

Structured computer organization

by Andrew Tanenbaum · Language: FR

Summary by booksummaryai.com —

<https://booksummaryai.com/country/fr/structured-computer-organization-resume>

Summary

En 60 secondes

Tanenbaum's argument is that a computer is not one machine but a stack of levels, each built on the one below: digital logic, microarchitecture, instruction set, operating system, assembly language. He starts with Pascal's brass gears and ENIAC's 18,000 vacuum tubes, then walks you up the stack. You build flip-flops and memory chips, microprogram a Java virtual machine, decode x86 instructions, and page memory like an operating system. By the top, the machine you feared is one you could have assembled yourself.

Pourquoi le lire

This is the book that turns the black box into a building you can walk through. Programmers who only ever touch high-level languages finally see what a cache, a page fault, or a stack push physically costs. Students of computer science get the standard text for architecture courses, written by the man who built MINIX to teach exactly this. Read it if you want to stop memorizing jargon and start seeing the machinery underneath.

Les idées clés

L'ordinateur est une pile de machines virtuelles

Tanenbaum ouvre sur un choix auquel IBM a été confronté au début des années 1960 : chaque nouvelle machine impliquait un nouveau code, et les clients se noyaient dans des jeux d'instructions incompatibles. La réponse fut le System/360, annoncé comme une famille où un seul jeu d'instructions couvrirait des machines aux prix et aux performances très différents. Cette décision scinda le domaine en deux : l'architecture, les attributs visibles par le programmeur, et l'organisation, les unités opérationnelles et les interconnexions qui réalisent concrètement la spécification. De là, le livre déploie sa colonne vertébrale, une hiérarchie de niveaux allant de la logique numérique jusqu'à la microarchitecture, le jeu d'instructions, la machine du système d'exploitation et le langage d'assemblage, couronnée par les langages de haut niveau.

Une fois que vous acceptez cette vue en couches, chaque partie déroutante de l'informatique trouve sa place. Un défaut de page appartient au niveau du système d'exploitation ; un échec de cache appartient à la microarchitecture située en dessous. L'intuition du System/360, c'est que l'interface peut survivre à n'importe quelle implémentation particulière, et c'est pourquoi des décennies de puces exécutent toujours le même code. Fait surprenant, les couches ne sont pas

qu'un artifice pédagogique ; elles sont la manière dont l'industrie a réellement géré le changement sans briser tout ce qui se trouvait au-dessus d'elles.

À essayer: Choisissez n'importe quel programme que vous avez écrit cette semaine et nommez quel niveau de la hiérarchie gère chacun de ses moments de lenteur : une lecture disque, une mauvaise prédiction de branchement, une pause de ramasse-miettes.

La Génération Zéro tournait à l'engrenage, pas au silicium

Avant les transistors, il y avait le laiton. La machine à additionner de Blaise Pascal (la Pascaline, construite en 1642 pour son père collecteur d'impôts) vint d'abord, et un siècle plus tard suivit la conception de la machine analytique de Charles Babbage avec son moulin et son magasin. Ada Lovelace, créditée comme la première programmeuse, vit ce qu'une telle machine pouvait accomplir. Puis en 1946 l'ère électronique arriva à l'Université de Pennsylvanie : l'ENIAC (1946), construit à l'Université de Pennsylvanie par John Mauchly et J. Presper Eckert, fut le premier ordinateur électronique à usage général. Il pesait 30 tonnes, contenait 18 000 tubes à vide et consommait 140 000 watts. Tanenbaum range tout cela sous la Génération Zéro dans ses jalons de l'évolution de l'ordinateur.

Le saut des engrenages de Pascal aux tubes de l'ENIAC n'est pas seulement une arithmétique plus rapide ; c'est l'arrivée de l'électronique comme support du calcul. Ce qui surprend les lecteurs, c'est l'ancienneté des idées : les programmes enregistrés, la programmabilité, voire le rôle du programmeur furent esquissés un siècle avant que le matériel puisse suivre. Le poids de 30 tonnes et l'appétit de 140 000 watts expliquent pourquoi la génération suivante, celle des transistors, compta tant. La miniaturisation n'était pas de la vanité ; c'était une question de survie.

