The Function and Purpose of Translators

What the Specification Says

Describe the need for, and use of, translators to convert source code to object code;
Understand the relationship between assembly language and machine code;
Describe the use of an assembler in producing machine code;
Describe the difference between interpretation and compilation;
Describe the purpose of intermediate code in a virtual machine;
Describe what happens during lexical analysis;
Describe what happens during syntax analysis, explaining how errors are handled;
Explain the code generation phase and understand the need for optimisation;
Describe the use of library routines

Notes

When computers were first invented, the only way to run programs on them, was to code them in binary. This is what the computer can understand. However it is very time consuming, with lots of repetition, resulting in inefficient programs with limited functionality and often full of errors.

Assembly Languages
A low level languages is a computer language which is very close to what the computer understands, but uses words rather than binary. Each binary instruction is given a word to represent it. Assembly language is a low level language. There are two key features to assembly language, it uses mnemonics and labels.

A mnemonic is a group of letters or keyword representing a particular operation. For example ADD could represent 01101000 which means add this number. Labels work in a similar way, they use a short word to represent the binary address, then store this information in a look-up table so it can be replaced when the program is run. Assembly language is translated by the assembler into machine code.

Assembler
This is the piece of software that translates assembly language into machine code. Machine code is all binary. The assembler must be machine specific, which means that a different assembler is needed for each different make of computer, as the machine code is also specific.

High Level Languages
A high level language is less like what the computer understands, and easier for the programmer. Each instruction gives rise to a series of machine instructions, so it is a one-to-many language. This means it has more functionality and it takes less code to compete each step in a program. Also high level languages are more portable between machines; it is not machine specific.

High level languages are written in source code and then is translated into object code. It contains keywords, which tell the computer what instruction to do and variables which store the addresses of data locations. There are two main methods of translating high level languages.

Interpreter
This is a translator which takes one line of source code, translates it, lets the computer run it, then takes the next line. This is very useful for finding errors, because when the program fails due to something like a logic error, the interpreter knows exactly where the error is. This makes the interpreter very useful for developing code.

Compiler
The compiler is a translator that takes source code and translates it into object code before allowing it to be run. They run more quickly that interpreted programs, as they don't have to be translated as they are being run.

Intermediate Code

Because each language has a different translator and every computer requires different machine code there would need to be a lot of additional software. A way round this would be far more efficient if the compiler or interpreter only translated halfway into intermediate code. A virtual machine will then translate if further into machine code.

Stages of Translation
When a high level language is translated with a compiler there are many stages, each done in parse with each other.

Lexical Analysis
Each of the keywords is looked up in a look up table and replaced with it's binary token. If the keyword is not recognised an error will be returned. It will then get rid of any superfluous characters like additional spaces, lines or tabs which made the code easier for the programmer to read. It will get rid of any comments which the programmer may also have added. Next it will recognise the variables and create a look up table for them called the symbol table containing the values for the variables being used, and the location.

Syntax Analysis
The compiler will use the keyword table to decide what to do with each instruction. It will compare what it gets with what it is expecting. It will return an error if it doesn't get what it's expecting.

Code Generation
The compiler takes each statement which is now just a string of binary, and converts it to low level/ intermediate code. It is a one-to-many process, as each high level instruction is translated into many low level ones. Code optimisation is then done, where the unnecessary instructions are removed.

Library Routines
Library routines are the pieces of code for carrying out a particular process which recurs many times throughout the running of a larger program. They are pre-written, pre-compiled and pre-tested. They are loaded into the memory with a utility program called the loader, and linked to the necessary parts of the code with a utility program called the linker.

Key Points

The use of the Translator

- Machine code is the very simple instructions written as a string of binary digits that the computer can understand
- Programs used to have to be written in machine code, which took a very long time, and made them prone to errors.
- When other languages were developed which were closer to English than machine code, there was a need for them to be translated into a form the computer could understand. This is what translators are for

Machine Code

- A form of language based on binary numbers, and using different combinations of digits to stand for different things.
- The codes are machine-specific, which means that they will only run on the type of machine they were written for.
- Each binary statement can be split in two, the first part represents the operation, and is called opp-code, the second part represents the data, or location of data to which the operation is to be applied to.

