Compilers

CS 346

3 – 0 – 0 – 6

Monday – Tuesday – Wednesday

9.00 --- 9.55 10.00 --- 10.55 11.00 --- 11.55

Instructors: Dr. A. Sur and Dr. A. Sarkar

Arijit Sur

Office: H-103

Department of Computer Science and Engineering

Email: arijit@iitg.ernet.in

Ph: 2361

Syllabus

- Overview of different phases of a compiler: front-end; back-end;
- Lexical analysis: specification of tokens, recognition of tokens, input buffering, automatic tools;
- Syntax analysis: context free grammars, top down and bottom up parsing techniques, construction of efficient parsers, syntax-directed translation, automatic tools;
- Semantic analysis: declaration processing, type checking, symbol tables, error recovery;
- Intermediate code generation: run-time environments, translation of language constructs;
- Code generation: flow-graphs, register allocation, code-generation algorithms;
- Introduction to code optimization techniques.

Books

- Text Book
  - Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Compilers: Principles, Techniques, and Tools, 2nd Edition, Prentice Hall, 2009
- Reference Books
  - V. Raghavan, Principles of Compiler Design, McGrawHill, 2010
  - C.N. Fischer, R.J. Le Blanc, Crafting a Compiler with C, Pearson Education, 2009
  - K. D. Cooper, L. Torczon, Engineering a Compiler, Morgan Kaufmann Publishers, 2004
  - Allen Holub, Compiler Design in C, Prentice-Hall software series, 1990

Grades

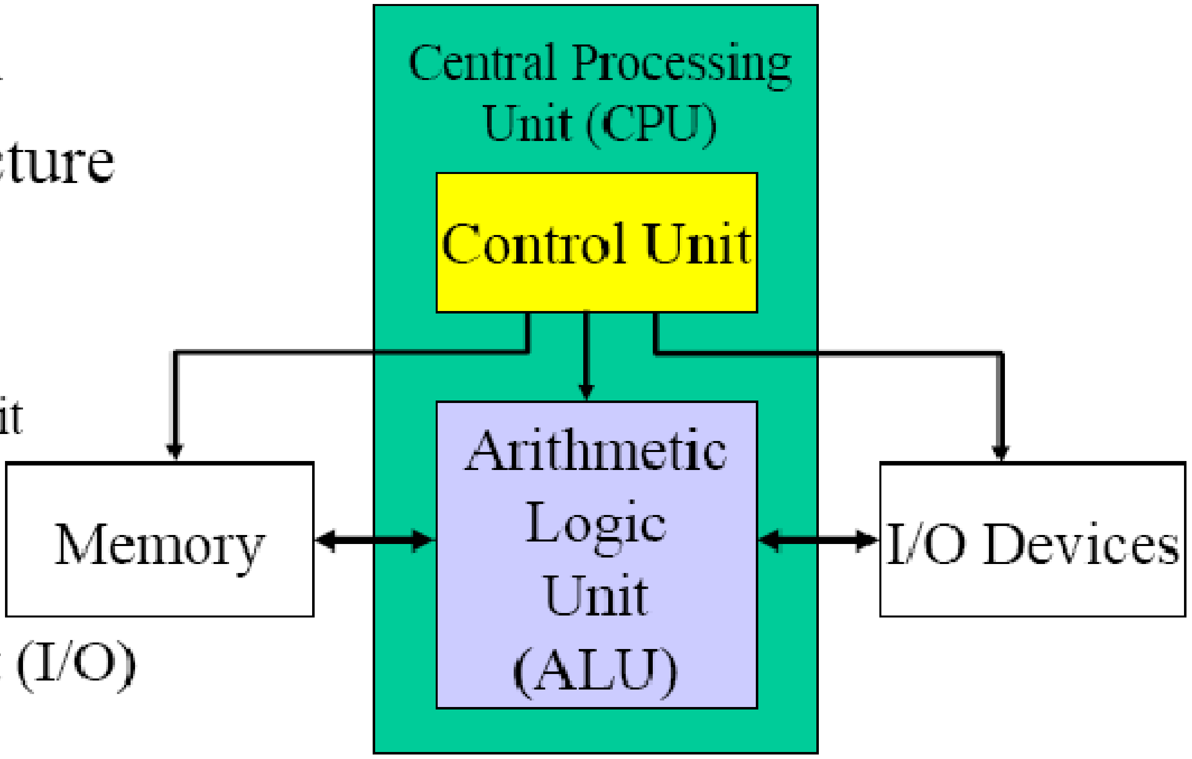
- 2 Quiz s
  - 20% of grade
- Mid-Semester
  - 30% of grade
- End-Semester
  - 50% of grade
- Attendance below 75% --- F Grade
- Random Attendance