À essayer: Esquissez côte à côte le moulin et le magasin de la machine analytique avec un processeur et une mémoire vive modernes, et étiquetez quelle partie joue quel rôle.

Les transistors ont transformé la taille d'une pièce en taille d'un bureau

La première génération fonctionnait aux tubes à vide, à la manière de l'ENIAC, et ils grillaient constamment. La deuxième génération les échangea contre des transistors, et les machines changèrent de caractère : l'IBM 1401 en 1959 était une machine populaire mais non scientifique, et l'IBM 7094 en 1962 était un typique grand ordinateur scientifique de deuxième génération. Puis les circuits intégrés arrivèrent pour la troisième génération, avec l'IBM System/360 comme famille emblématique de troisième génération, et les générations suivantes allèrent plus loin avec le LSI et le VLSI. Sous tout cela court la loi de Moore, créditée à Gordon Moore (cofondateur d'Intel) : le nombre de transistors sur une puce double tous les 18 à 24 mois.

Les générations ne sont pas des étiquettes marketing ; chacune a changé ce qu'un ordinateur pouvait être. La distinction entre le 1401 et le 7094 montre que même au sein d'une même génération de transistors, les machines commerciales et scientifiques étaient des produits différents. La loi de Moore est le moteur de toute cette histoire, une exponentielle qui a

transformé la lignée de l'ENIAC grand comme une salle en puces comptant des milliards de commutateurs. La surprise tient à la prévisibilité du doublement, et à la part de l'architecture moderne qui n'est qu'une réponse à celui-ci.

À essayer: Cherchez le nombre de transistors de la puce de votre téléphone et comptez combien de doublements le séparent d'un processeur du début des années 1970.

Un programme enregistré a tout changé

L'ouvrage cite John von Neumann et son rapport de 1945 sur l'EDVAC, qui introduisit le concept de programme enregistré, et l'architecture de von Neumann classique en découle. La conception de l'IAS comporte cinq parties essentielles : la mémoire, l'unité arithmétique et logique (UAL), l'unité de commande, et les équipements d'entrée, la mémoire de l'IAS stockant 4096 mots de 40 bits chacun. Le CPU vit au rythme de son cycle fetch-decode-execute, et les bus d'adresses, de données et de commande acheminent le trafic entre le CPU, la mémoire et les périphériques, avec des contrôleurs de périphériques comme un contrôleur de disque interposés entre eux. Le même chapitre parcourt les pistes et secteurs d'un plateau de disque dur, définissant le temps de positionnement (seek time), la latence rotationnelle et le temps de transfert, puis empile le cache face à la RAM et au disque dans la hiérarchie mémoire.

L'idée du programme enregistré est d'une simplicité trompeuse : les instructions résident dans la même mémoire que les données, si bien qu'un programme peut être chargé, modifié et traité comme une information. Tout ce qui a suivi, des compilateurs aux systèmes d'exploitation, dépend de ce geste unique. La hiérarchie mémoire explique un fait que tout programmeur finit par ressentir : la mémoire rapide est petite et coûteuse, si bien que les systèmes parient sur la localité, en gardant près de vous un peu de ce dont vous avez besoin et le reste au loin. Le temps de positionnement et la latence rotationnelle expliquent pourquoi les lectures aléatoires sur disque sont pénalisantes.

À essayer: Chronométrez une boucle qui lit un gros fichier de façon séquentielle par opposition à des sauts aléatoires dans le fichier, et observez apparaître les coûts de positionnement et de latence.

