High Level Language (HLL) vs Assembly Language

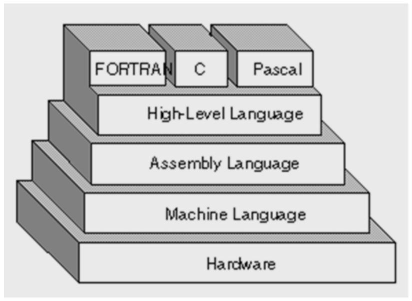
J T Wunderlich PhD

Assembly Language is:

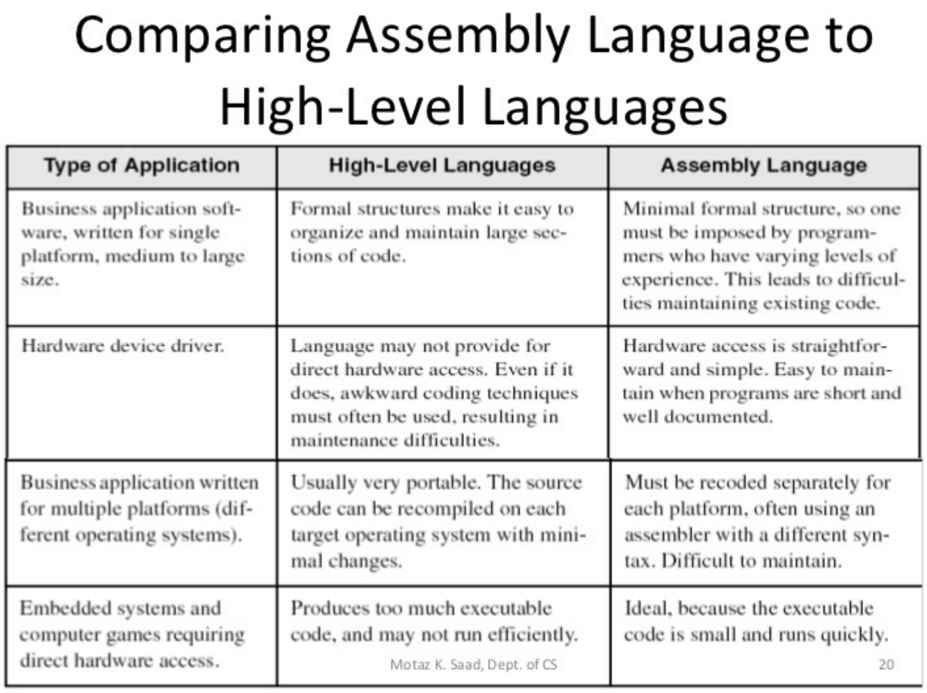
1. Faster
2. Uses less memory
3. Better control over hardware (since it is directly translated one to one into machine code)

But is:

1. Machine dependent (i.e., written for a specific Microprocessor or Microcontroller)
2. More difficult to program
3. Difficult to maintain



https://cdn.educba.com/academy/wp-content/uploads/2015/11/11.png

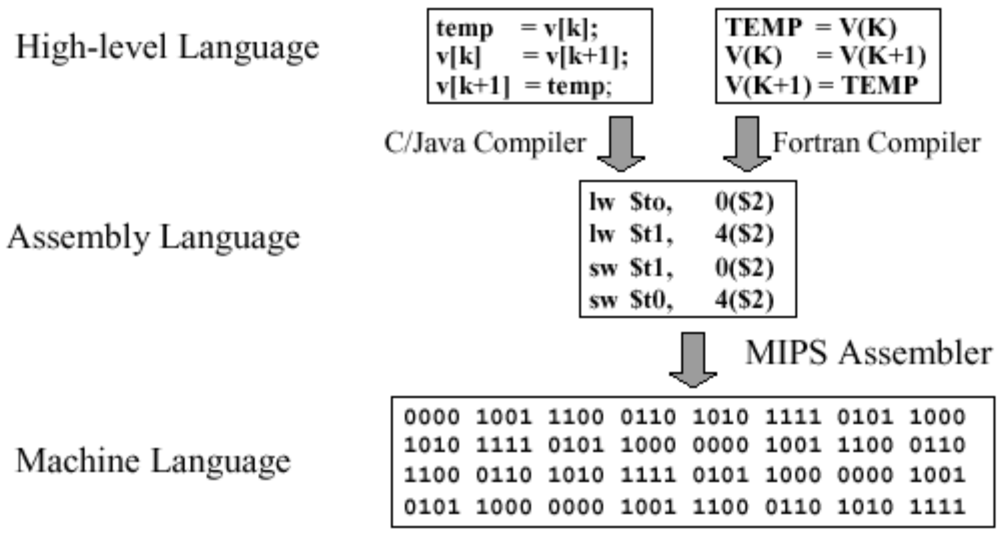


Comparing Assembly Language to High-Level Languages

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

Motaz K. Saad, Dept. of CS 20

https://image.slidesharecdn.com/assemblylanguageintro-1220345279381385-9/95/introduction-to-assembly-language-20-728.jpg?cb=1220320740



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

```
TEMP   = V(K)
V(K)   = V(K+1)
V(K+1) = TEMP
```

```
lw  $to,  0($2)
lw  $t1,  4($2)
sw  $t1,  0($2)
sw  $t0,  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
```

https://www.cise.ufl.edu/~mssz/CompOrg/Figure2.1-CompLang.gif