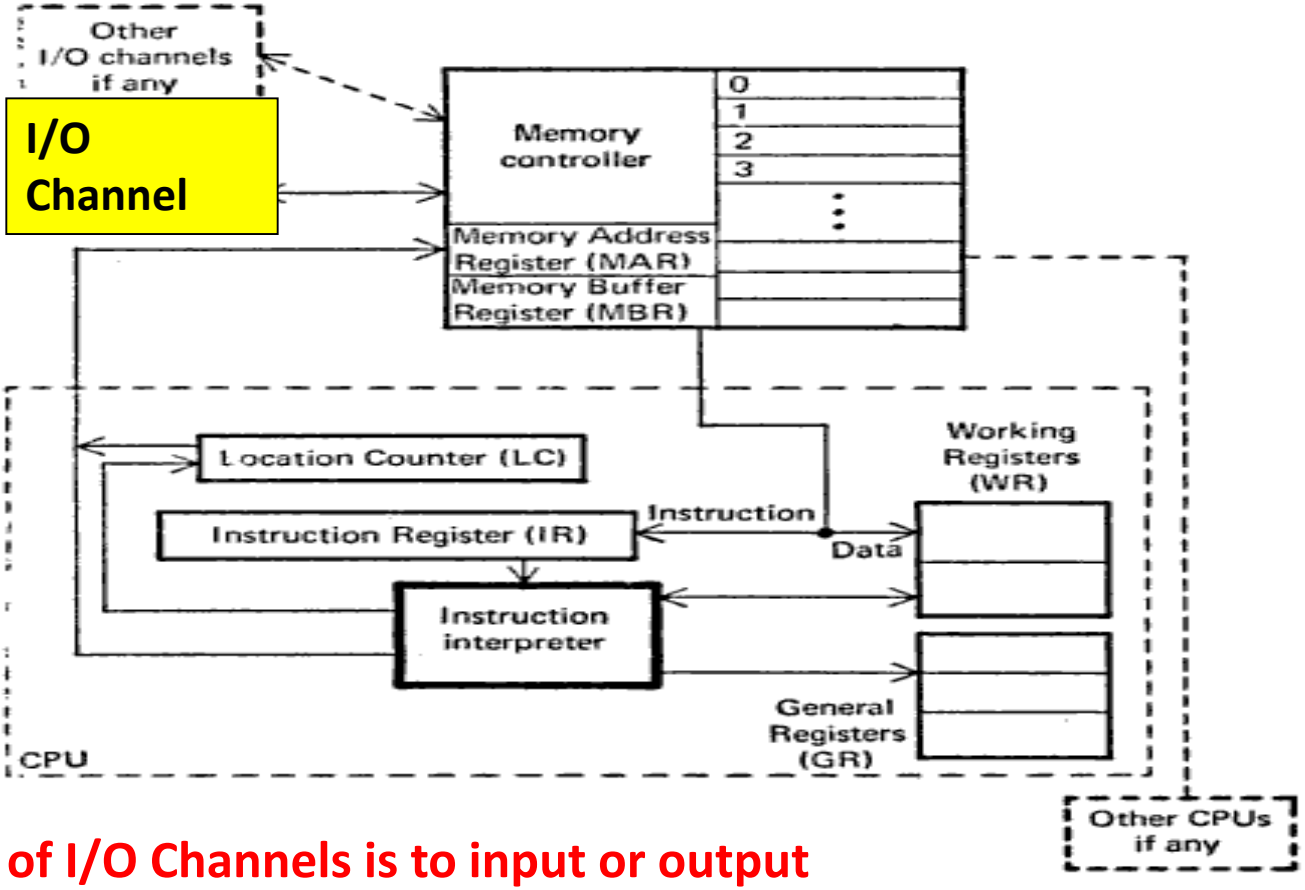
IBM 360 Architecture



Memory controller is used to transfer data between MBR & the memory location specified by MAR

General Machine Architecture

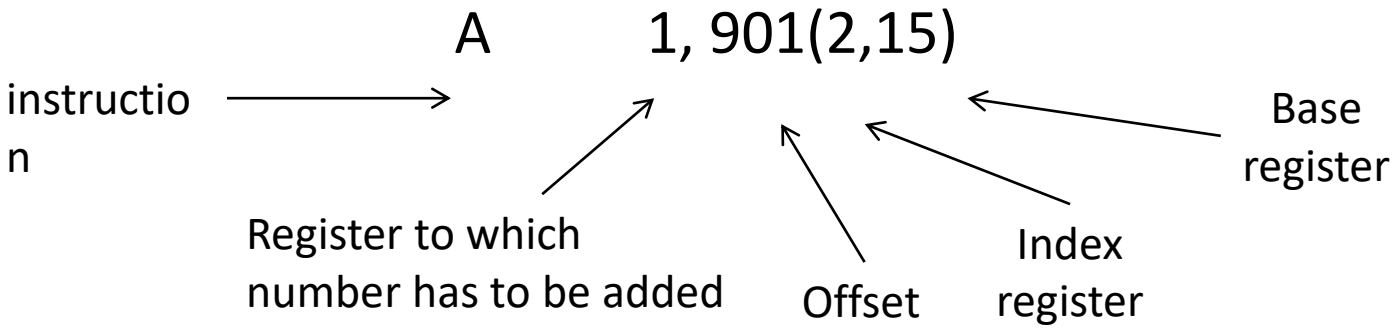
IBM 360 Architecture



The role of I/O Channels is to input or output information from memory.

IBM 360/370 Structure

- Memory
  - 2^{24} bytes (16 million bytes)
  - 3 components
    - Offset
    - Base Register
    - Index Register
  - Example: add instruction

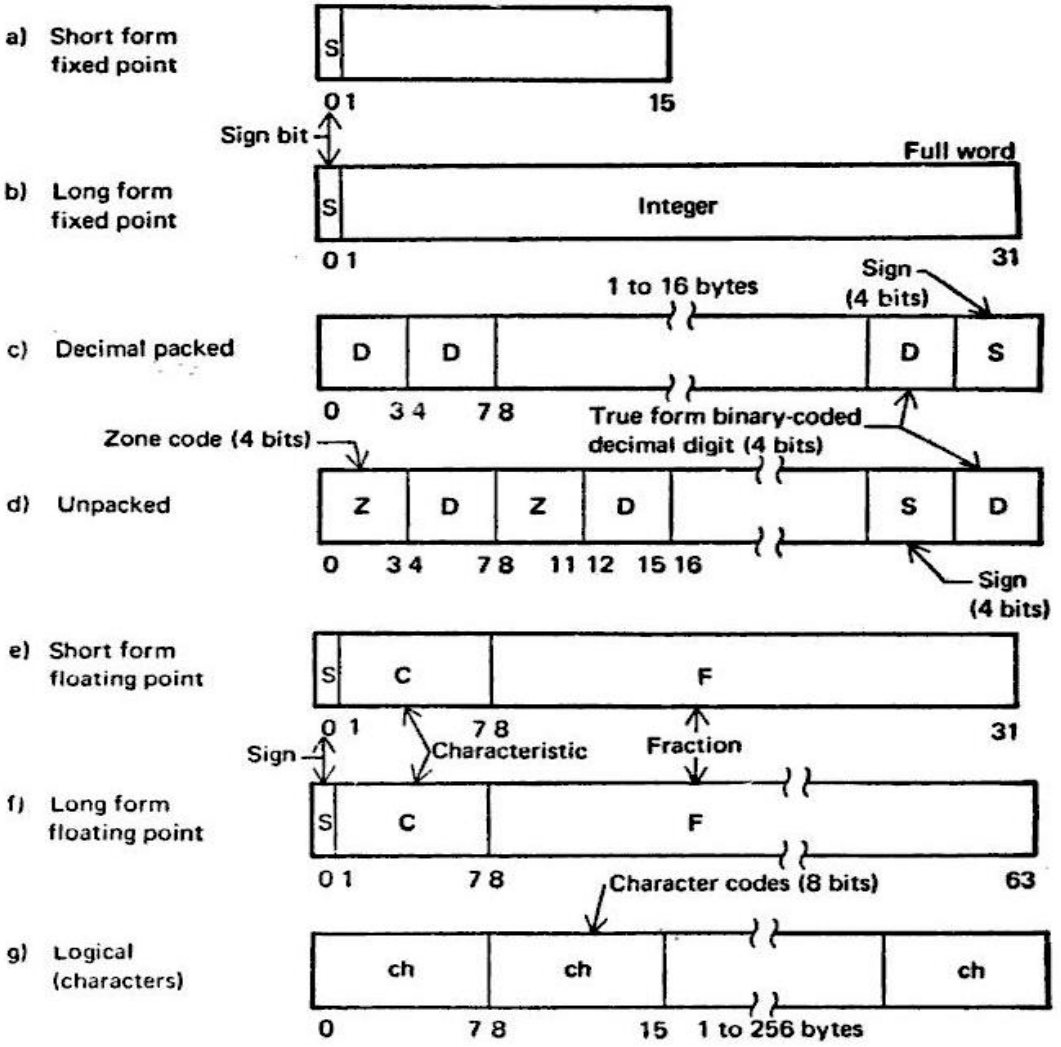


IBM 360/370 Structure

- Registers
  - 16 general purpose registers of 32 bits each
    - Can be used for various arithmetic and logical operations as base registers
    - Can be used as temporary storage for operations
    - In base register form, they aid in formation of address
  - 4 floating point registers of 64 bits each
  - 64 bit Program Status Word (Flags)

