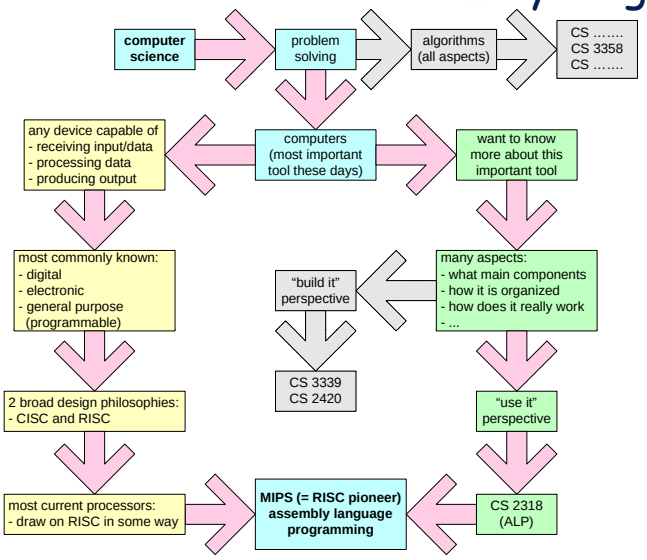
Introduction
(computer science and assembly language)



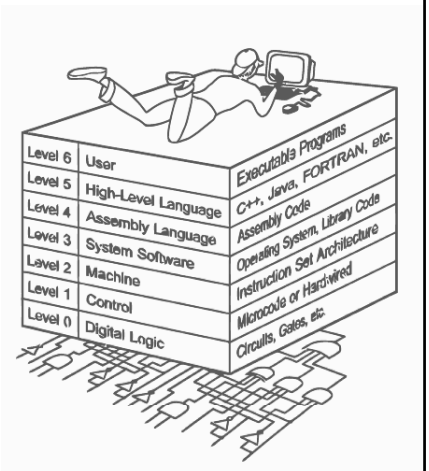
Introduction
(computer level hierarchy)

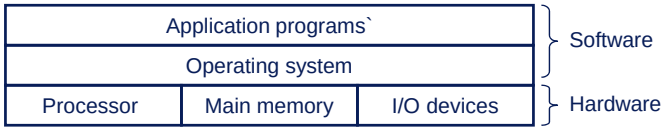
digital, electronic, general-purpose

- **Typical computer: has many things besides chips**
  - Case in point: needs ***software*** (***programs***) to do anything useful
- **Building complex software (programs): no easy task**
  - Crucial: use ***abstraction*** to help *deal with (manage) complexities*
    - Divide-and-conquer: solve overall problem by solving smaller problems
- **Building typical computer: adopt similar approach**
  - Crucial: use ***abstraction*** to help *deal with (bridge) semantic gap*
    - Divide into series of ***virtual machine*** layers

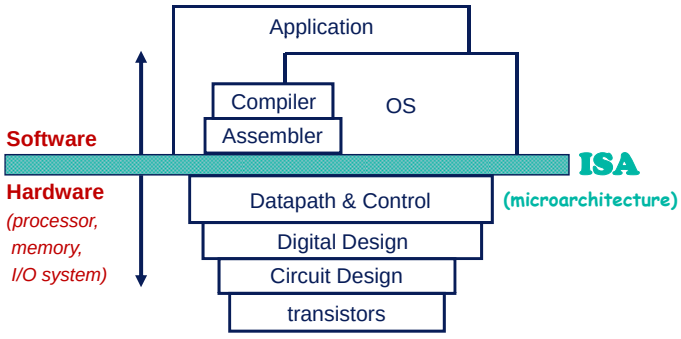
Introduction
(computer level hierarchy)

- **Each virtual machine layer: *abstraction* of level below it**
- **Machine at each level: execute its own particular instructions**
  - Calling machines at lower levels to perform tasks as required
- **Computer circuits ultimately carry out the work**





Introduction
(Hardware-Software Interface)

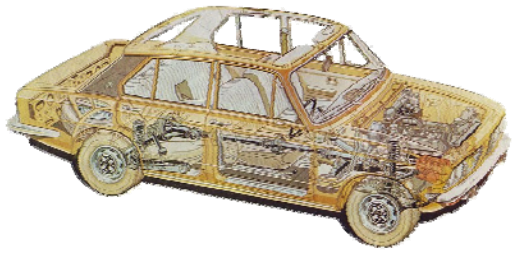


Instruction Set Architecture (ISA): Describes, without implementation details, the interface that HW provides to SW.

