Code Generation

In computer science, code generation is the process by which a compiler's code generator converts some internal representation of source code into a form(e.g., machine code)that can be readily executed by a machine.

Issues in the Design of a Code Generator:-

1. **Input to the Code Generator :**The input to the code generator consists of the intermediate representation of the source program(Optimized IR),together with information in ST that is used to determine the Run Time Addresses of the data objects denoted by the names in IR. Finally, the code generation phase can therefore proceed on the assumption that its input is free of the errors.
2. **Target Programs :** The output of the code generator is the target program. The output code must be **Correct** and of **high Quality**, meaning that it should make effective use of the resources of the target machine. Like the IR ,this output may take on a variety of forms:
    a. **Absolute Machine Language //** Producing this form as output has the advantage that it can placed in a fixed location in memory and immediately executed. A small program can be compiled and executed quickly.
    b. **Relocatable Machine Language //** This form of the output allows subprograms to be compiled separately. A set of relocatable object modules can be linked together and loaded for execution by linking-loader.
3. **Memory Management :** Mapping names in the source program to addresses of data objects in run time memory. This process is done cooperatively by the Front-end & code generator.
4. **Major tasks in code generation :** In addition to the basic conversion from IR into a linear sequence of machine instructions, a typical code generator tries to optimize the generated code in some way. The generator may try to use

faster instructions, use fewer instructions ,exploit available registers ,and avoid redundant computations. Tasks which are typically part of a compiler's code generation phase include:

i. Instruction selection: Is a compiler optimization that transforms an internal representation of program into the final compiled code(either Binary or Assembly).The quality of the generated code is determined by its Speed & Size. For example, the three address code (x=y+z) can be translated into:

MOV y,R0
ADD z,R0
MOV R0,x

If three-address code is :

a=b+c
d=a+e

then the target code is :

MOV b,R0
ADD c,R0
MOV R0,a
MOV a,R0
ADD e,R0
MOV R0,d

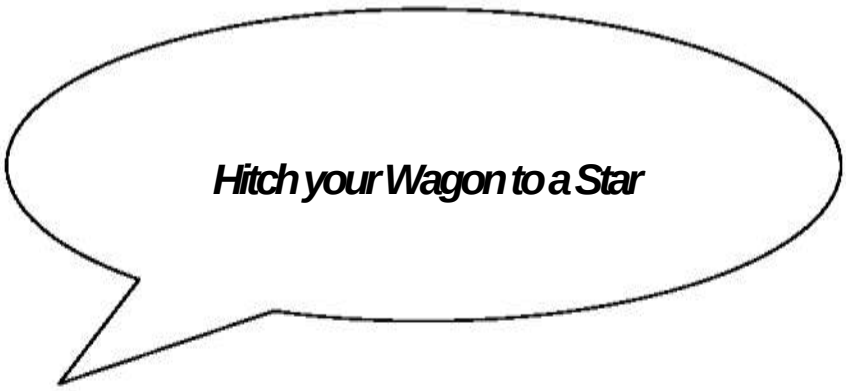
Finally, A target machine with "**Rich**" instruction set may be provide several ways of implementing a given operation. For example, if the target machine has an "increment" instruction (**INC**) ,then the IR a=a+1 may be implemented by the single instruction (INC a) rather than by a more obvious sequence :

MOV a,R0
ADD #1,R0
MOV R0,a

ii. Instruction Scheduling : In which order to put those instructions. Scheduling is a *speed optimization*. The order in which computations are performed can

effect the efficiency of the target code, because some computation orders require fewer registers to hold intermediate results than others.

iii. Register Allocation : Is the process of multiplexing a large number of target program variables onto a small number of CPU registers. The goal is to keep as many operands as possible in registers to maximize the execution speed of software programs (*instructions involving register operands are usually shorter and faster than those involving operands in memory*).



References

1. A.Aho,R.Sethi,J.D.Ullman," **Compilers- Principles, Techniques and Tools**"Addison-Weseley,2007

2. J.Tremblay,P.G.Sorenson,"**The Theory and Practice of Compiler Writing** ",McGRAW-HILL,1985

3. W.M.Waite,L.R.Carter,"**An Introduction to Compiler Construction**",Harper Collins,New york,1993

4. A.W.Appel,"**Modern Compiler Implementation in ML**" ,CambridgeUniversity Press,1998

5. Internet Papers