IBM 360/370 Structure

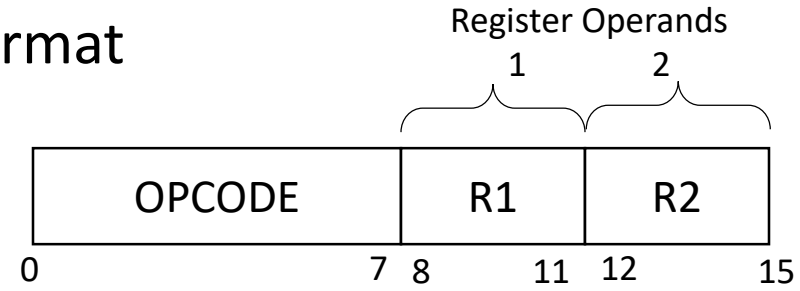
- **Data**
  - B’ 0000 0001 0010 1100
  - X’ 0 1 2 C
  - Prefix
    - X – Hexadecimal
    - B – Binary



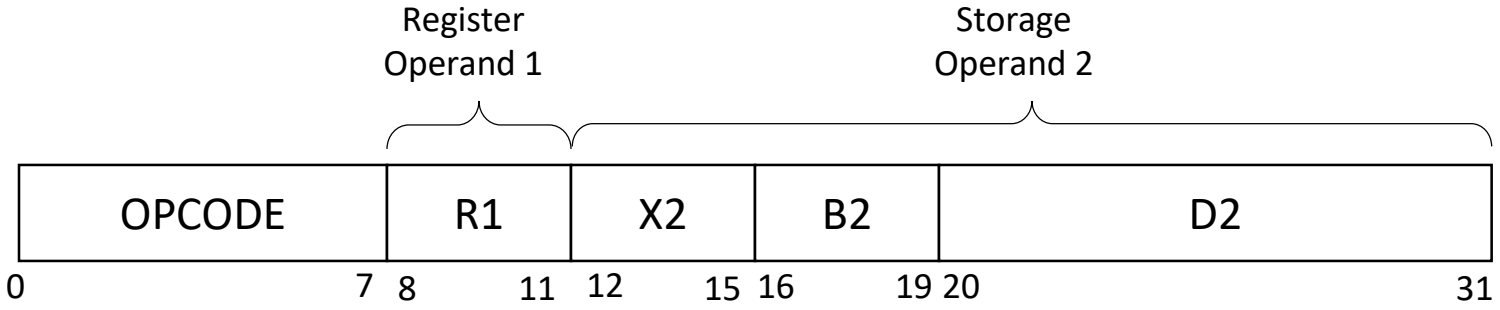
IBM 360/370 Structure

Instructions

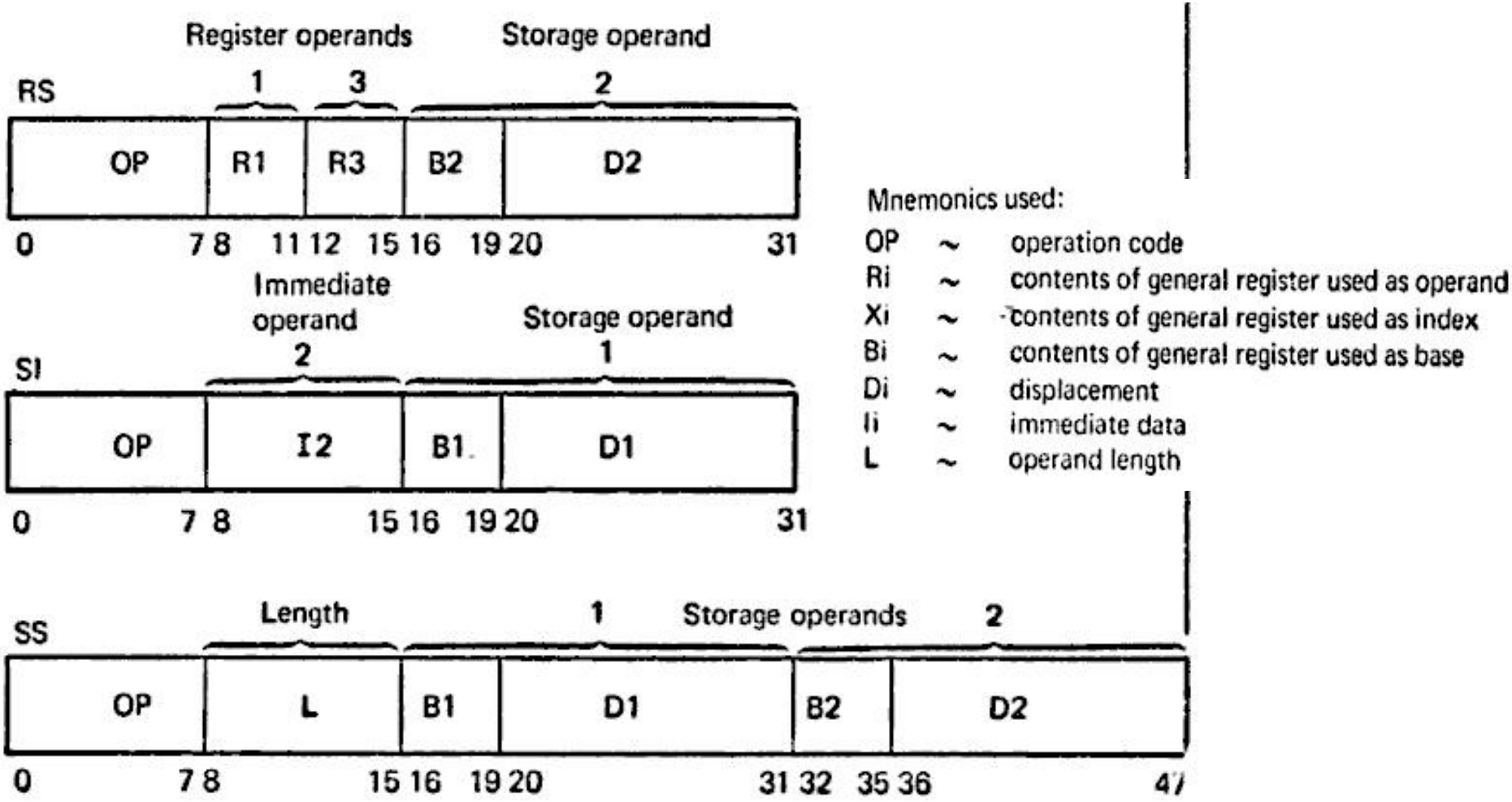
- RR format



- RX format



IBM 360/370 Structure



IBM 360/370 Structure

- Special Features
  - Has hardware protection in blocks of 2048 bytes
  - Has elaborate interrupt structure

Pseudo Operation Code (Pseudo-op)

"Pseudo Op can also be termed as Assembler Directives"

START : START is a pseudo-op that tell the assembler where the beginning of the program is and allows the user to give a name to the program.

Eg: TEST START

END : END is a pseudo-op tell the assembler that the end of the program has been reached

Eg: END

Pseudo Operation Code (Pseudo-op)

USING : USING is a pseudo-op that indicates to the assembler which general purpose register to use as a base register and what its content will be.

Eg: USING *,15

USING BEGIN+2,15

DROP : DROP is a pseudo-op tell which indicates unavailable base register.

Eg: DROP 15

Pseudo Operation Code (Pseudo-op)

DS : DS is a pseudo-op which make the assembler reserve a space in memory. DS is only for define storage.

Eg: BLOCK DS 5F

DC : DC is used to define storage and store a constant.

Eg: FOUR DC 1F '4'

Pseudo Operation Code (Pseudo-op)

LTORG : LTORG is a pseudo-op which tells the assembler to place the encountered literals at an earlier locations.

Eg: BLOCK DS 5F

EQU : EQU is pseudo-op allow the programmer to define variables.

Eg: BASE EQU 15

Machine Operation Code (Machine-op)

BALR : Branch and link- load the register with next address and branch to the address in the second field. (RR format instruction)

Eg: BALR 15,0

BR : Unconditionally Branch to the location whose address in register.