Assembly Languages

- Low-level language, because they are close the language used by the computer.
- Uses mnemonics, which are groups of letters or keywords that represent the opp-code part of the instruction.
- The references to the locations are also given alpha-numeric representation to make them easier to use and understand. These are called labels.
- One-to-one relationship with the machine code, meaning one assembly language instruction translates to one machine code instruction.

The Assembler

- The code written in assembly language is now impossible for the computer to understand. In order for it to be of any use, it must be translated to machine code, and the program used to translate assembly language to machine code is called the assembler.
- To convert the mnemonics to their binary tokens, the assembler has a look up table, which it searches, and then makes the replacement accordingly.
- The labels are done in a similar way, although the values are populated as the assembler goes.

High-Level Languages

- Closer to the language spoken by the person writing them, i.e. it's in English not binary
- Each instruction gives rise to a series of machine code instructions, meaning they are one-to-many languages.
- They are also more portable between machines
- There are two ways of translating a high-level language to machine code, using a compiler or an interpreter.

The Compiler

- Converts a program written in a high-level language into machine code.

- The high-level language is called the source code, and the machine code is called object code.
- Uses a lot of computer resources, because it has to be loaded into the memory at the same time as the source code, and have sufficient space to store the intermediate results.
- When an error occurs it is difficult to pin-point where it has occurred
- Converts code all at the same time, as a unit. The first instruction cannot be run until it is all converted

The Interpreter

- Takes one line of the source code translates it, lets the computer run it, then moves on to the next line, and so on through all the code.
- Very useful for finding errors, because it knows what line it got to when it failed. Often used for debugging code.
- This system was developed because early personal computers lacked the power and memory needed for compilation

Intermediate Code in a Virtual Machine

- Different designs of computer have different versions of machine code.
- This would mean that every computer would need a different compiler for each high-level language
- An alternative would be to use a compiler to do most of the translating and end up with a version of the program which is close to all the different machine codes. This is called intermediate code. It is halfway between high-level and machine code.
- It is not machine specific, but can be translated into particular machine code needed.

Stages of Translation

Translation of high-level is a one-to-many process, so it's quite complicated. As a result there are three main stages. Each stage is called a parse. The three stages are lexical analysis, syntax analysis and code generation.

Lexical Analysis

- The lexical analyser uses the source program as input and turns the high level language code into a stream of tokens for the next stage of the compilation
- Tokens are normally groups of 16-bits, and each group of characters in the code is replaced by a token.
- Single characters, which have a meaning in their own right, are replaced by their ASCII values.
- Variable names will need to have extra information stored about them. This is done by creating a symbol table. This table is used throughout compilation to build up information about names used in the program. Only their name is stored in this parse.
- The lexical analyser may output some error messages and diagnostics.
- The lexical analyser also removes redundant that the programmer may have added to make the code more understandable for example spaces, tabs, extra lines and comments
- Often the lexical analysis takes longer than the other stages of compilation. This is because it has to handle the original source code, which can have many formats.

Syntax Analysis

- The code generated in lexical analysis is checked to see if it is grammatically correct.
- Vague error messages can be given if something like a keyword is not recognised
- During syntax analysis certain semantic checks are carried out. These include label checks, flow of control checks and declaration checks.
- The syntax analyser verifies all variables and updates the symbol table with necessary information like type, size and scope.

Code Generation

- All the errors should have been removed by now, and the source code is just a string of binary digits that the compiler can understand.
- The addresses of the variables are calculated and stored in the symbol table.
- The intermediate code is then produced.
- When ready the compiler can produce machine code from this intermediate code by looking each binary token up in a look-up table.

Library Routines

- Many short pieces of code for carrying out a particular process recur many times in larger programs
- It would be a waste to go through rewriting and compiling them each time
- Library routines can be called whenever task is necessary to be done
- Pre-written, pre-compiled and pre-tested.
- Loaded into the memory by a utility program called the loader
- Linked to the relevant places in the existing code by a utility routine called the linker

High Level Languages

High level languages are computer languages that are closer to English and further away from what the computer understands.

They are one-to-many languages, as one high level instruction gives rise to a series of machine code instructions.

Source code is the code written by the programmer in the high level language. It is then translated, either with a compiler or interpreter into object code – that the computer can understand.

Linkers and Loaders

The **Loader** is a small program that loads the jobs and adjusts the addresses into necessary places. It helps the OS with memory management

The **Linker** is the program that links all the different memory locations and parts of the program together.

