LiveOverflow BinExp Notes

0x04

8-32 fixed sized global variables called registers, depending on architecture

- 32bit architecture = 32 bits of information, 2,147,483,647 biggest number
- 64bit arch = 64 bits of information, 18446744073709551615 biggest number

Special registers

- Program counter / PC / IP / EIP / Instruction pointer - which instructions to execute next, every time an instruction is ran, it gets incremented.
  - If the value of the IP is 5, we would say the register "points to line 5"
- Stack Pointer / ESP - points to the top of the stack (a region of memory). The stack is at the top of RAM and as it is pushed to, **it grows downwards** in addresses

Most operations on registers, like subtraction and addition, have the side effect of altering status flags. The status flags live in a special register. Eg if the operation is `sub eax, ebx`, both registers having the value 8. then the `zeroflag` will be set to 1, since the output is 0.

If you run out of registers, you use memory, ram.

- Memory is accessed either with loads and stores at addresses, as if it were a big array. Or through PUSH and POP operations on a stack
- loads and stores are when (in 32 bit assembly) you `mov <register>, [address]`, eg `mov eax, [14]`

Control flow is done via GOTOs — jumps, branches or calls. These change the program counter directly

- A jump is an uncondtitional GOTO. Eg `jmp 5` is the same as `mov eip, 5`
- a jump if equal to zero, `je`, checks the zero flag
- a call is an unconditional GOTO that pushes the next address on the stack, and when the function you called is finished, it will hit a `RET` statement, which will pop the address value off the stack into the PC, which continues the program.

If you optimise your code, and only use cache and registers, it will be signifinantly faster than using ram or non volatile memory