Eg: BR 14

Machine Operation Code (Machine-op)

BCT : Branch on Count- RX Instruction

This command decrements the given register by one and if the result is positive, branches to the address in the second field

Eg: BCT 3, LOOP

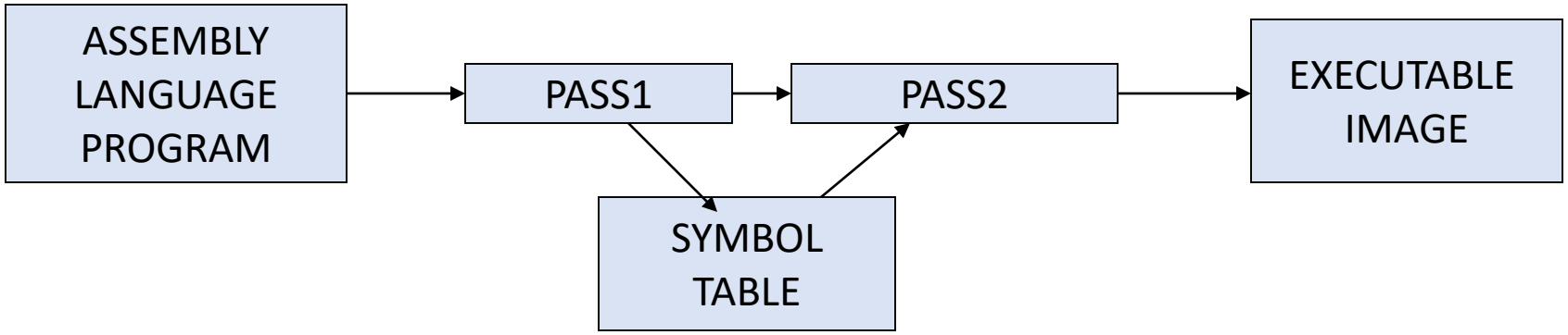
Assembly language program

	Program		*Comments*
TEST	START		Identifies name of program
BEGIN	BALR	15,0	Set register 15 to the address of the next instruction
	USING	BEGIN+2,15	Pseudo-op indicating to assembler register 15 is base register and its content is address of next instruction
	SR	4,4	Clear register 4 (set index=0)
	L	3,TEN	Load the number 10 into register 3
LOOP	L	2,DATA(4)	Load data (index) into register 2
	A	2,FORTY9	Add 49
	ST	2,DATA(4)	Store updated value of data (index)
	A	4,FOUR	Add 4 to register 4 (set index = index+4)
	BCT	3,LOOP	Decrement register 3 by 1, if result non-zero, branch back to loop
	BR	14	Branch back to caller
TEN	DC	F'10'	Constant 10
FOUR	DC	F'4'	Constant 4
FORTY9	DC	F'49'	Constant 49
DATA	DC	F'1,3,3,3,3, 4,5,8,9,0'	Words to be processed
	END		

Types of Assembler

- Single pass Assembler
- Multi pass Assembler

Assembler Process



First Pass:

scan program file

find all labels and calculate the corresponding addresses;

this is called the *symbol table*

Second Pass:

convert instructions to machine language,

using information from symbol table

Problem of Forward Reference

- Variables whose declaration is later in the code is encountered, the **forward reference problem** occurs.

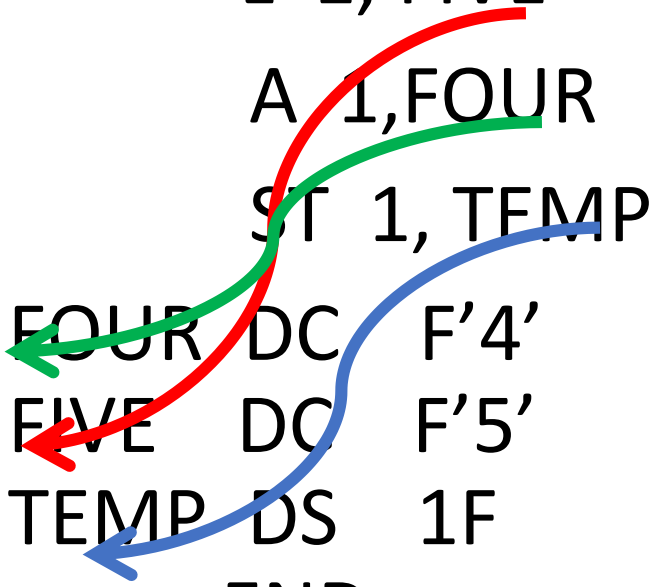
- To avoid the forward reference problem:
  i) Declare the symbols before using it
  ii) Use multi pass assemblers

Example : Assembly Program

```
JOHN   START    0
       USING *, 15
       L  1, FIVE
       A  1,FOUR
       ST  1, TEMP
FOUR   DC   F'4'
FIVE   DC   F'5'
TEMP   DS   1F
       END
```

Problem of Forward Reference

```
JOHN   START    0
       USING *, 15
       L  1, FIVE
       A  1,FOUR
       ST  1, TEMP
FOUR   DC   F'4'
FIVE   DC   F'5'
TEMP   DS   1F
       END
```



Assembler

SOURCE PROGRAM	FIRST PASS	SECOND PASS
JOHN START 0 USING *, 15 L 1, FIVE A 1,FOUR ST 1, TEMP FOUR DC F'4' FIVE DC F'5' TEMP DS 1F END		

Register 15,base register and at execution time will contain the address of the first instruction of the program

Assembler

SOURCE PROGRAM	FIRST PASS	SECOND PASS
JOHN START 0 USING *, 15 L 1, FIVE A 1,FOUR ST 1, TEMP FOUR DC F'4' FIVE DC F'5' TEMP DS 1F END	0 L 1,_(0,15) 4 A 1,_(0,15) 8 ST 1,_(0,15) 12 4 16 5 20 --	

Register 15,base register and at execution time will contain the address of the first instruction of the program

Assembler

SOURCE PROGRAM	FIRST PASS	SECOND PASS
JOHN START 0		
USING *, 15		
L 1, FIVE	0 L 1,_(0,15)	0 L 1, 16(0,15)
A 1,FOUR	4 A 1,_(0,15)	4 A 1, 12(0,15)
ST 1, TEMP	8 ST 1,_(0,15)	8 ST 1, 20(0,15)
FOUR DC F'4'	12 4	12 4
FIVE DC F'5'	16 5	16 5
TEMP DS 1F	20 --	20 --
END		

Register 15,base register and at execution time will contain the address of the first instruction of the program

Example : Assembly Program

LC	Statement no	Source Code		
	1	PRGM2	START	0
	2		USING	*,15
	3		LA	15,SETUP (RX Format)
	4		SR	TOTAL,TOTAL (RR Format)
	5	AC	EQU	2
	6	INDEX	EQU	3
	7	TOTAL	EQU	4
	8	DATABASE	EQU	13
	9	SETUP	EQU	*

Base table	
Register	Value

Symbol table	
Symbol	Value

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0 (Start address)
	2		USING	*,15
	3		LA	15,SETUP (RX Format)
	4		SR	TOTAL,TOTAL (RR Format)
	5	AC	EQU	2
	6	INDEX	EQU	3
	7	TOTAL	EQU	4
	8	DATABASE	EQU	13
	9	SETUP	EQU	*

Base table	
Register	Value

Symbol table	
Symbol	Value

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0
0	2		USING	*,15
	3		LA	15,SETUP (RX Format)
	4		SR	TOTAL,TOTAL (RR Format)
	5	AC	EQU	2
	6	INDEX	EQU	3
	7	TOTAL	EQU	4
	8	DATABASE	EQU	13
	9	SETUP	EQU	*

Base table	
Register	Value
15	0

* - Use value of LC

USING specifies contents of Base register

Symbol table	
Symbol	Value

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0
0	2		USING	*,15
0	3		LA	15,SETUP (RX Format)
	4		SR	TOTAL,TOTAL (RR Format)
	5	AC	EQU	2
	6	INDEX	EQU	3
	7	TOTAL	EQU	4
	8	DATABASE	EQU	13
	9	SETUP	EQU	*

Base table	
Register	Value
15	0

Symbol table	
Symbol	Value
SETUP	

Value of SETUP unknown

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0
0	2		USING	*,15
0	3		LA	15,SETUP (RX Format)
4	4		SR	TOTAL,TOTAL (RR Format)
	5	AC	EQU	2
	6	INDEX	EQU	3
	7	TOTAL	EQU	4
	8	DATABASE	EQU	13
	9	SETUP	EQU	*