Why Compilers

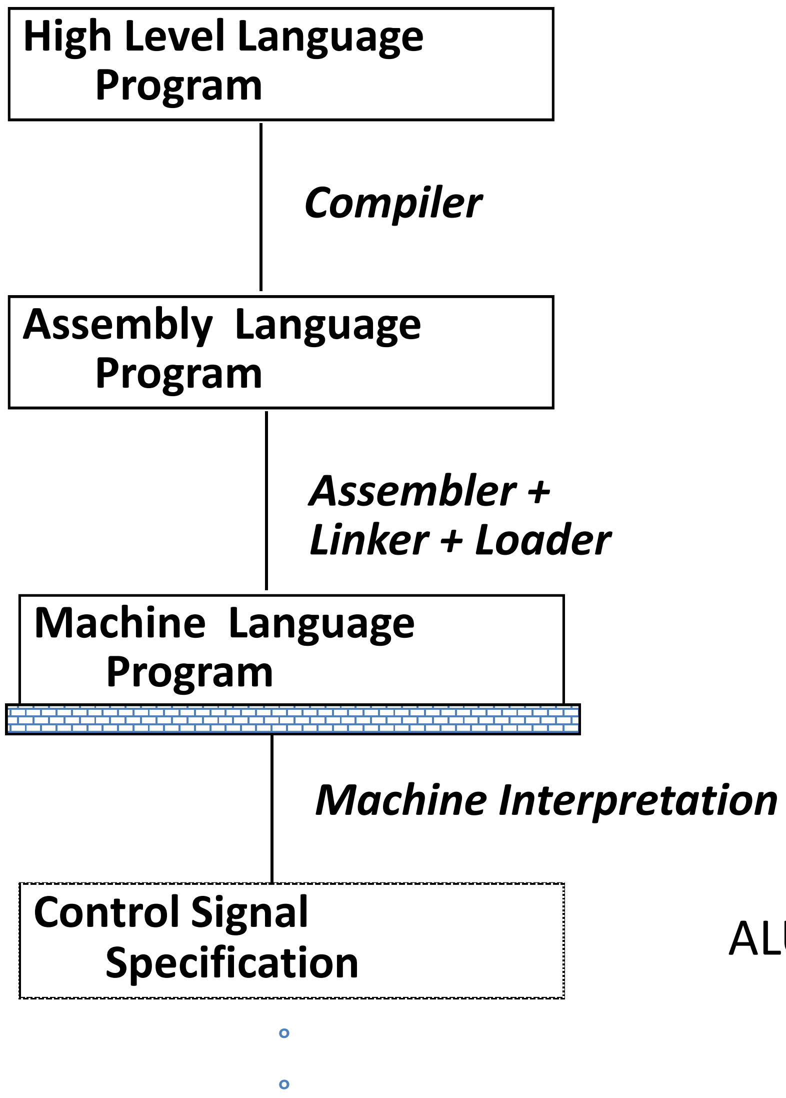
How Program is Executed through a Computer?

Basic Blocks of Computing System

- Good example of a digital system
- Basic architecture consists of:
  - CPU
    - Control Unit
    - ALU
  - Memory
  - Input/Output (I/O) Devices



Program Execution: Layered Architecture


High Level Language Program
Compiler
Assembly Language Program
Assembler + Linker + Loader
Machine Language Program
Machine Interpretation
Control Signal Specification

