5.2 Language Translator Notes

Objective:

- Show understanding of the need for;
  - Assembler software for the translation of an assembly language program.
  - A compiler for the translation of a high-level language program.
  - An interpreter for translation and execution of a high-level language program.
- Benefits and drawbacks of using either compiler or interpreter and justify use of each.
- Show awareness that high-level language programs may be partially compiled and partially interpreted, such as Java.
- Describe features found in a typical Integrated Development Environment (IDE)

  Including: • For coding, including context-sensitive prompts • For initial error detection, including dynamic syntax checks • For presentation, including prettyprint, expand and collapse code blocks • For debugging, including single stepping, breakpoints, i.e. variables, expressions, report window.

Language Translators

Language Translators are programs that translate **source code** written in Assembly Language or High Level Language into **object code** of machine language.

Types of Language Translators:

- **Assembler:**

  Language translator that translates **source program** written in **Assembly** Language into **executable object code** in machine language.

Assembly Language:

Assembly language uses **mnemonic code** to represent each low-level machine instruction or opcode. Assembly language program contain features that are used to help programmer to inform assembler program. Assembly language programs are **machine dependent**; they are not **portable** from one type of computer/chip to another.

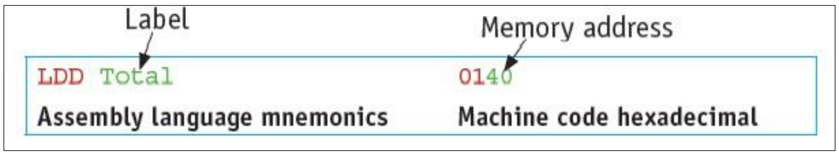
Types of Assembler:

- **Single Pass Assemblers**:

  A single pass assembler puts **machine code** instructions straight into **computer memory** to be executed.

- **Two Pass Assemblers.**

  In *Two-pass assemblers* source code will have to be scanned **twice** in order to complete **symbol table** and produce final **machine code**. During scanning, assembler replaces **labels** in assembly language program with memory **addresses** in **machine code** program. It produces **object code** that can be stored, loaded then executed later.



First Pass of Two-Pass Assembler: Typical actions are:

- Read assembly language program line by line.
- Removal of **comments** and **white spaces**.
- **Data values** are converted to **binary values**. **Opcode** converted to binary in **pass two**.
- Check **opcode** for errors.
- Creation of **Symbol Table** containing **binary codes** for symbolic names and labels.
- If **absolute address** is known it is entered in **symbol table** else it is marked unknown.
- Creation of a **literal table** if programmer has used **constants** in program.
- Identification of **System Calls** and **Subroutines** used.
- Expansion of **macros**. Macro is a **sequence** of instructions, assigned by a name and could be used anywhere in program.
- Any **directives** are acted upon – directives are **special instruction** for assembler.

Second Pass of two-pass Assembler:

- Read assembly language program **line by line**.
- Any **symbolic address** is replaced by an **absolute address**.
- **Forward references** are resolved.
- **Symbolic opcode** are replaced with **binary opcode** using opcode table.
- Generate object code – using symbol table created in pass 1.
- For second pass **opcode table** is required.
- If errors are not found, **Second Pass** of assembler generates **object code**.

Steps in Creation of symbol table – (entering symbolic addresses in symbol table)

- Assembler scans assembly language instructions in **sequence**.
- When it meets a symbolic address checks to see if already in symbol table.
- If not, it adds it to symbol table in symbolic address column
- If it is already in symbol table check if absolute address known
- If absolute address is known, it is entered in appropriate cell else it is marked unknown.

FORWARD REFERENCE PROBLEM: Rules for an assembly program states that symbol should be defined somewhere in program. But in some cases a symbol may be used prior to its definition. Such a reference is called forward reference. Due to this assembler cannot translate instructions and such a problem is called **forward reference problem**.

Assemblers either store program directly in main memory, ready for execution, as it is translated, or they store translated program on a **storage medium** to be used later. If stored for later use, then a **loader program** is needed to load stored **translated** program into **main**

memory before it can be executed. Stored translated program can be executed many times without being **re-translated**.

Assembly language programs are written for tasks that need to be **speedily executed**, for example, parts of an operating system, central heating system or controlling a robot.

Example: Steps to fill the symbol table:

Label	Instruction
StartProg:	LDV #Offset
	CMP Value
	JPE EndProg
	OUTCH
	LDD Offset
	INC
	STO Offset
	JMP StartProg
EndProg:	END
Offset:	10
	50
	65
	89
	32
Value:	32

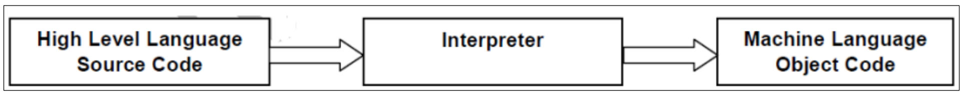
Symbol Table

Symbolic Address		Absolute address			
StartProg	1	0	2		
Offset	3	UNKNOWN	4	9	10
Value	5	UNKNOWN	6	14	11
EndProg	7	UNKNOWN	8	8	9

Interpreters:

It is language translator that translates **source program** written in High Level Language into **object code** in machine language during **step by step** execution of program but **no executable** file of machine code is produced.

Interpreter is used when a program is being **developed**.

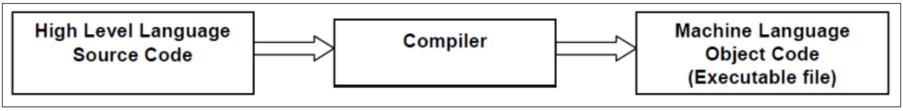


ESQ: How Interpreter Translate source code into object code?