Base table	
Register	Value
15	0

RX Format instruction is 4 bytes long

Symbol table	
Symbol	Value
SETUP	
TOTAL	

Value of TOTAL unknown

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0
0	2		USING	*,15
0	3		LA	15,SETUP (RX Format)
4	4		SR	TOTAL,TOTAL (RR Format)
6	5	AC	EQU	2
	6	INDEX	EQU	3
	7	TOTAL	EQU	4
	8	DATABASE	EQU	13
	9	SETUP	EQU	*

Put symbols in symbol table

Value specified by EQU is absolute

Base table	
Register	Value
15	0

RR Format instruction is 2 bytes long

Symbol table	
Symbol	Value
SETUP	
TOTAL	4
AC	2
INDEX	3
DATABASE	13

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0
0	2		USING	*,15
0	3		LA	15,SETUP (RX Format)
4	4		SR	TOTAL,TOTAL (RR Format)
6	5	AC	EQU	2
6	6	INDEX	EQU	3
6	7	TOTAL	EQU	4
6	8	DATABASE	EQU	13
6	9	SETUP	EQU	*

EQU is a directive

Value of LC is not affected

Here use value of LC as value of SETUP (Value is relative since it uses current LC)

Base table	
Register	Value
15	0

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0
0	2		USING	*,15
0	3		LA	15,SETUP (RX Format)
4	4		SR	TOTAL,TOTAL (RR Format)
6	5	AC	EQU	2
6	6	INDEX	EQU	3
6	7	TOTAL	EQU	4
6	8	DATABASE	EQU	13
6	9	SETUP	EQU	*

Base table	
Register	Value
15	0

Symbol table	
Symbol	Value
SETUP	6

Calc value of offset for SETUP

(Base index addressing is used for symbols)

Offset = symbol value – base reg value

Offset = 6 – 0 = 6

Base reg = 15

Index reg = 0

Instruction becomes – **LA 15, 6(0,15)**

Offset

Index

Base

Example : Assembly Program

LC	Statement no	Source Code		
0	1	PRGM2	START	0
0	2		USING	*,15
0	3		LA	15,SETUP (RX Format)
4	4		SR	TOTAL,TOTAL (RR Format)
6	5	AC	EQU	2
6	6	INDEX	EQU	3
6	7	TOTAL	EQU	4
6	8	DATABASE	EQU	13
6	9	SETUP	EQU	*

Base table	
Register	Value
15	0

For RR Format instructions, value of symbol is taken directly. Instruction becomes – **SR 4,4**

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13

Example : Assembly Program

LC	Statement no	Source Code			
6	10		USING	SETUP,15	
	11		L	DATABASE,=A(DATA1)	(RX)
	12		USING	DATAAREA,DATABASE	
	13		SR	INDEX,INDEX	(RR)
	14	LOOP	L	AC,DATA1(INDEX)	(RX)

USING specifies contents of base reg.
Base reg 15 has value of symbol SETUP

Base table	
Register	Value
15	6

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13

Example : Assembly Program

LC	Statement no	Source Code			
6	10		USING	SETUP,15	
6	11		L	DATABASE,=A(DATA1)	(RX)
	12		USING	DATAAREA,DATABASE	
	13		SR	INDEX,INDEX	(RR)
	14	LOOP	L	AC,DATA1(INDEX)	(RX)

Literal table	
Literal	Value
A(DATA1)	

Values starting with '=' is a literal.
Add to Literal Table
Value Unknown

A(DATA1) means Address of Symbol DATA1

Base table	
Register	Value
15	6

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13

Example : Assembly Program

LC	Statement no	Source Code			
6	10		USING	SETUP,15	
6	11		L	DATABASE,=A(DATA1)	(RX)
10	12		USING	DATAAREA,DATABASE	
	13		SR	INDEX,INDEX	(RR)
	14	LOOP	L	AC,DATA1(INDEX)	(RX)

Base table	
Register	Value
15	6
13	DATAAREA

Literal table	
Literal	Value
A(DATA1)	

USING specifies value of base reg DATABASE (13) as value of symbol DATAAREA (add to symbol table, value unknown)

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	

Example : Assembly Program

LC	Statement no	Source Code			
6	10		USING	SETUP,15	
6	11		L	DATABASE,=A(DATA1)	(RX)
10	12		USING	DATAAREA,DATABASE	
10	13		SR	INDEX,INDEX	(RR)
	14	LOOP	L	AC,DATA1(INDEX)	(RX)

Literal table	
Literal	Value
A(DATA1)	

For RR Format instructions, value of symbol is taken directly.
Instruction becomes – **SR 3,3**

Base table	
Register	Value
15	6
13	DATAAREA

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	

Example : Assembly Program

LC	Statement no	Source Code			
6	10		USING	SETUP,15	
6	11		L	DATABASE,=A(DATA1)	(RX)
10	12		USING	DATAAREA,DATABASE	
10	13		SR	INDEX,INDEX	(RR)
12	14	LOOP	L	AC,DATA1(INDEX)	(RX)

Literal table	
Literal	Value
A(DATA1)	

Add DATA1 to symbol table.
Value unknown
Add LOOP to symbol table
with value of LC

Base table	
Register	Value
15	6
13	DATAAREA

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12

Example : Assembly Program

LC	Statement no	Source Code			
12	14	LOOP	L	AC,DATA1(INDEX)	(RX)
16	15		AR	TOTAL,AC	(RR)
	16		A	AC,=F'5'	(RX)
	17		ST	AC,SAVE(INDEX)	(RX)

Literal table	
Literal	Value
A(DATA1)	

For RR Format instructions, value of symbol is taken directly.
Instruction becomes – **AR 4,2**

Base table	
Register	Value
15	6
13	DATAAREA

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12

Example : Assembly Program

LC	Statement no	Source Code			
12	14	LOOP	L	AC,DATA1(INDEX)	(RX)
16	15		AR	TOTAL,AC	(RR)
18	16		A	AC,=F'5'	(RX)
	17		ST	AC,SAVE(INDEX)	(RX)

Literal table	
Literal	Value
A(DATA1)	
F'5	

Values starting with '=' is a literal.
Add to Literal Table
Value Unknown

Base table	
Register	Value
15	6
13	DATAAREA

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12

Example : Assembly Program

LC	Statement no	Source Code			
12	14	LOOP	L	AC,DATA1(INDEX)	(RX)
16	15		AR	TOTAL,AC	(RR)
18	16		A	AC,=F'5'	(RX)
22	17		ST	AC,SAVE(INDEX)	(RX)

Literal table	
Literal	Value
A(DATA1)	
F'5	

Add SAVE to symbol table.
Value unknown
Index reg value is specified in brackets

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code	
22	17	ST	AC,SAVE(INDEX)
26	18	A	INDEX, =F'4'
	19	C	INDEX, =F'8000'
	20	BNE	LOOP

Literal table	
Literal	Value
A(DATA1)	
F'5	
F'4	

Values starting with '=' is a literal.
Add to Literal Table
Value Unknown

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code	
22	17	ST	AC,SAVE(INDEX)
26	18	A	INDEX, =F'4'
30	19	C	INDEX, =F'8000'
	20	BNE	LOOP

Literal table	
Literal	Value
A(DATA1)	
F'5	
F'4	
F'8000	

Values starting with '=' is a literal.
Add to Literal Table
Value Unknown

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code	
22	17	ST	AC,SAVE(INDEX)
26	18	A	INDEX, =F'4'
30	19	C	INDEX, =F'8000'
34	20	BNE	LOOP

Literal table	
Literal	Value
A(DATA1)	
F'5	
F'4	
F'8000	

BNE – Branch if not equal
Equivalent to **BC 7,label**
Where 7 is the default reg and label is the label specified
Offset value to be calculated

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code
34	20	BNE LOOP
38	21	LR 1, LOOP (RR)
	22	BR 14
	23	LTORG

Literal table	
Literal	Value
A(DATA1)	
F'5	
F'4	
F'8000	

For RR Format instructions, value of symbol is taken directly.
Instruction becomes – **LR 1,12**

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code
34	20	BNE LOOP
38	21	LR 1, LOOP (RR)
40	22	BR 14 (RR)
	23	LTORG

Literal table	
Literal	Value
A(DATA1)	
F'5	
F'4	
F'8000	

BR – Unconditional Branch from current register to specified register
Equivalent to **BCR 15, 14**
(15 is current base reg)

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code
34	20	BNE LOOP
38	21	LR 1, LOOP (RR)
40	22	BR 14
48	23	LTORG

Literal table	
Literal	Value
A(DATA1)	
F'5	
F'4	
F'8000	

LTORG forces LC to **jump** to next word boundary.
Word boundaries are 16, 32, 48,.....
Next boundary will be at 48
Literals are assigned storage space at this point

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code
34	20	BNE LOOP
38	21	LR 1, LOOP (RR)
40	22	BR 14
48	23	LTORG

Literal table	
Literal	Value
A(DATA1)	48
F'5	52
F'4	56
F'8000	60

Literals are assigned storage at current LC which is 48.