NOTE: **ISA** is to *machine design* (in *Computer Architecture*) as **ADT** is to *data type design* (in *Data Structures*). Both ISA and ADT (Abstract Data Type) are concepts resulting from the use of *abstraction*.)

Introduction
(Architecture vs Microarchitecture)

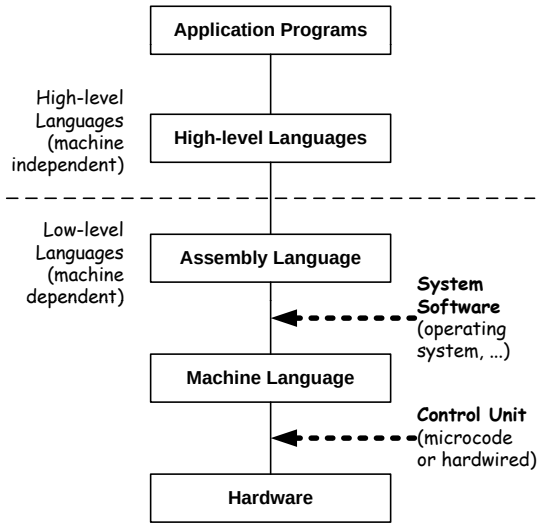
Architecture (ISA)
Interface hardware presents to software



Microarchitecture
Detailed implementation of architecture

Introduction
(what is assembly language)

- **Programming the Computer**
  - *More than one* way
    - 1GL
    - 2GL
    - 3GL
    - 4GL
    - . . .
- **Programming it in Assembly Language**
  - *Lower-level* way (*closer-to-hardware* way)
    - 2GL



Introduction
(what is assembly language)

high-level, assembly, and machine languages

high-level language program ← C++ (much of CS 1428/2308) belongs here

```
void swap(int v[], int k)
{
   int temp;
   temp = v[k];
   v[k] = v[k + 1];
   v[k + 1] = temp;
}
```

compiler

```
swap:
   muli $2, $5, 4
   add  $2, $4, $2
   lw   $15, 0($2)
   lw   $16, 4($2)
   sw   $16, 0($2)
   sw   $15, 4($2)
   jr   $31
```

assembler

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

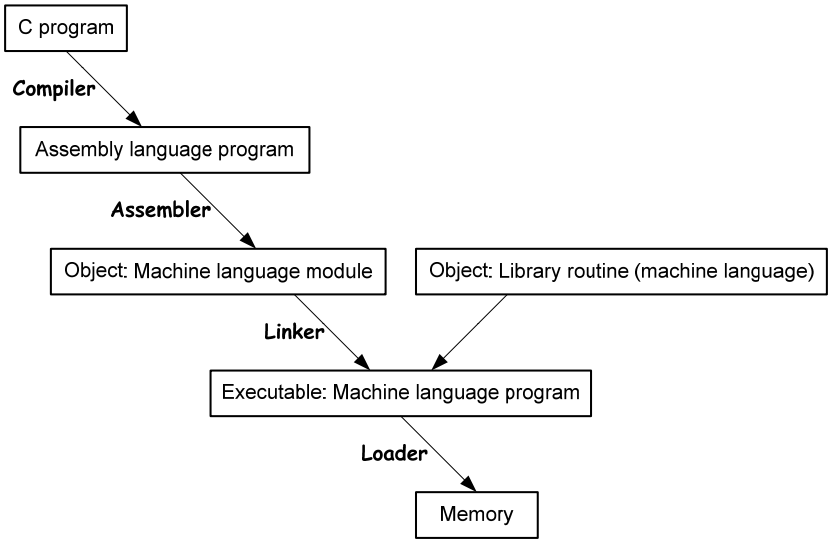
assembly language program ← CS 2318 — binary machine language program

High-level language program compiled into assembly language program and then into binary machine language program.

NOTE: Although the translation from high-level language to binary machine language is shown here as two steps, some compilers produce binary machine language directly.