- Interpreter program, source code file and data to be used by source code program are all made available.
- Interpreter program start execution and first line of source code is read and analyzed.
- If **an error** is found this is reported and interpreter program **halts** execution.
- If **no error** is found, line of source code is converted to an intermediate code.
- Interpreter program uses this **intermediate code** to execute required action.
- Next line of source code is read.

Compiler:

Compiler is language translator that translates **whole** source program written in High Level Language into **object code** in machine language before execution of program. Object code is saved as object file, which is **executable**. Object files don't need compiler for execution. A compiled program is usually **distributed** for general use.


High Level Language Source Code
Compiler
Machine Language Object Code (Executable file)

ESQ: List down steps that Compiler takes to translate source code into machine code?

- Compiler program and source code file are made available but no data is needed.
- Compiler program begins execution and first line of source code is read.
- Line is analysed. If an error is found this is recorded.
- If no error is found the line of source code is converted to an **intermediate code**.
- Next line of source code is read.
- When **whole source code** has been dealt with one of following happens:
  - ✓ If **no error** is found in whole source code, complete intermediate code is converted into object code.
  - ✓ If **any errors** are found a list of these is output and no object code is produced.

Execution of program can only begin when compilation has shown **no errors**. And after that, object code is stored and program is executed later with no involvement of compiler.

Advantage of using Interpreter for Programmer:

- When program is being developed, errors can be identified as they occur and corrected without having to wait for whole of source code to be read and analyzed.

Disadvantage of using Interpreter for Programmer :

- When program is error free and is distributed to users, source code has to be sent to each user.

Advantage of using Compiler for Programmer :

- Executable file which is output of compiler can be distributed to users so the users have no access to the source code.

Advantages and disadvantages of using interpreted or compiled programs for users:

- For interpreted program, interpreter and source code have to be available each Time that an error-free program is run.
- For compiled program, only **object code** has to be available each time that an error free program is run.
- Compiled object code provide **faster execution** than an interpreted program.
- Compiled object code is **less secure** because it could contain a virus.

5.2 Language Translator Notes

	Assembler	Compiler	Interpreter
Source program written in	assembly language	high-level language	high-level language
Machine dependent	yes	no	no
Object program generated	yes, stored on disk or in main memory	yes, stored on disk or in main memory	no, instructions are executed under the control of the interpreter
Each line of the source program generates	one machine code instruction, one to one translation	many machine code instructions, instruction explosion	many machine code instructions, instruction explosion

Partial Compiling and Interpreting

To achieve **shorter execution** times, many high-level languages (Java and Python) programs use system that is partially compilation and partially interpretation. Source code is checked and translated by a compiler into object code. Compiled object code is a low-level machine independent code, called **intermediate code**, **p-code** or **bytecode**. To **execute program**, object code can be interpreted by an interpreter or compiled using a compiler.

For Example:

When programming language Java was created. Each computer has to have Java Virtual Machine created for it. When programmer writes a **Java program** this is compiled first to produce **Java Byte Code**. When the program is run, this code is interpreted by the **Java Virtual Machine**. The Java Byte Code can be transferred to any computer that has a **Java Virtual Machine** installed.

Writing a program directly in **machine code** would take very **long time** and undoubtedly would lead to a **multitude** of errors.

Integrated Development Environment (IDE)

Integrated development environment (IDE) is used by programmers to aid writing and development of programs.

Features of IDE

- **Prettyprinting:**

  Prettyprint refers to **presentation** of program code typed into an editor. Python IDE automatically **colour-codes** keywords, built-in function calls, comments, strings and identifier in function header. In addition, **indentation** is automatic.

- **Source Code Editor:**

  Source code editor allows a program to be **written** and **edited** without need to use a separate text editor. Use of an integrated source code editor speeds up **development process**, as editing can be done without changing to a different piece of software each time program needs correcting or adding to.

- **Context-Sensitive Prompts:**

  This feature displays hints (choice of keywords) and available identifiers that might be appropriate at current insertion point of program code. This display predictions of code being entered.

Dynamic Syntax checks:

Dynamic syntax checking finds possible **syntax errors** as program code is being typed into source code editor and alerts programmer at time, before source code is interpreted. Many errors can therefore be found and corrected during program writing and editing before program is run. **Underline** or **highlights** statements that do not meet **rule** of language.

Expanding and Collapsing Code Blocks:

For larger programs that have more than one code block, some code blocks can be collapsed to a single line in the editor allowing the programmer to just see the code blocks that are currently being developed.

Compilers and Interpreters:

Most IDEs provide a compiler and/or an interpreter to run program. Interpreter is used for developing the program and compiler to produce the final version of the object code.

Debugging:

Debugging is a process of **finding** and **correcting errors** (bugs), in program. A debugger is a program that runs program **under development** and aids process of debugging. It allows programmer to:

- **Single Step** through program a line at a time (**single stepping**)
- To set a **breakpoint** to stop execution of program at a **certain point** in source code.

Report window then shows contents of variables and expressions evaluated at that point in program. This allows programmer to see if there are any **logic errors** in program and check that program works as intended.

Auto-documenter

Most IDEs provide an **auto-documenter** to explain **function** and purpose of programming code.

****

ESQ: Identify and describe two features of an editor that can help a programmer to write program code. 9608/42/M/J/19 **[4]**

Feature 1 ....

Description ....

....

Feature 2 ....

Description ....

....

Ans:

- Colouring code//Pretty printing
- This is how the code is presented in the IDE e.g. colour coding and indentation

- Context-sensitive prompts
- Displays keywords or hints at the point of insertion e.g. drop-down list of commands

- Auto-indent
- Automatically indent your code for selection/iteration/procedures/methods

- Auto-complete
- Avoid typing errors // speeds up process of typing

- Expand/collapse subroutines/code
- To make it easier to view code currently working on

****