Each literal will take 4 bytes.

Next LC will be at 60+4 = 64

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	

Example : Assembly Program

LC	Statement no	Source Code		
64	24	SAVE	DS	2000F
8064	25	DATAAREA	EQU	*
	26	DATA1 DC	F'25,26,97,…..[2000 nos]	
	27		END	

Literal table	
Literal	Value
A(DATA1)	48
F'5	52
F'4	56
F'8000	60

DS – Define Storage of size
Value as specified.
Type of value can be
C – Character (1 byte)
H – Half Word (2 byte)
F – Full Word (4 byte)
Q – Quad Word (8 byte)

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	
DATA1	
LOOP	12
SAVE	64

DS forces LC to be **incremented** by (value * Type)
LC = LC+(2000*4) = 64+8000 = 8064

Example : Assembly Program

LC	Statement no	Source Code		
64	24	SAVE	DS	2000F
8064	25	DATAAREA	EQU	*
	26	DATA1 DC	F'25,26,97,…..[2000 nos]	
	27		END	

Base table	
Register	Value
15	6
13	8064

Here use value of LC as value of DATAAREA (Value is relative since it uses current LC)

Update Base table with value of DATAAREA

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	8064
DATA1	
LOOP	12
SAVE	64

Example : Assembly Program

LC	Statement no	Source Code		
64	24	SAVE	DS	2000F
8064	25	DATAAREA	EQU	*
8064	26	DATA1	DC	F'25,26,97,…..[2000 nos]
	27		END	

Base table	
Register	Value
15	6
13	8064

Here use value of LC as value of DATA1

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	8064
DATA1	8064
LOOP	12
SAVE	64

DC – Define constant of Type Full Word (4 bytes)
2000 numbers are stored sequentially starting from 8064 (this storage area was defined by statement 24)

Example : Assembly Program

LC	Statement no	Source Code		
64	24	SAVE	DS	2000F
8064	25	DATAAREA	EQU	*
8064	26	DATA1 DC	F'25,26,97,…..[2000 nos]	
	27		END	

Base table	
Register	Value
15	6
13	8064

Reached End of program

Symbol table	
Symbol	Value
SETUP	6
TOTAL	4
AC	2
INDEX	3
DATABASE	13
DATAAREA	8064
DATA1	8064
LOOP	12
SAVE	64

Intermediate Generated Code

Stmt no	Location	Instruction	
3	0	LA	15, 6(0,15)
4	4	SR	4, 4
11	6	L	13, __(0,15)
13	10	SR	3, 3
14	12	L	2, __(3,13)
15	16	AR	4, 2
16	18	A	2, __(0,15)
17	22	ST	2, __(3,15)
18	26	A	3, __(0,15)
19	30	C	3, __(0,15)
20	34	BC	7, __(0,15)
21	38	LR	1, 4
22	40	BCR	15, 14
23	48	8064	
	52	5	
	56	4	
	60	8000	

Points to remember :-

- For all symbols and literals used in RX Format, Base index addressing is used.
- It is of the form – **offset (index, base)**
- To calculate offset, value of current base register is subtracted from symbol/literal value, index register value is specified in instruction inside brackets (if not specified then value is 0)
- Base register that is chosen is register with value closest to symbol/literal value.

Assembler Example→

- Till statement 10, value of base register 15 is 0
- After statement 10, value of base register 15 is 6
- After statement 12, value of base register 13 is 8064

Statement 11 – L DATABASE,=A(DATA1)

Symbol/ Literal	Value	Base Register	Value	Offset	Final Value (with index)
A(DATA1)	48	15	6	48 – 6 = 42	42(0,15)

Statement 11 – L 13, 42(0,15)

Assembler Example→

Statement 14 – LOOP L AC,DATA1(INDEX)

Symbol/ Literal	Value	Base Register	Value	Offset	Final Value (with index)
A(DATA1)	48	15	6	48 – 6 = 42	42(0,15)
DATA1	8064	13	8064	8064 – 8064 = 0	0(3,13)

Statement 14 – LOOP L 2, 0(3,13)

Assembler Example→

Statement 16 – A AC,=F'5'

Symbol/ Literal	Value	Base Register	Value	Offset	Final Value (with index)
A(DATA1)	48	15	6	48 – 6 = 42	42(0,15)
DATA1	8064	13	8064	8064 – 8064 = 0	0(3,13)
F'5	52	15	6	52 – 6 = 46	46(0,15)

Statement 16 – A 2, 46(0,15)

Assembler Example→

Statement 17 – ST AC,SAVE(INDEX)

Symbol/ Literal	Value	Base Register	Value	Offset	Final Value (with index)
A(DATA1)	48	15	6	48 – 6 = 42	42(0,15)
DATA1	8064	13	8064	8064 – 8064 = 0	0(3,13)
F'5	52	15	6	52 – 6 = 46	46(0,15)
SAVE	64	15	6	64 – 6 = 58	58(3,15)

Statement 17 – ST 2, 58(3,15)

Assembler Example→

Statement 18 – A INDEX, =F'4'

Symbol/ Literal	Value	Base Register	Value	Offset	Final Value (with index)
A(DATA1)	48	15	6	48 – 6 = 42	42(0,15)
DATA1	8064	13	8064	8064 – 8064 = 0	0(3,13)
F'5	52	15	6	52 – 6 = 46	46(0,15)
SAVE	64	15	6	64 – 6 = 58	58(3,15)
F'4	56	15	6	56 – 6 = 50	50(0,15)

Statement 18 – A 3, 50(0,15)

Assembler Example→

Statement 19 – C INDEX, =F'8000'

Symbol/ Literal	Value	Base Register	Value	Offset	Final Value (with index)
A(DATA1)	48	15	6	48 – 6 = 42	42(0,15)
DATA1	8064	13	8064	8064 – 8064 = 0	0(3,13)
F'5	52	15	6	52 – 6 = 46	46(0,15)
SAVE	64	15	6	64 – 6 = 58	58(3,15)
F'4	56	15	6	56 – 6 = 50	50(0,15)
F'8000	60	15	6	60 – 6 = 54	54(0,15)

Statement 19 – C 3, 54(0,15)

Assembler Example→

Statement 20 – BNE LOOP

Symbol/ Literal	Value	Base Register	Value	Offset	Final Value (with index)
A(DATA1)	48	15	6	48 – 6 = 42	42(0,15)
DATA1	8064	13	8064	8064 – 8064 = 0	0(3,13)
F'5	52	15	6	52 – 6 = 46	46(0,15)
SAVE	64	15	6	64 – 6 = 58	58(3,15)
F'4	56	15	6	56 – 6 = 50	50(0,15)
F'8000	60	15	6	60 – 6 = 54	54(0,15)
LOOP	12	15	6	12 – 6 = 6	6(0,15)

Statement 20 – BC 7, 6(0,15)

Final Generated Code

Stmt no	Location	Instruction	
3	0	LA	15, 6(0,15)
4	4	SR	4, 4
11	6	L	13, __(0,15)
13	10	SR	3, 3
14	12	L	2, __(3,13)
15	16	AR	4, 2
16	18	A	2, __(0,15)
17	22	ST	2, __(3,15)
18	26	A	3, __(0,15)
19	30	C	3, __(0,15)
20	34	BC	7, __(0,15)
21	38	LR	1, 12
22	40	BCR	15, 14
23	48	8064	
	52	5	
	56	4	
	60	8000	

Practice :-Compute Pass 1 & Pass 2

SOURCE PROGRAM		
TEST	START	
BEGIN	BALR	15, 0
	USING	BEGIN+2, 15
	SR	4, 4
	L	3, TEN
LOOP	L	2, DATA(4)
	A	2, FORTY9
	ST	2, DATA(4)
	A	4, FOUR
	BCT	3, LOOP
	BR	14
TEN	DC	F'10'
FOUR	DC	F'4'
FORTY9	DC	F'49'
DATA	DC	F'1,3,3,3,3,4,5,8,9,0'
	END	

Solution :-

SOURCE PROGRAM		
TEST	START	
BEGIN	BALR	15, 0
	USING	BEGIN+2, 15
	SR	4, 4
	L	3, TEN
LOOP	L	2, DATA(4)
	A	2, FORTY9
	ST	2, DATA(4)
	A	4, FOUR
	BCT	3, LOOP
	BR	14
TEN	DC	F'10'
FOUR	DC	F'4'
FORTY9	DC	F'49'
DATA	DC	F'1,3,3,3,3,4,5,8,9,0'
	END	

FIRST PASS		
0	BALR	15, 0
2	SR	4, 4
4	L	3, -(0, 15)
8	L	2, -(4,15)
12	A	2, -(0, 15)
16	ST	2, -(4, 15)
20	A	4, -(0,15)
24	BCT	3, *-16
28	BCR	15, 14
32	10	
36	4	
40	49	
44	1	
48	3	
52	3	
56	3	
60	3	
64	4	
68	5	
72	8	
76	9	
80	0	

