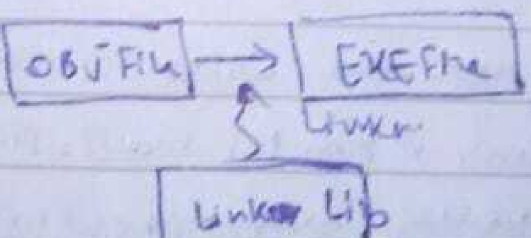
Nashit Budhwani 20K-0274

1) High level language (HLL) compilers help translate statements from from their language into assembly language and machine code (1's and 0's). Usually a program written in a specific language (such as ~~[illegible]~~ Java) will be translated into respective byte code by the compiler

(low level code)

Job of OS is to load the executable file (.EXE) into memory and branches the CPU to the programs starting address (may initialize some registers as instruction pointer) and then begins executing program (program starts running)



The process of linking is done by linker which reads object file and checks if there are any calls to PROCS from Link Lib.

This is basically done to copy any required procedure from Link library, combine them with OBJ file to create exe file (which goes to OS loader)

2.) REALADDRESS

In real address mode, segment registry are used to hold base address for program code, data and stack.

L> code segment - holds address of all executable in program

L> data segment - holds add for variables. It stores data for program.

L> extra segment - extra data segment (used for shared data usually)

L> stack segment - holds address for stack (stores interrupt and subroutine data).

Protected MOde :

Segment registers are called Selectors when operating system in protected mode. There selectors point to segment descriptors( inorder to access physical memory

(1) 32-bit general Registers

L> EAX (Extended accumulator register)
L> EDX (" data " )
L> ECX ("
L> EDX (" Data ")
L> EBP (" Base pointer)

↳ESP (〃 stack 〃)

↳ESI (〃 source index)

↳EDI (〃 destination index)

↳EDI (〃 destination indx)

↳EIP (〃 instruction pointer)

16 bit Segment Registers:

↳CS

↳SS

↳DS

↳ES

↳FS

↳GS.

(iii) 8 bit:

↳AH, AL

↳BH, BL

↳CH, CL

↳DH, DL.

1 4
8

3(a) Segment : ABOG

Offset : 5D89

~~ABOE~~
~~+ 5D89~~

$$\begin{array}{r} \text{AB0E0} \\ +\ \text{5D89} \\ \hline \text{B0E69} \end{array}$$ 4

(b) Segment = 8FE3

Offset : ?

Real Add : A835

$$\begin{array}{r} \text{A835f} \\ -\ \text{8FE30} \\ \hline \text{1852f} \end{array}$$ ← Offset

(c) Segment :

Offset : 5E6D

Real Add : FF41D.

$$\begin{array}{r} \text{FF41D} \\ +\ \text{5E6D} \\ \hline \text{F95B0} \end{array}$$ ← Segment

6

(d

(d) Real Address: A5B6D.

offset : 440ED → (assumption)

RA = (S* 10)+off

A5B6D - 440ED = S(segment)

segment = 61A8

* Basically when Java code is run (in a machine with any OS) the HLL code is trasformed into Java byte code (.class extension). This byte-code is platform independent (key feature of Java potertabity). Now but for this byte code or. class to be executed we uses a JVM ( Java virtual machine) which resides in RAM of OS.

JVM's job is to see which platform is on rom and conver the bytecode to machine code c this is platform independent (diff machine code fo MAC .OS windows 32, Linux)

| REAL ADDRESS | Virtual -8086. |
|---|---|
| -> only 1MB of memory can be addresed (0 to FFFFF) | -> Each program can address a max 4GB memory. |
| -> programs can access any part of memory. | -> Programs can't access any other memory other than their own. |
| -> runs MS-DOS | -> Allocates 1MB of memory to run MS-DOS |
| -> programs in real-address can cause OS to crash, because | -> Even if MSDOS crashes, it will not affect other programs running at same tim |
| -> program runs directly on hardware, it has to unlimited memory. | -> In virtual, protected mode runs in a background and decides what memory has access some might be virtual and others simulated by system |

Date

6) Control flags:

↳determines how instructions are carried out

↳Enable or disable certain ~~options~~ operations.

↳ They include:

↳ Direction flag (affects direction of block data transfers)

↳ Interrupt flag (determines when interruptions can occur)

↳ Trap flag (determines wether CPU is halted after every instruction).

Status flags:

↳ reflect the outcomes of arethmetic and logic operations performed by CPU.

↳ enable an instruction based on result of previous instruction.

↳ They include:

↳ Carry (when there is a carry digit after arithmetic op)

↳ Overflow (same as carry but for signed arithmetic)

↳ sign (1 - negative, 0 - positive).

↳ Zero (when arethmetic result indicates includes all zeros, 1-yes; 0-no)

↳ Auxillary Carry (when carry occurs from 3rd bit)

↳ parity (even number of bits or odd).