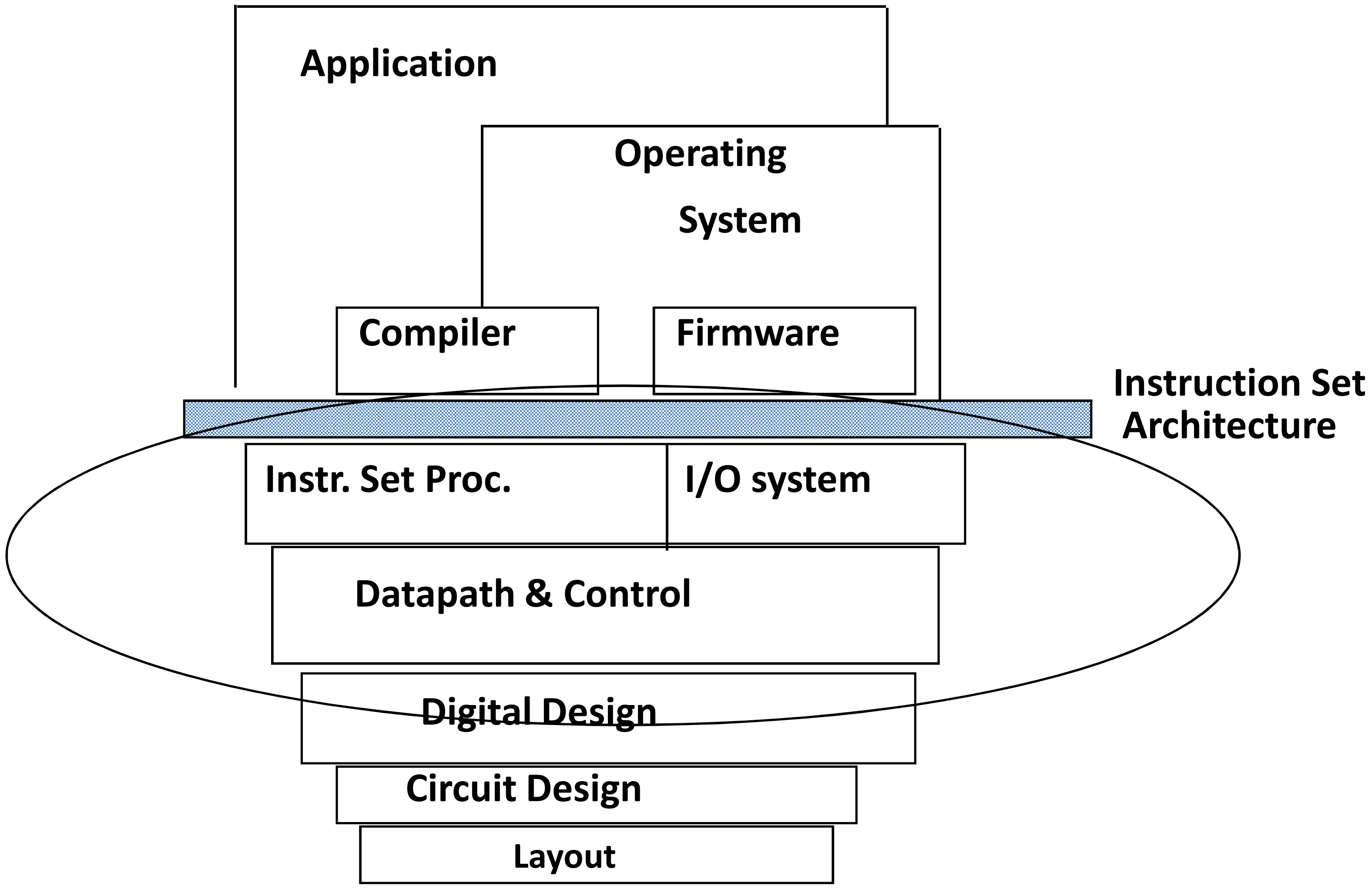
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw  $15,    0($2)
lw  $16,    4($2)
sw  $16,    0($2)
sw  $15,    4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