SECOND PASS		
0	BALR	
2	SR	4, 4
4	L	3, 32(0, 15)
8	L	2, 44(4, 15)
12	A	2, 40(0, 15)
16	ST	2, 44(4, 15)
20	A	4, 36(0, 15)
24	BCT	3, 8(0, 15)
28	BCR	15, 14
32	10	
36	4	
40	49	
44	1	
48	3	
52	3	
56	3	
60	3	
64	4	
68	5	
72	8	
76	9	
80	0	

Practice :-

For the following assembly language program Construct ST, LT, BT and formulate the corresponding object code using these tables

```
1.  CSE     START 0        ; Beginning of the CSE program
2.          USING   *,4
3.          SR    6,6      ; Clear register 6
4.          L     6,MARK
5.          A     6,QUIZ
6.          USING *,5
7.          ST    6, MARK
8.          ST    6,COPY
9.          C     6,=F'85'
10. MARK    DC    F'79'
11. QUIZ    DC    F'7'
12. COPY    DS    1F
13.         LTORG
14.         END
```

Assembler Design

General Design Procedure of Assembler

1. Specify the problem
2. Specify data structures
3. Define format of data structures
4. Specify algorithm
5. Look for modularity [capability of one program to be subdivided into independent programming units.]
6. Repeat 1 through 5 on modules.

Assembler Design

General Design Procedure of Assembler

1. Specify the problem :

Pass1: Define symbols & literals.

1) Determine length of m/c instruction [MOTGET1]
2) Keep track of Location Counter [LC]
3) Remember values of symbols until pass2 [STSTO]
4) Process some pseudo ops, eg. EQU, DS etc [POTGET1]
5) Remember Literals [LITSTO]

Assembler Design

General Design Procedure of Assembler

1. Specify the problem :

Pass2: Generate object program

1) Look up value of symbols [STGET]
2) Generate instruction [MOTGET2]
3) Generate data (for DS, DC & literals)
4) Process pseudo ops [POTGET2]

Assembler Design

General Design Procedure of Assembler

2. Specify Data Structure:

Pass1: Databases

- Input source program
- "LC" location counter is used to keep track of each instruction's address.
- Machine operation table (MOT) [Symbolic mnemonic & length]
- Pseudo operation table [POT], [pseudo mnemonic & action]
- Symbol Table (ST) stores each table & it's value.
- Literal Table (LT), to store each literal (variable) & it's location.
- Copy of input to used later by PASS-2.

Assembler Design

General Design Procedure of Assembler

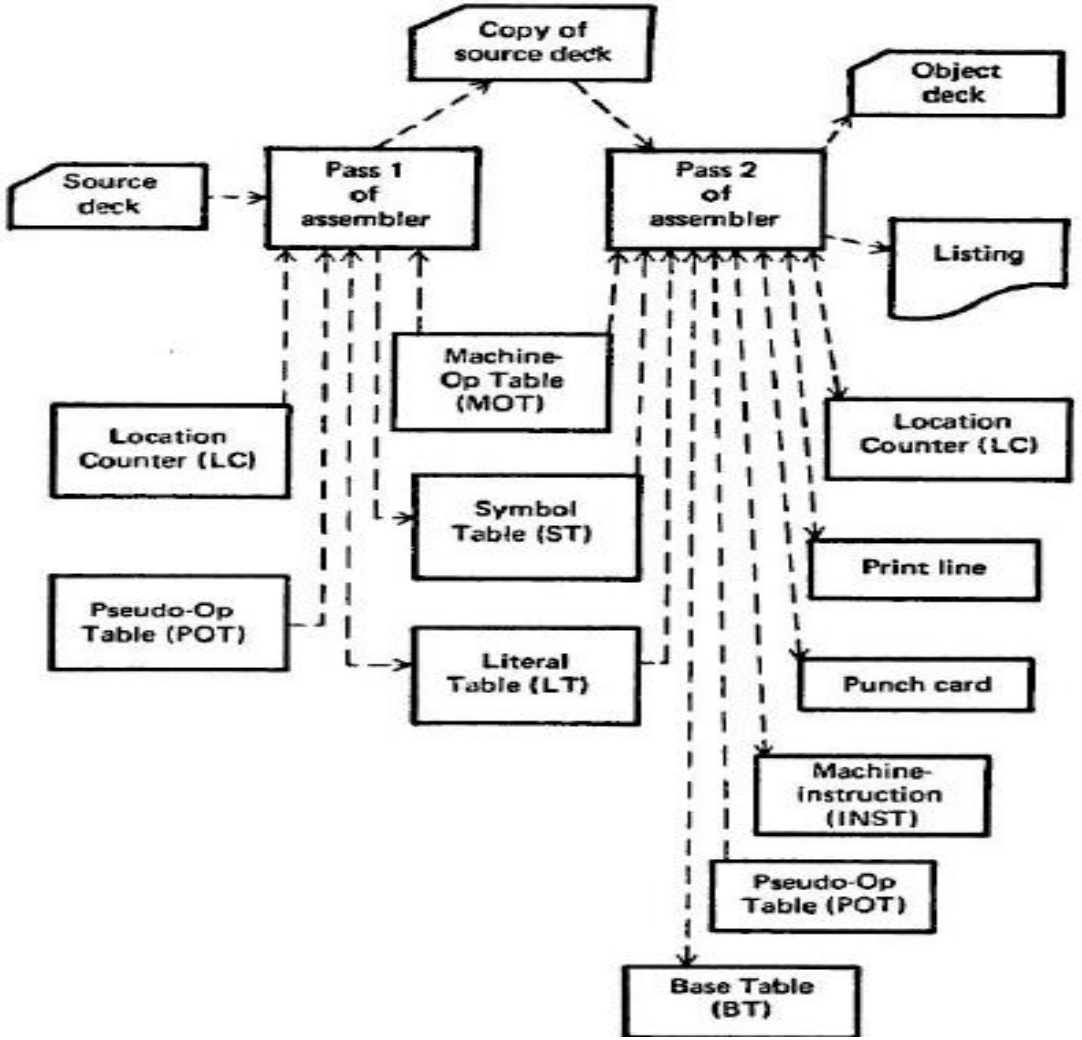
2. Specify Data Structure:

Pass2: Databases

- Copy of source program input to Pass1.
- Location Counter (LC)
- MOT [Mnemonic, length, binary m/c op code, etc.]
- POT [Mnemonic & action to be taken in Pass2
- ST [prepared by Pass1, label & value]
- Base Table [or register table] indicates which registers are currently specified using 'USING' pseudo op & what are contents.
- Literal table prepared by Pass1. [Lit name & value].

Assembler Design

Use of Databases by Assembler Passes



Assembler Design

General Design Procedure of Assembler

3. Format of Data Structure:

- Machine Operation Table
  - The op-code is the key and it's value is the binary op code equivalent, which is used for use in generating machine code.
  - The instruction length is stored for updating the location counter.
  - Instruction format is use in forming the m/c language equivalent

Assembler Design

General Design Procedure of Assembler

3. Format of Data Structure:

- Machine Operation Table (MOT)

6-bytes per entry

Mnemonic op-code (4-bytes) (characters)	Binary op-code (1-byte) (hexadecimal)	Instruction length (2-bits) (binary)	Instruction format (3-bits) (binary)	Not used in this design (3-bits)
"A*bbb*"	5A	10	001	
"AH*bb*"	4A	10	001	
"AL*bb*"	5E	10	001	
"ALR*b*"	1E	01	000	
"AR*bb*"	1A	01	000	
...	...	...	...	
"MVC*b*"	D2	11	100	
...	...	...	...	

b ~ represents the character "blank"

Codes:

Instruction length

01 = 1 half-words = 2 bytes
10 = 2 half-words = 4 bytes
11 = 3 half-words = 6 bytes

