Introduction to Compilation Techniques

Sudakshina Dutta

IIT Goa

25^{th} January, 2022

What is a Compiler ?

100000	00100	00010	00011	00000000000
opcode	rc	ra	rb	unused

- $Reg[4] \leftarrow Reg[2] + Reg[3]$
- We'd rather write $Add\ R2, R3, R4$ (Assembly code)
- Or write $a = b + c$

What is a Compiler ?

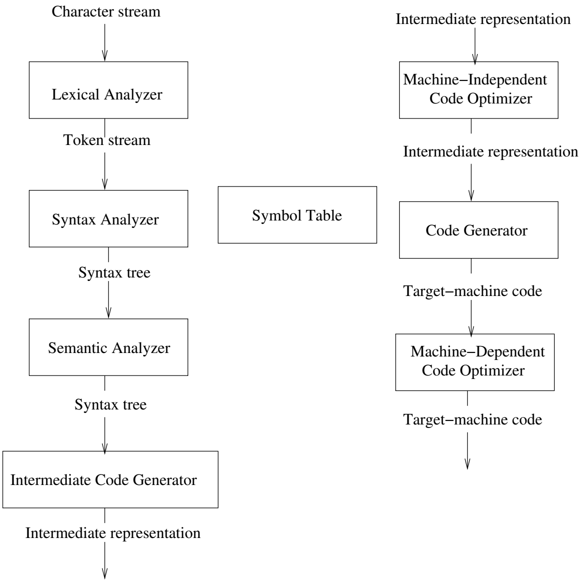
Advantage over assembly

- Productivity : concise, readable, maintainable
- Portability : run the same program on different hardware

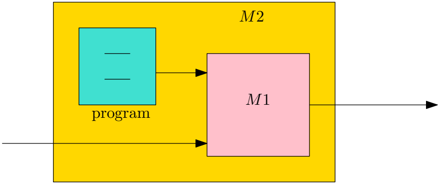
Disadvantage over assembly

- Efficiency

The Phases of a Compiler



Interpreter



- $M1$ with the interpreter can be thought of as the implementation of $M2$
- The interpreter emulates the behavior of $M2$
  - If we run a SciPy application, it might generate many python instructions which may generate many x86 instructions

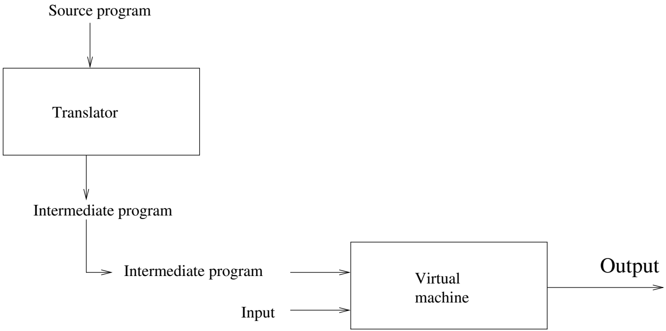
Interpreter

- A language processor
- Instead of generating target program, it directly executes the program on supplied inputs
- Machine language target program generated by compiler is much faster
- No optimization phase
- Advantage - Better error diagnostics, less space
- Disadvantage - Slower
- Example - Perl, Python, Ruby, etc

Other language processor

- Java language processor combines compilation and interpretation
- Java source program is first compiled into bytecodes (optimized and compressed)
- Bytecodes are then interpreted by a virtual machine
- The benefit is portability — bytecodes of one machine can be interpreted in another machine
- Just-in-time (JIT) compiler translates the bytecodes into machine language code to make the execution faster

JIT Compiler



- A technique in which the intermediate representation is compiled to native machine code at runtime
- Advantage : efficiency

Parameter Passing Mechanisms

Parameter Passing Mechanisms

- **Actual Parameter :** The parameter used in the call of a procedure
- **Formal Parameter :** The parameters that are used in procedure definition

Call-by-Value

- Two steps
  - The actual parameter is evaluated (if an expression)
  - It is placed in the location of the formal parameters (if a variable)
- It has the effect that all computations involving formal parameters done by the called procedure is local to that procedure and the actual parameters cannot be changed

```
int add(int a, int b)
{
  return a + b;
}

int triple(int x)
{
    return add(x + 5, 5);

}
```

Call-by-Value

Exception

- If an array name is passed by values, then the address gets copied and the change is visible in the corresponding address
- The caller copies the entire actual parameter into the space belonging to the corresponding formal parameter
- Expensive operation for large objects
- C, C++ (commonly), Java use Call-by-Value mechanism

Call-by-reference

- Used for “ref” parameter in C++
- The effect of call-by-reference is achieved by passing address for C language
- Found in C++

```
void Init(int &x, int &y){

    for(k = 0; k < 10; k++){
            y = k;
            x++;
    }

}
main(){
    j = 0;
    Init(j, A[j]);
}
```

0	1	2	3	4	5	6	7	8	9
9									

Call-by-name

- The callee executes if the actual parameter were substituted literally for the formal parameter
- It is as if the formal parameter were a macro standing for the actual parameter with the relevant renaming done
- Used in early language e.g., in Algol 60.

```
void Init(int x, int y){

    for(k = 0; k < 10; k++){
            y = k;
            x++;
    }

}
main(){
    j = 0;
    Init(j, A[j]);
}
```

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

Some programming language basics

- **Scope** : Scope of a declaration x is the region of the program in which uses of x refer to the declaration
  1. **Static scope** : Scope can be determined in compile time
     - Usually defined by block structures ('{' and '}') or begin-end
  2. **Dynamic scope** : Otherwise

Static scoping

$main()$ {

$int\ a = 1;$ $\quad B_1$

$int\ b = 1;$

{

$int\ b = 2;$ $\quad B_2$

{

$int\ a = 3;$ $\quad B_3$

$cout << a << b;$

}

{

$int\ b = 4;$ $\quad B_4$

$cout << a << b;$

}

$cout << a << b;$

}

$cout << a << b;$

}

- If declaration D of name x belongs to block B, then the scope of D is all of B, except for any blocks B' nested to any depth within B, in which x is redeclared

Dynamic scoping

$$\#define\ a\ (x+1)$$

$$int\ x = 2;$$

$$void\ b()\ \{int\ x = 1; print(a);\}$$

$$void\ c()\ \{print(a);\}$$

$$void\ main()\ \{b(); c();\}$$

- Outputs are 2 and 3
- We examine the function calls which are currently active and take the most recently called function that has a declaration of x
- Dynamic scope rule resolutions are essential for polymorphic procedures