Compiler

- Takes the source code and translates it into object code
- The compiler, the source code and the object code must all be stored in the memory, there must also be additional memory for intermediate results
- When an error occurs it is hard to pin point where it is in the source code

Interpreter

- Each instruction is taken in turn and translated into machine code, it is then executed before the next instruction is translated.
- It was developed because early personal computers lacked the memory and resources needed for compilation.
- It can produce error messages as soon as the error is encountered. This makes it very useful for developers.
- Slower compared to a compiled program because the original program has to be translated every time it is executed. Instructions inside a loop have to be translated each time the loop is entered.

Assembly Language and Machine Code

Machine Code

- The language that the computer can understand, that uses different binary digits to stand for different instructions and locations.

Assembly Language

- a basic language which uses mnemonics to stand for instructions and labels to represent data locations. Assembly language has a one-to-one language with machine code. It is translated by the assembler.

Assembler

- the piece of software that translates assembly language to machine code. It converts the mnemonics to their binary representation through finding them in the look-up table. It builds a look-up table for the labels as data locations as it translates the program.

Stages of Translation

Lexical Analysis

Uses the source code as input and replaces the key words with their binary tokens from the look-up table.

A symbol table is created for the variables.

It will output error messages if it finds an error. For example if a keywork is not in the look up table, or a variable is undefined.

It will remove all additional lines, spaces tabs and comments.

Syntax Analysis

The code is checked to see if it is gramatically correct.

Further errors may be picked up, these won't be as accurate as the error logging in the lexical stage.

Semantic checks are carried out, including label checks, flow of control checks and Declaration checks.

Must ensure that certain control constructs are used in the right places.

Code Generation

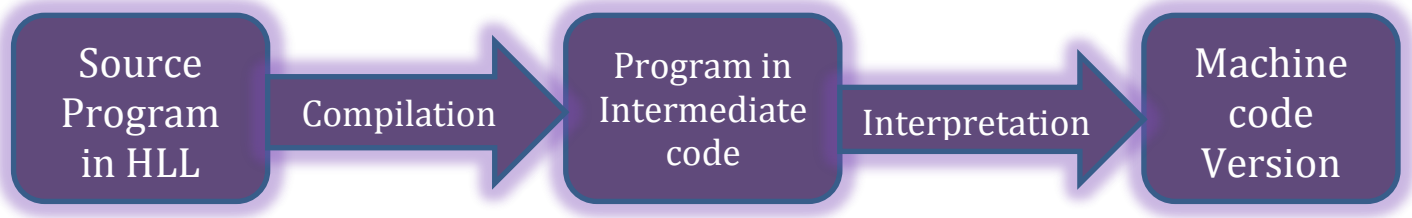
By now, all errors have been removed or reported and the code is all binary.

Complier takes each statement and translates it into low level/ intermediate code. One-to-many process

Optimisation is when the compiler gets rid of any lines which are not strictly necissary, this makes the program shorter so it takes up less memory and funs faster.

INTERMEDIATE CODE IN A VIRTUAL MACHINE

Different designs of computer have different versions of machine code, so different instructions mean different things. This would mean every computer would need a different compiler for each high level language.
Alternatively a compiler could be used to do most the translation, and end up with a program which is close to all the machine codes. This is the intermediate code.
Intermediate code is halfway between the high level language and machine code. It is not machine specific but it is now translated into the particular machine code needed by an interpreter specific to that machine.
This means the program written in a high level language will be able to run on different machines, and is portable.



