Man Tik Li
CS 360
Krzysztof Nowak
July 5, 2020

Assignment 1

1. Machine learning is the sequence of bits that directly controls a processor, causing it to add, compare, move data from one place to another, and so forth. Assembly languages are designed with a one-to-one correspondence between mnemonics and machine language instructions. It is translating from mnemonics to machine language known as an assembler.

2. A high-level language is a language that is easier to understand by the programmer. Assembly language is a language that consists of a sequence of bits which directly instruct the processor to perform mathematical operations. A high-level language can include mathematical formulae for numerical computations. It also provides the facility for user interaction. In some circumstances, it still makes sense to program in assembler. When assembly languages used for the writing programs, which are machine-dependent. Assembly languages can provide the facility to control the degree of performance of the machine because of assembly languages were machine level.

3. Programming languages provide a wide range for programmers to interact with the machine. Every programming languages act as an interface between the machine and the user. Each of them has specific syntax and semantics. There are several reasons for the development of so many programming languages:

    - Evolution: computer science is a continuously changing field, tremendous advancements in technology, leading to the innovations for changing requirements.
    - Particular purposes: Each of the programming languages developed for specific purposes only.
    - Personal preference: different people have different levels of reasoning and can adopt different ways of doing the same.

4. Several reasons consider which make a language successful:
    - Expressive Power: Commonly hears arguments that one language is powerful than another.
    - Ease of User for the Novice: Programming language must structure that it could be easily learn and understand by programmer. It should easily understand and keep as simple as possible.
    - Ease of User for the Novice: Programming language must structure that it could be easily learned and understood by the programmer. It should readily understand and keep as simple as possible.

- Standardization: Standardization provides the facility to make a programming language portable.
- Open Source: Programming language must have at least one source compiler or platform available worldwide.
- Excellent Compilers: Compliers help a programmer to manage large projects efficiently.
- Economics, Patronage, and Inertia: The new technology is successful when it has economic feasibility and its technical merits. A language that can meet the requirements of its customer and extensible so that new techniques may be incorporated without much effort is considered successful.

5. Von Neuman language: C, Ada, Fortran
   Functional language: Lisp, Scheme, Fortran
   Object-Oriented language: Smalltalk, C++, Java
   Logic language: Prolog, Spreadsheets
   Widely used concurrent language: C#, Ada

6. Declarative language
   - It takes less effort and easy to write programs in this language
   - Concerned with the structure of the program
   - Simple, it's not concerned with the internal logical details

   Imperative language
   - It is a tedious task for writing programs in this language.
   - Concerned with the complexities of the control flow.
   - Complex, it's focusing on internal logic, which results in a more readable and easily understandable code.

7. All the success language is dependent on the sponsor of that language, making the language available worldwide U.S. Department of Defense is considered backing support for Ada. It made this successful despite its implementation complexities.

8. The first high-level programming language is Fortran, which developed in the mid-1950s.

9. The first functional language was Lisp.

10. Concurrent language divides the available languages as either declarative or imperative. But concurrent languages cannot be classified as such because concurrent languages are implemented using some special libraries or compilers in conjunction with the sequential languages. Therefore, it cannot be clearly distinguished as either declarative or imperative language.

11. The interpretation has complete control of the execution of the programs. It reads the line one by one continues the execution simultaneously. Compilation considered a two-

step process. The compiler transforms the input program into machine language in the first phase. In the next step, the user takes responsibility for executing the machine language program to get the final output. Interpretation provides a better and early debugging facility. The compilation is faster than interpretation. It saves the result in the cache. It keeps the time of interpretation, which scans the program every time a variable is requested. Overall, the compilation is more efficient than interpretation.

12. Java considered compiled and interpreted. Java has use Bytecode to compiled intermediate. Byte code is known as the standard form, which provides the facility of platform independence. Java started using a time compiler, which used to quickly execute programs by translating the bytecode into machine language immediately before every execution.

13. The compiler considered a software program that translates a high-level language into the machine or assembly language or machine language. It consists of five phases: lexical analysis, syntax analysis, intermediate code generation, code optimization, and code generation. The preprocessor scans the program before the compilation begins. The primary focus is to remove comments, delete spaces, and expand macro definitions.

14. The original AT&T C++ compiler, the source code was translated into C code, which again was compiled through the C compiler to produce the final assembly language code. The C compiler doesn't come into action unless the source program translated into C code. The compiler used to analyze the source code, and it also produced the error reports.

15. P-code produced as output in the bootstrapping process of the Pascal compiler.

16. Bootstrapping used in the process of compilation of few languages. It defined as the process of compilation in the same language. This technique used to implement the self-hosting feature of compilers.

