

# Assembly Languages

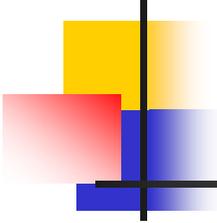
---

Alexandre David

1.2.05

[adavid@cs.aau.dk](mailto:adavid@cs.aau.dk)

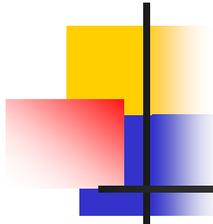




# High vs. Low Level Languages

---

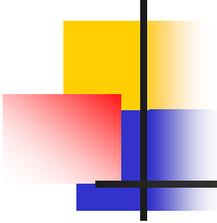
- One-to-many translation.
- Hardware independence.
- Application oriented.
- General purpose.
- Powerful abstractions.
- One-to-one translation.
- Hardware dependence.
- System oriented (OS).
- Special purpose.
- Few abstractions, no complex data-structures.



# Assembly Languages

---

- Low level – tight to processors.
  - One assembly language per processor (family).
  - The structure is always the same, the devil is in the details.
- Different from
  - Java, C, C++, etc...
  - These are defined once.
- Sometimes several assembly languages for the same processor.
  - Mentioned in the book: Intel & Bell Lab.
  - Exercise: Intel manuals & gnu tools.



# Interesting Patterns

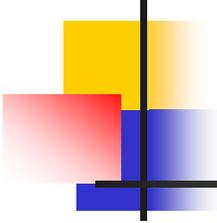
---

- Conditional execution
  - code that changes conditional flags
  - conditional jump

- ex:

```
test $20, %%eax  
je equal_20
```

```
cmp %%eax, %%ebx  
ja greater
```



# Interesting Patterns

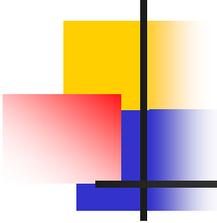
---

- For-loops [ for(init; invariant; next) ]

- similar with while-loops

- ex:

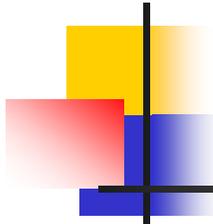
```
# Assume EAX contains the end.
    xor %%ecx, %%ecx      # for i = 0
    cmp %%ecx, %%eax     # if i ≥ n
    jae end              # then end
loop:
    ...
    inc %%ecx            # i++
    cmp %%ecx, %%eax     # if i < n
    jb loop              # then loop
end:
```



# Interesting Patterns

---

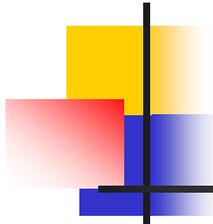
- Functions:
  - (save stack base pointer  
load registers)  
execute  
(save result - register or stack)  
(restore stack)  
ret
- Function call:
  - load registers  
call func\_label



# Storing Constants

---

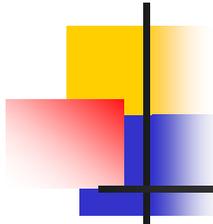
- Simple data declarations with labels.
  - Size of the data only, no type.
  - Ex: .long, .word...



# Interaction With Assembly

---

- Specific purposes (special hardware access).
- Specific optimizations
  - simd/multimedia/special arithmetics...
- Ex: `asm` or `__asm__` directives.
- Possible to write a `.s` file, assemble it, and link the object file with the rest of the program.



# Compiler vs. Assembler

---

- A compiler transforms the original program into assembly.
  - Freedom, optimizations.
  - Maintain semantics.
- An Assembler makes a one-to-one translation.
  - From mnemonics (opcode shortcut) to opcode (binary representation).
  - Computes offsets for jumps.