Key Words

- **Translator** – piece of software that converts one form of code to another form more understandable by the computer. Three type, assembler, interpreter and compiler
- **Assembly Language** – basic low-level language with a one-to-one relationship with machine code, developed in the late 1940's. Uses mnemonics and labels.
- **Mnemonics** - keywords or groups of letters representing basic operations.
- **Labels -** are alpha-numeric representations of data locations.
- **Assembler** – piece of software that translates assembly language to machine code.
- **Machine Code** – the binary code that the computer can understand. Machine-specific, meaning that different computers need different machine code.
- **High-Level Language** – languages closer to English. One-to-many language, meaning each high-level instruction gives rise to a series of machine code instructions. More portable between machines.
- **Source Code** – the high-level code written by a programmer
- **Object Code** – after the source code has been translated, it becomes object code.
- **Keyword** – special word used in high-level languages that is associated with a statement that has its own syntax.
- **Interpreter** – translator program that translates one line of code at a time. Especially useful for debugging and testing as can return accurate error message. Was developed because it uses less computer resources than compiling, but slower.
- **Compiler** – translator program that translates the whole program as a unit. Quicker, but requires a lot of memory, and error diagnosis are vague.
- **Intermediate Code** – half translated language, that is not machine-specific but can be translated the rest of the way.
- **Virtual Machine** – this is the piece of software required to run intermediate code.
- **Parse** – a look through, or stage of translating a program.
- **Lexical Analysis** – the first stage of translation, where each keyword is replaced with its binary token, that's been found in the look-up table. Variables are added to the symbol table, and all superfluous characters are removed.
- **Syntax Analysis** – uses the keyword table to decide what the instructions for that particular keyword is and what rules to apply.
- **Code Generation** – the final stage of translation, where the code is actually generated/ converted to machine code.
- **Optimisation** – this is done during code generation, just removes the unnecessary parts.
- **Library Routines** – pre-written, pre-tested and pre-compiled sub-routines
- **Loader** – utility program that loads library routines into the memory
- **Linker** – utility program that links library routines to the relevant places

Past Exam Questions and Answers

These are questions that have appeared in past papers relating to the function and purpose of translators, and the mark scheme answers.

Describe assembly language

a language related closely to the computer being programmed/low level language/machine specific
uses descriptive names (for data stores)
uses mnemonics (for instructions)
uses labels to allow selection
each instruction is generally translated into one machine code instruction
may use macros

Describe machine code

binary notation
set of all instructions available
to the architecture/which depend on the hardware design of the processor
instructions operate on bytes of data

What tasks are performed by the assembler when producing machine code?

reserves storage for instructions and data
replaces mnemonic opcodes by machine codes
replaces symbolic addresses by numeric addresses
creates symbol table to match labels to addresses
checks syntax/offers diagnostics for errors

What are the features of the interpreter?

translates one line/statement…
…hen allows it to be run before translation of next line
reports one error at a time…
…nd stops

What are the features of a compiler?

translates the whole program as a unit
creates an executable program/intermediate program
may report a number of errors at the same time
optimisation

Describe lexical analysis

source program is used as the input
tokens are created from individual symbols and from…
…he reserved words in the program
a token is a fixed length string of binary digits
variable names are loaded into a look-up table / symbol table
redundant characters (eg spaces) are removed
comments are removed
error diagnostics are given
prepares code for syntax analysis

What is the purpose of a translator?

convert from source code…
…to object code
detect errors in source code

Why may intermediate code may be more useful than executable code?

can run on a variety of computers
same intermediate code can be obtained from
different high level languages
improves portability

What additional software is needed to run intermediate code?

interpreter / virtual machine

What is a disadvantage of using intermediate code?

the program runs more slowly/has to be translated each time it is run / need additional software

What does code optimisation do?

makes code as efficient as possible increases processing speed reduces number of instructions

Describe syntax analysis

accepts output from lexical analysis statements/arithmetic expressions/tokens are checked... …against the rules of the language/valid example given eg matching brackets errors are reported as a list (at the end of compilation) diagnostics may be given (if no errors) code is passed to code generation further detail is added to the symbol table… …eg data type /scope/address

What's intermediate code, and it's use?

simplified code / partly translated code… …which can be run on any computer/virtual machine/improves portability… …using an interpreter sections of program can be written in different languages runs more slowly than executable code

What software converts source code into object code?

translator

What is source code?

the original code/code written by the programmer…
…often in a high level language
may be in assembly language
source code can be understood by people…
…but cannot be executed (until translated)

Why might library routines help programmers, and when are they used

Library routines:
routines are pieces of software…
…which perform common tasks…
…such as sorting/searching
routines are compiled
Why library routines help programmers:
routines are error-free/have already been tested
already available/ready to use/saves work/saves time
routines may be used multiple times
routines may have been written in a different source language
allows programmer to use others' expertise
How routines are used:
linker is used…
…to link routine with program
loader handles addresses…
…when program is to be run

Further Resources

Resources on the VRS (http://vrs.as93.net)

- This hand out
- The presentation that goes with it
- More revision posters
- More past exam questions
- User contributed documents

Quizzes (http://revisionquizzes.com)

- High-Level Languages
- Low-Level Languages
- Library Routines
- Stages of Translation
- Translators summary