17. The just in-time compiler associated with Java language. It developed to provide the faster implementations. These compilers translate the byte code into machine language before every execution.

18. Lisp and Prolog are two languages in which a program can write new pieces of itself "on the fly."

19. TEX and TROFF are text formatter compilers that convert the high-level programs into instructions for the laser printer. Query language processors constitute this type of compilers, which translates the SQL queries into file operations, which used to provide the desired query.

20. Assemblers, debuggers, preprocessors, linkers, style checkers, configuration, and management tools.

21. IDE provides the facility to execute programs in various languages and more user interaction than command-line tools by providing the user interface. Modifying the control structure in IDE's is relatively easy and less time-consuming.

22. Compilation considered to be as a process involving various steps.
    1. Scanner/Lexical Analysis: The function of the scanner is to read the program from left to right and group the elements into legal tokens. The scanner also assigns a number to every single line of the program, making the task of tracing the errors easy.
    2. Parser/Syntax Analysis: This function checks the program's structure without concerning the meaning of the constructs. This phase drives some rules by using some languages.
    3. Semantic Analysis: This phase is concerned with the identification of the meaning of the program. Its function is to maintain the consistency throughout the program by determining that the variables that occur multiple times in a program refer to the same program entity. It consists of two types: static semantic analysis and dynamic semantic analysis.
    4. Intermediate Code Generation: The intermediate code generated in this phase is generally in the form of register operations. After assigning the locations, various arithmetic and other operations of load and store performed.
    5. Code Optimization: intermediate code obtained in the previous phase is optimized to minimize the space and time requirements.
    6. Code Generation: The final code generated for the target program in this phase.

23. First phase: Lexical analysis
    Second phase: Syntax analysis
    Third phase: Semantic analysis
    Fourth phase: Direct execution

24. The program passed to the scanner in the form of a sequence of characters, scanned one by one to group them into a collection of characters known as tokens. These tokens are passed to the parser to generate a parse tree that determines the program's structure at a higher level in terms of expressions, structure, and subroutines.
    Then parser generates the parse tree, which is passed to the semantic analyzer to assign the meaning to the program construct. A semantic analyzer performs static and dynamic semantic checks to determine the programs' static and dynamic errors.
    The semantic analysis phase generates the abstract syntax tree, which used to generate the intermediate code. The intermediate code generated in this phase has some limitations, such as redundant or unnecessary program segments. These limitations eliminated by applying various techniques such as dead-end elimination. This optimized code transferred to the code generation phase.

25. The front end and the back end separated are responsible for language and machine-independent optimizations.
    Front end of the compiler:
    - The front-end handles with language-specific, such as parsing and building up the syntax tree
    - The front end analyzes the source program and produces intermediate code
    - It includes all analysis phases to end the intermediate code generator
    - May shared by compilers for more than one machine
    - The first three phases determine the meaning of the source program, which combined are known as the front end of the compiler

    Back end of the compiler:
    - It focused on low-level machine-dependent optimizations
    - The back end synthesizes the target program from the intermediate code
    - Compilers for more than one source language
    - It includes the code optimization phase and final code generation phase
    - The last three phases used to translate the source program in assembly language or machine language. These phases are collectively known as the back end of the compiler.

26. A phase determines the information, which is useful for the next step. Each phase takes the source program in some form and transforms it into another form, which could be the next phase in the compilations process. Pass refers to a set of inter-related phases in such a manner that the next step cannot start until the completion of the previous phase.
    A compiler having multiple passes is useful in situations where compilers may share the front end for more than one target program, and back end may share for more than one source program.

27. The semantic analyzer maintains the purpose of the compiler's symbol table. The Symbol Table is a data structure. It contains the information about the program's identifiers such as data type, size of the element, and scope of the program's identifier.

28. The static semantics refers to the checks that performed at compile time. The semantic rules which checked at run-time are known as dynamic semantics.

    Static Semantics:
    - Whether each identifier is declared in the program before it used
    - In the switch constructs, all the cases must be associated with a distinct constant
    - Any function which is defined to be having non-void return type must return a be value explicitly
    - The subroutine called with the correct number and type of arguments

- No identifier should call in an illegal context. It should call an integer as a subroutine, performing an addition operation between a string and an integer, referring to a field of incorrect structure

Dynamic Semantics:

- Arithmetic operations must be within the defined limit
- Arrau should be assigned the values within the bounds of the array
- Unassigned variables should not be used in an expression
- If a pointer does not refer a valid object, then it can't be dereferenced

29. Assembly programmers will not be able to generate better code than compilers on modern machines. Because of implementation is not easy with assembly language, as it provides the machine dependent programs. Any register can't be freed until the execution is complete, so concurrent execution of instructions is not possible in assembly language.