Instruction format

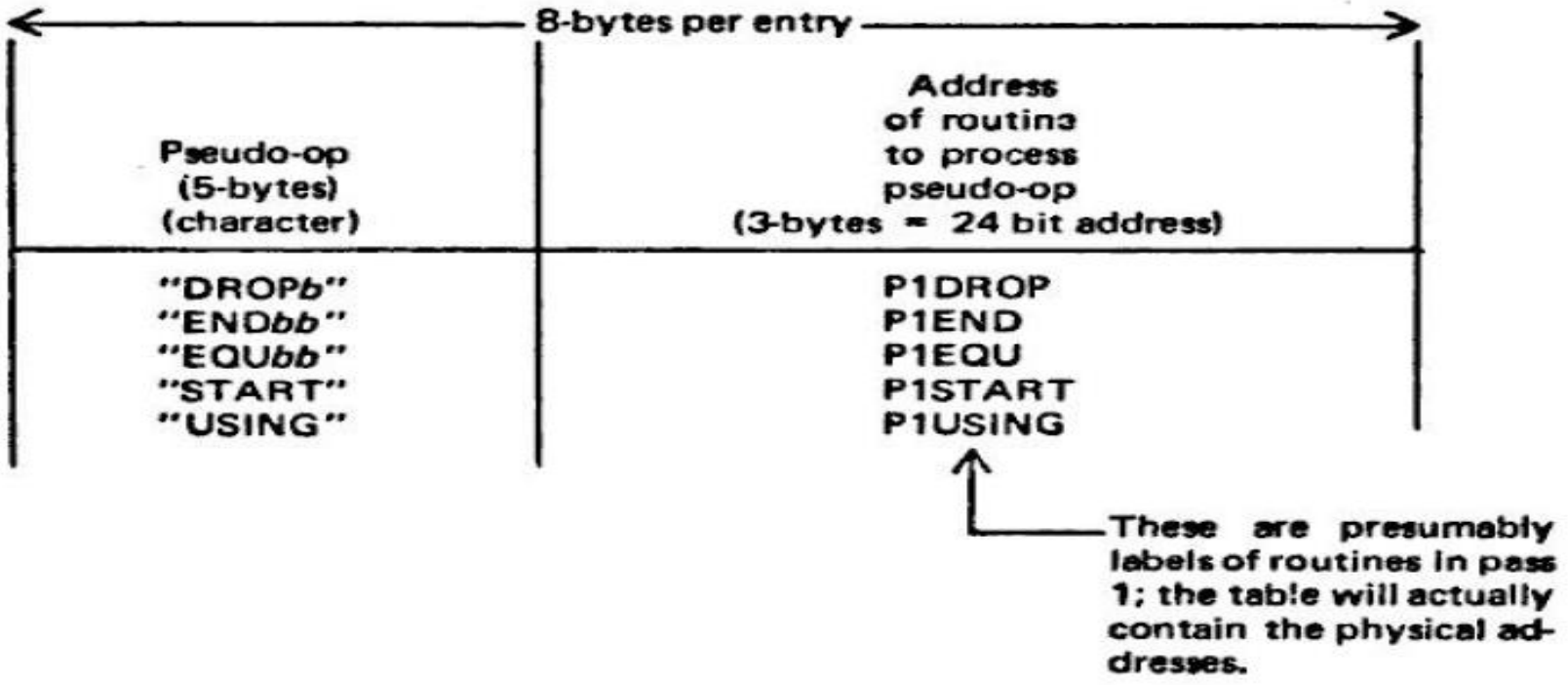
000 = RR
001 = RX
010 = RS
011 = SI
100 = SS

Assembler Design

General Design Procedure of Assembler

3. Format of Data Structure:

- Pseudo Operation Table (POT)



Assembler Design

General Design Procedure of Assembler

3. Format of Data Structure:

- Symbol Table (ST) and Literal Table (LT)

14-bytes per entry

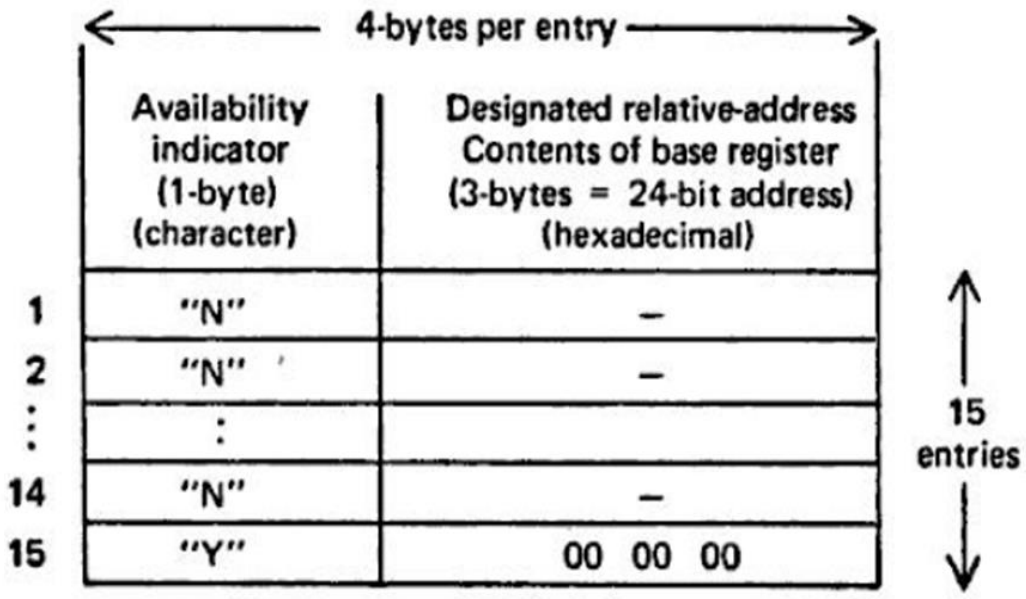
Symbol (8-bytes) (characters)	Value (4-bytes) (hexadecimal)	Length (1-byte) (hexadecimal)	Relocation (1-byte) (character)
"JOHN*bbbb*"	0000	01	"R"
"FOUR*bbbb*"	000C	04	"R"
"FIVE*bbbb*"	0010	04	"R"
"TEMP*bbbb*"	0014	04	"R"

Assembler Design

General Design Procedure of Assembler

3. Format of Data Structure:

- Base Table (BT)



Code=

Availability

Y ~ register specified in USING pseudo-op

N ~ register never specified in USING pseudo-op or subsequently made unavailable by the DROP pseudo-op

Assembler Design

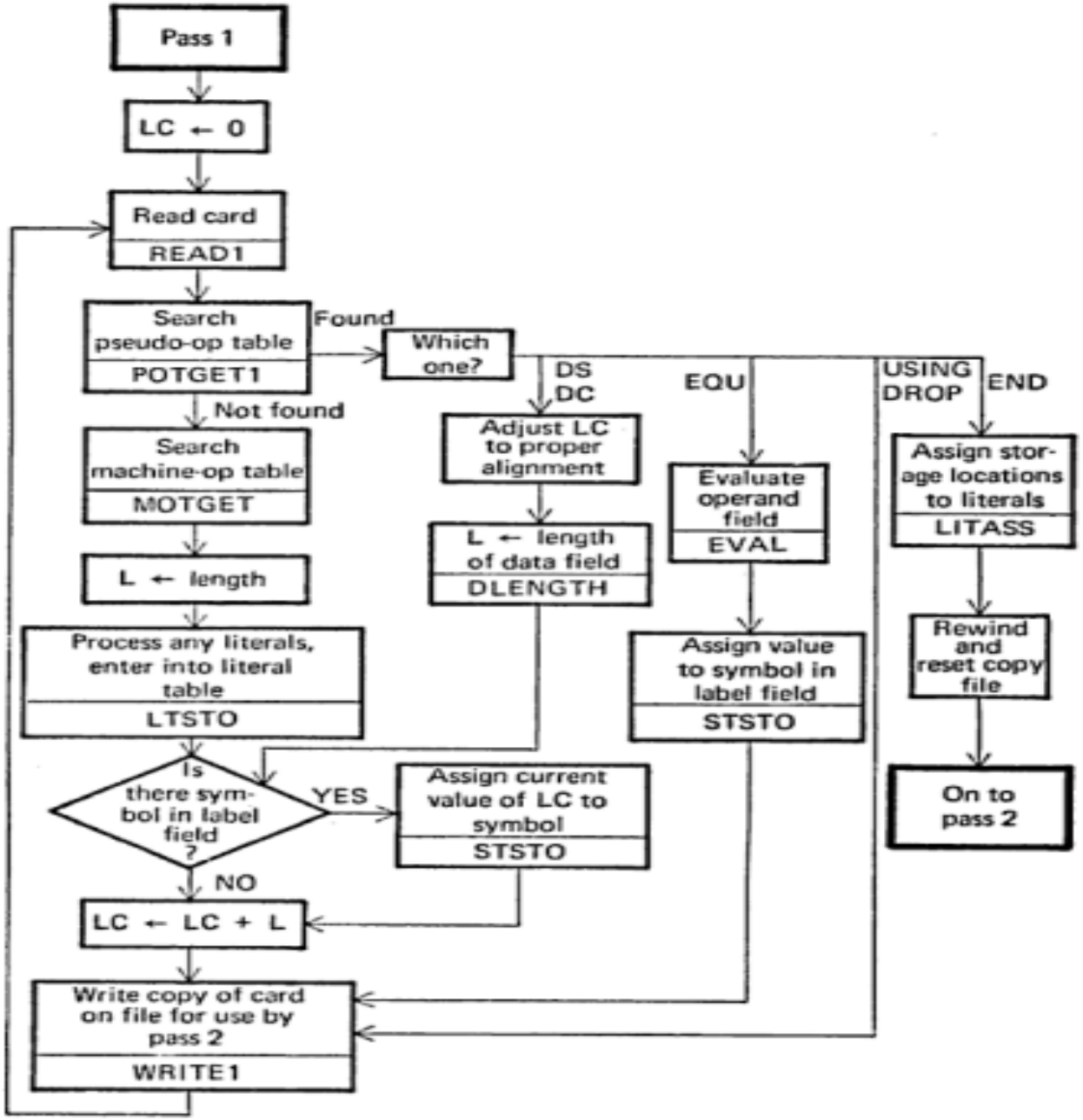
What goes where?