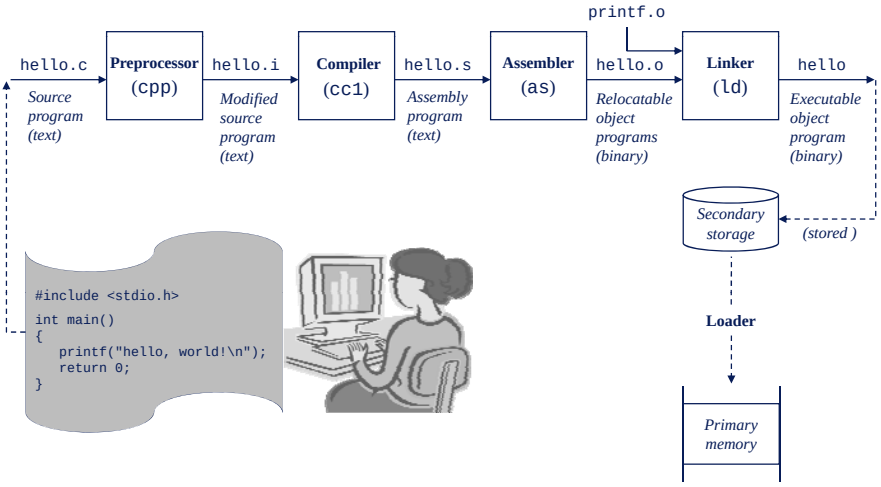
Introduction
(what is assembly language)

a translation hierarchy for C



Introduction
(what is assembly language)

a translation hierarchy for C



Introduction
(what is assembly language)

compared to high-level language

- **High-Level Language (e.g.: C)**
  - Architecture of processor chip is not visible
    - *One* statement usually corresponds to *many* basic operations of processor

```
...
...
...
y = (x + 4) * 3;
...
...
...
```

- **Assembly Language**
  - Architecture of processor chip is clearly visible
    - *One* (“true”) instruction corresponds to *one* basic operation of processor

```
...
lw      $t0, 0($a1)
addi    $t1, $t0, 4
addi    $t2, $0, 3
mult    $t1, $t2
mflo    $t3
sw      $t3, 0($a0)
...
```

(assumes addresses of `x` & `y` are in `$a1` & `$a0`)

Introduction
(what is assembly language)

in reality

- **In practice:**
  - Rare to just use “true" assembly language
  - Assembler provides various "extras"
    - For programmer convenience
  - But processor chip is still visible
- **Particularly for this course...**
  - MARS (a MIPS emulator)
    - Provides *pseudoinstructions* (or *macro instructions* or *synthetic instructions*), *directives*, *etc.*
  - SPIM (another MIPS emulator)
    - Similarly

Introduction
(what's good about assembly language)

what ***some*** would say

- Speed
  - Fast
- Space
  - Compact
- Unique capability
  - OS, BIOS, hardware levels
- Knowledge
  - How computer really works

Embedded systems programs
Device drivers
Critical section optimization
Systems programs (OS and compilers)
Games, games, games
. . .

How processor works
Basic computer architecture
Representation of data/instructions
Insight into hardware concepts
. . .

Introduction
(what's bad about assembly language)

what ***many*** would say

- Assembly language is hard to learn/write
- Assembly language is hard to debug/maintain
- Assembly language is hard to read/understand
- Assembly language programming is time consuming
- Improved compiler technology has eliminated the need for assembly language
- Today, machines are so fast that we no longer need to use assembly language
- If you need more speed, you should use a better algorithm rather than switch to assembly language
- Machines have so much memory today, saving space using assembly language is not important

All (and always) true?
Some (and sometimes) true?
Background/situation dependent?

- Assembly language is *not portable* ⇐ undeniably true!

Introduction
(where assembly language stands)

not so encouraging reality according to some

- Few programmers write more than a tiny fraction (< 1%) of code in assembly language
- Few expect > 5% of students will end up working in environments where assembly language is the primary programming language

The BIG Picture Assembly language is a programming language. Its principal difference from high-level languages such as BASIC, Java, and C is that assembly language provides only a **few, simple types of data and control flow**. Assembly language programs **do not specify the type of value held in a variable**. Instead, a programmer must apply the appropriate operations (e.g., integer or floating-point addition) to a value. In addition, in assembly language, programs **must implement all control flow with *go to*s**. Both factors make assembly language programming for any machine — MIPS or 80x86 — more difficult and error-prone than writing in a high-level language.

(Source: *Computer Organization and Design - The Hardware/Software Interface* by David Patterson & John Hennessy, 3ed, Appendix A, p. A-12.)

Introduction
(readings)

- http://instructors.coursesmart.com/0131328980/1
  - Pages 1 through 13 (until end of Section 1.1)
- http://pages.cs.wisc.edu/~larus/HP_AppA.pdf
  - Pages A-3 through A-10 (until end of Section A.1)
- http://onlamp.com/lpt/a/4804
  - "Why Learning Assembly Language Is Still a Good Idea"