ALUOP[0:3] <= InstReg[9:11] & MASK

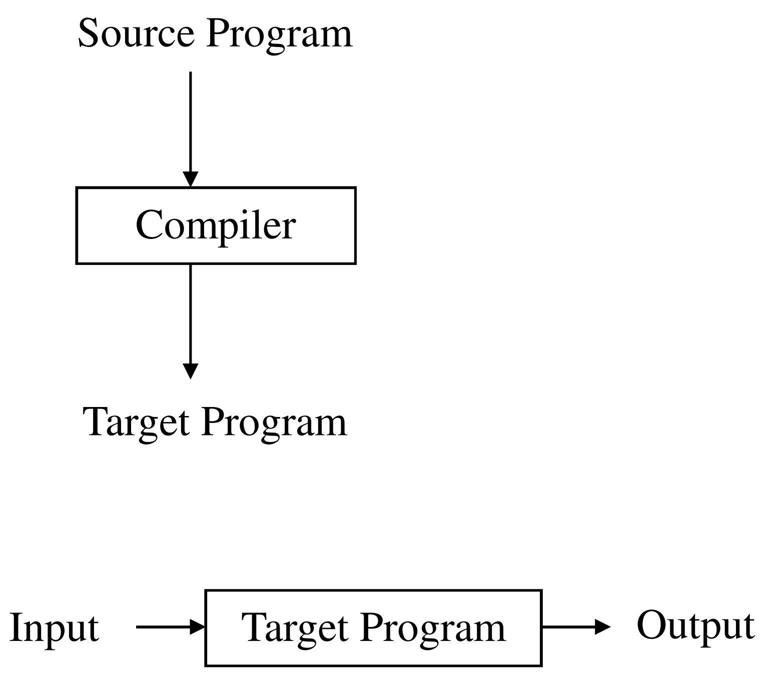
Architecture of Computing System

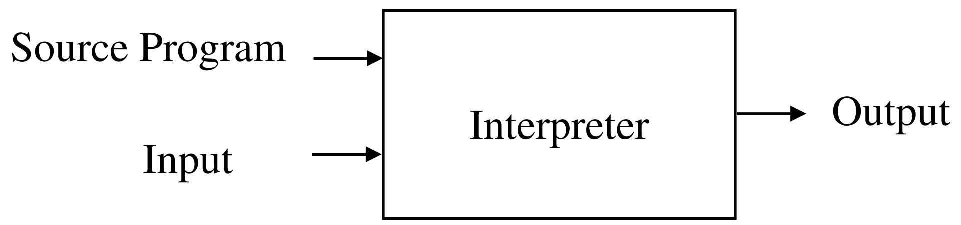


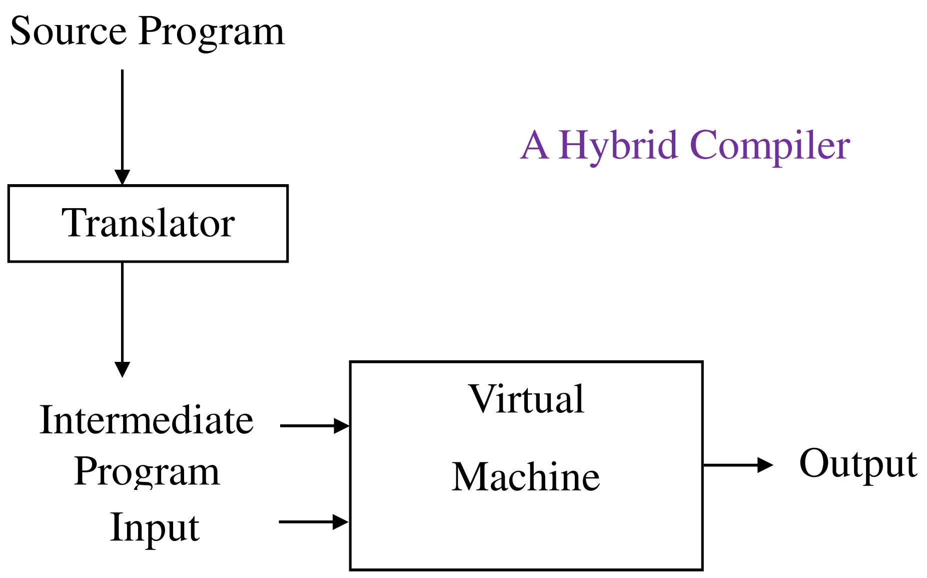
Definitions

- What is a compiler?
  - A program that accepts as input a program text in a certain language and produces as output a program text in another language, while preserving the meaning of that text (Grune *et al*, 2000).
  - A program that reads a program written in one language (source language) and translates it into an equivalent program in another language (target language) (Aho *et al*)
  - Report errors of source language which are detected during the translation process
- What is an interpreter?
  - A program that reads a source program and produces the results of executing this source.
- *We deal with compilers! Many of these issues arise with interpreters!*

Compiler Vs. Interpreter







A Hybrid Compiler

Examples

- C is typically compiled
- Lisp is typically interpreted
- Java is compiled to bytecodes, which are then interpreted

- source-to-source compiler, transcompiler, or transpiler :
  - C++ to C
  - High Performance Fortran (HPF) to Fortran (parallelising compiler)

- Cross Compilers (wiki)
  - A **cross compiler** is a compiler capable of reading executable code for a platform other than the one on which the compiler is running. For example, a compiler that runs on a Windows 7 PC but generates code that runs on Android Smartphone is a cross compiler.

Qualities of a Good Compiler

What qualities would you want in a compiler?

- generates correct code (first and foremost!)
- generates fast code
- conforms to the specifications of the input language
- copes with essentially arbitrary input size, variables, etc.
- compilation time (linearly)proportional to size of source
- good diagnostics
- consistent optimisations
- works well with the debugger

Principles of Compilation

The compiler must:

- *preserve the meaning of the program being compiled*.
- *“improve” the source code in some way*.

Other issues (depending on the setting):

- Speed (of compiled code)
- Space (size of compiled code)
- Feedback (information provided to the user)
- Debugging (transformations obscure the relationship source code vs target)
- Compilation time efficiency (fast or slow compiler?)

Applications

- Most common use: translate a high-level program to object code
  - Program Translation: binary translation, hardware synthesis, …
- Optimizations for computer architectures:
  - Improve program performance, take into account hardware parallelism, etc…
- Interpreters: e.g., BASIC, Lisp etc.
- Software productivity tools
  - Debugging aids: e.g., purify
- Text formatters, just-in-time compilation for Java, power management, global distributed computing, …

Key: Ability to extract properties of a source program (analysis) and transform it to construct a target program (synthesis)

Thanks