- ST (Symbol Table) – Labels & their values
- LT (Literal Table) – Data (F'10', F'49', F'4'…)
- MOT (Machine Op Table) – Machine opcodes, i.e., instructions along with their length (2/4/6 bytes) in Pass1.
- **In Pass2**, we store the name, length, binary code and format (RR, RX, SS…).
- POT (Pseudo Op Table) – Pseudo opcodes or action to be taken in Pass1 and Pass2
- BT (Base Table) – the registers used for base addresses

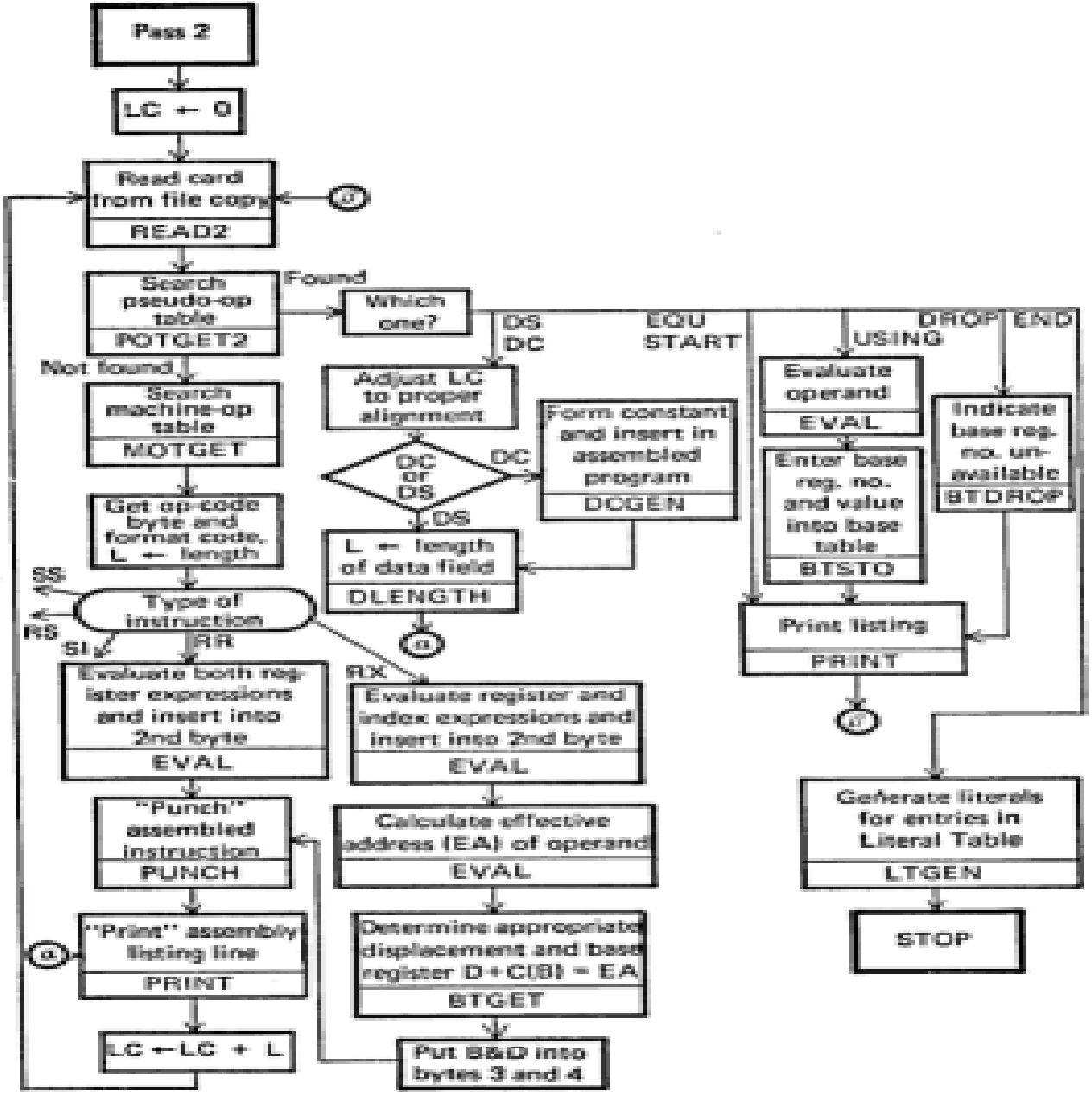
Problems with single pass assembly?

- Forward Reference
  - Omits the operand address if the symbol has not yet been defined
  - Enters this undefined symbol into SYMTAB and indicates that it is undefined
  - Adds the address of this operand address to a list of forward references associated with the SYMTAB entry
  - When the definition for the symbol is encountered, scans the reference list and inserts the address.
  - At the end of the program, reports the error if there are still SYMTAB entries indicated undefined symbols.

Pass 1- Assembler Flow chart



Pass 2- Assembler Flow chart



Assembler Design

General Design Procedure of Assembler

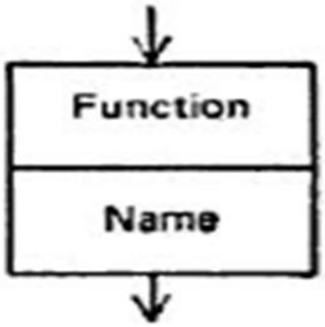
5. Look for Modularity :-

We, now review our design, looking for functions that can be isolated,

Mostly such functions fall in two categories:

1. Multiuse
2. Unique

Some functions that may be isolated in the two passes



Assembler

PASS 1:

1. READ1	———	Read the next assembly source card.
2. POTGET1	———	*Search* the pass 1 Pseudo-Op Table (POT) for a match with the operation field of the current source card.
3. MOTGET1	———	*Search* the Machine-Op Table (MOT) for a match with the operation of the current source card.
4. STSTO	———	*Store* a label and its associated value into the Symbol Table (ST). If the symbol is already in the table, return error indication (multiply-defined symbol).
5. LTSTO	———	*Store* a literal into the Literal Table (LT); do not store the same literal twice.
6. WRITE1	———	Write a copy of the assembly source card on a storage device for use by pass 2.
7. DLENGTH	———	*Scan* operand field of DS or DC pseudo-op to determine the amount of storage required.
8. EVAL	———	*Evaluate* an arithmetic expression consisting of constants and symbols (e.g. 6, ALPHA, BETA + 4 * GAMMA).
9. STGET	———	*Search* the Symbol Table (ST) for the entry corresponding to a specific symbol (used by STSTO, and EVAL).
10. LITASS	———	Assign storage locations to each literal in the literal table (may use DLENGTH).

Assembler

PASS 2:

1. READ2	——	Read the next assembly source card from the file copy.
2. POTGET2	——	Similar to POTGET1 (search POT).
3. MOTGET2	——	Same as in pass 1 (search MOT).
4. EVAL	——	Same as in pass 1 (evaluate expressions).
5. PUNCH	——	Convert generated instruction to card format; punch card when it is filled with data.
6. PRINT	——	Convert relative location and generated code to character format; print the line along with copy of the source card.
7. DCGEN	——	Process the fields of the DC pseudo-op to generate object code (uses EVAL and PUNCH).
8. DLENGTH	——	Same as in pass 1.
9. BTSTO	——	Insert data into appropriate entry of Base Table (BT).
10. BTDROP	——	Insert "unavailable" indicator into appropriate entry of BT.
11. BTGET	——	Convert effective address into base and displacement by searching Base Table (BT) for available base registers.
12. LTGEN	——	Generate code for literals (uses DCGEN).

University Questions

- State the reason for the assembler to be a multi pass program
- What is forward reference problem? How is it handled in two pass assembler? Explain with the help of flowchart the working of a two pass assembler.
- Explain the databases used by each pass of the 2-pass assembler in detail.
- With reference to assembler explain the following tables with suitable examples:

(a) POT (b) MOT (c) ST (d) LT

- Explain the working of single pass assembler. Show the structure of the database used
- Practice Assembler programs

THE END!

Have a nice day!