L'ordinateur exécute des programmes encodés en binaire.

Un mot de données est constitué de bits.
Un processeur 64 bits gère des mots de 64 bits avec pour premier rang 2^0 et comme dernier rang 2^{63}.

On trouve des composants électroniques qui travaillent en 12 bits ou 14 bits, pour de la conversion analogique-numérique par exemple.

Pour les instructions processeur, il me faut une table de correspondance entre un nombre codé en binaire et une opération.
C'est ce qu'on appelle l'ISA : Instruction Set Architecture, présente au niveau du décodeur du processeur.

Programme écrit en langage de haut niveau (comme C) compilé ou assemblé en langage assembleur qui permet de transmettre les instructions au processeur.
Plusieurs architectures processeurs : x86, AMD64, ARM, IA64, PPC PIC, etc. Pour les microcontrôleurs : PIC, AVR, etc.
Chaque type de processeur a ses propres instructions.

Micro-code : table de conversion intermédiaire pour comprendre des instructions appartenant à d'autres architectures.
Un émulateur ou un hyperviseur permet d'assurer ce type de compatibilité.

Les menaces ne sont pas les mêmes d'une architecture à l'autre.
L'architecture la plus répandue est l'architecture ARM.
Microcontrôleur = processeur + entrés + sorties.

L'OS est un binaire exécuté par le processeur. C'est un code qui a été compilé.
Sa fonction est de gérer kes fonctionnalités de base du système :
> I/O et interface HardWwre/Software
> Ordonnancement
> Gestion de la mémoire (pour l'exécution de programmes par exemple)
> Lancement d'autres binaires
Le processeur ne peut gérer des données que si elles sont sockées dans la mémoire !

Architecture en couches :
> Application Software
> Operating System
> Hardware
> CPU / Controller
La couche applicative ne voit pas directement la mémoire, elle doit passer par le système d'exploitation.

Quelques OS : Linux, BSD, zOS, IBM i (OS400), Windows, DOS, Android, iOS, Solaris, Mac OS, BeOS, TOP Family, etc.
Linux ne descend pas de UNICS ! OpenBSD, oui, MacOS aussi. Mais les OS sont géénralement développer pour rester sur des standards partagés par tous.

Les OS sont souvent organisés en couche mais selon plusieurs structures : monolithic, layered ou en microkernel.
Notion de UserSpace et de KernelSpace !

Le jeu d'instruction fait le lien entre le software et le hardware (C > Binaire par exemple).
Pour ça, on utilise des compilateurs : on compile pour un OS et pour une cible (architecture CPU).
3 étapes :
> Front-end
> Middle-end
> Back-end

Le front-end est correlé à un langage de programmation donnée (C, C++, Fortran, ADA).
Il traduit des séquences d'instructions en une représentation intermédiaire (IR, Gimple) commune à tous les langages.
Il y a une analyse lexicale (scanner), sémantique (parser), contextuelle (checker) qui

aboutit a une Réprésentation Intermédiare.
La représentation intermédiaire consiste à convertir une séquence de tokens en un langage "universel" simplifié.
On utilise :
> des basic-blocks simples, un entrée et deux sorties max
> Opérateurs à deux entrées
> opérations ariothmétiques binaires (2 opérandes, 1 destination)
> des sauts conditionnels
> Variables en assignement unique (Static Single Assignement SSA)

Le middle-end permet d'optimiser le graph de la représentation intermédiare.
C'est une étape indépendante du langage et de la cible.
Toutefois, certaines spécificités de la cible peuvent être introduites à ce stade : opérations vectorielles, exhibition du parallélisme.
Les optimisations courantes : propagation de constantes, déroulage de boucle, supression des branches mortes.

Registres : cases mémoire des opérateurs dans les CPU

Le CPU réalise des opération simples : deux entrées, une sortie.
Il les fait sur des données stockées dans une file de registres (32 positions max) qu ne peut gérer que peu de données.
On y adjoint un cache qui va permettre de stocker les données à traiter, les résultats obtenus et à sauvegarder le contexte (opérations réalisées pour différentes applications).
La mémoire RAM vient ensuite assister cette espace cache !
Les fréquences de fonctionnement des ces différents composants sont cependant différents et se réduisent ! C'est ce qu'on appelle le Memory Wall !

Les allocations mémoires :
Les processeurs modernes disposent d'une Memory Management Unit :
> Elle gére matériellement les caches
> Elle gère matériellement les pages mémoire
> La distribution des données sur les différentes banques mémoire et leur accès
> La protection de la mémoire (contexte)
L'utilisateur/compilateur ne travaille donc qu'avec des adresses logiques, qui n'ont aucune réalite physique.
Ensuite, la MMU convertit ces adresses en adresses physiques, afin de pointer aux bons endroits (cache, banque mémoire, etc.)

Il y a dans le mémoire une zone partagée qui permet de stocker du code utile à toutes les applications (librairies partagées .dll ou .so).

La génération consiste à convertir l'arbre d'instructions et les différents basic-blocks générés indépendamment en un exécutable complet.
Généralement, l'étape de link est la dernière étape : elle liste les appels de fonctions externes pour demander leurs adresses à la prochaine exécution.

Bash, Perl, Ruby, PHP, Python, Java sont des langages interprétés, pas compilés, ils ne génèrent pas de binaire !
Un interpréteur shell est un binaire exécutable qui va être chargé en mémoire RAM et qui va gérer du script (fichier texte) pour appeler des commandes précompilées qui elles sont binaires !

NB : ls -alh : affiche aussi la taille des éléments sur le disque dur

Python, Perl, PHP diffère dans la mesure où les commandes utilisées sont préchargées dans la mémoire RAM.
Pour ces langages interprétés, toute la partie front-end est exécutée au fur et à mesure.
Parfois des optimisations de middle-end sont exécutées.
L'interpréteur déroule le code au fur et à mesure (avec un peu d'avance sur l'exécution afin de permettre des optimisations).
Il suffit de faire correspondre une focntion à chaque opération à exécuter et l'appeler au fur et à mesure (Perl, Ruby).
Certains produisent un Bytecode - une représentation intermédiaire optimisé - comme PHP, JAVA, TCL). L'environnement virtuel Java lit le Bytecode !
Bytecode :

Dans le cas de Java, toutes les étapes de compilation du front-end au back-end sont respectées, sauf que le jeu d'instruction ciblé est un jeu d'instruction "JAVA".
Le binaire produit est du Bytecode JAVA qui comprend des formats, des mnemonic... presque comme un ISA !
L'octet est l'unité de base en Bytecode (c'est le bit pour les langages machine.
Python : gestion du Front-end
Java : gestion du Front-end et du Middle-end