L'algèbre de Boole rencontre les circuits à commutation en 1937

George Boole avait inventé une algèbre de la logique, mais elle resta inutilisée pour les circuits jusqu'à ce que Claude Shannon, dans son mémoire de master de 1937, l'applique aux circuits à commutation et fonde la conception de circuits numériques telle que nous la connaissons. De ce mariage naissent les opérations fondamentales : AND, OR, NOT, NAND, NOR, XOR et XNOR. Le chapitre présente les portes logiques de base (AND, OR, NOT, NAND, NOR, XOR) et montre comment construire à partir de portes un décodeur (activant l'une de plusieurs sorties) et un multiplexeur (sélectionnant l'une de plusieurs entrées). Deux siècles de mathématiques deviennent soudain des schémas de câblage.

L'intuition est que calcul et logique sont la même chose sous des habits différents. Une fois que Shannon les eut reliés, toute expression logique put être réalisée physiquement, et tout circuit put être analysé algébriquement. NAND et NOR méritent leur renommée parce que tout le reste peut être construit à partir d'elles. Le décodeur et le multiplexeur sont les premiers motifs dignes d'être mémorisés ; ils réapparaissent à chaque niveau de la machine, de la sélection des puces mémoire au routage des chemins de données.

À essayer: Construisez sur papier un multiplexeur 2-vers-1 à partir de portes AND, OR et NOT, puis écrivez sa table de vérité à partir du câblage.

L'horloge apprivoise le chaos dans les circuits

Des fils qui calculent, c'est facile ; des fils qui se souviennent, voilà la difficulté. Le chapitre présente les bascules et les verrous (par exemple, le verrou SR, la bascule D) comme cellules mémoire fondamentales d'un bit pour les circuits séquentiels, couvrant les circuits cadencés, les bascules D à déclenchement sur front, et les registres construits à partir d'elles. Les registres faits de bascules deviennent des registres à décalage et des compteurs, et l'horloge, avec le déclenchement sur front, est ce qui permet aux circuits synchrones d'éviter les conditions de course. Puis les cellules sont massées en matrices : des puces mémoire adressées via la sélection de puce, les lignes d'adresse et les lignes de données, avec une description de la SRAM et de la DRAM, en notant que la DRAM doit être rafraîchie périodiquement parce que ses condensateurs fuient.

La conception synchrone est un traité de paix : au lieu de laisser les signaux se battre de vitesse, tout le monde convient de ne bouger que lorsque l'horloge bat. Cette discipline unique rend analysables les conceptions à des millions de portes. Les condensateurs qui fuient de la DRAM sont un hack magnifique, échangeant un travail de rafraîchissement constant contre une densité énorme, raison pour laquelle la mémoire principale est de la DRAM et les caches de la SRAM. La mémoire comme gigantesque tableau adressé est le modèle mental qui survit jusqu'à la mémoire virtuelle.

À essayer: Dessinez un diagramme de temps d'une bascule D montrant pourquoi la sortie ne peut changer que sur le front d'horloge, et non quand l'entrée frétille.

Un processeur est une ville avec un horaire de bus

En prenant du recul par rapport aux portes individuelles, le chapitre assemble les processeurs comme une intégration complexe d'une UAL, de registres et de logique de contrôle sur une seule pièce de silicium, avec des broches organisées en signaux d'adresse, de données et de contrôle. Pour faire dialoguer les puces, le livre explique les protocoles de bus avec des diagrammes temporels de lecture et d'écriture, et aborde les bus au niveau logique, y compris les maîtres de bus, l'arbitrage et le tampon trois états comme dispositif clé qui permet à plusieurs pilotes de partager un même fil. C'est là que la machine cesse d'être un diagramme pour devenir un objet physique que l'on peut tenir en main, avec ses pattes métalliques et ses règles sur qui a le droit

de parler, et quand.

Le tampon trois états est le héros discret : sans un moyen de déconnecter une puce d'un fil partagé, les bus seraient impossibles. L'arbitrage transforme une ressource partagée en ressource ordonnée — le même problème que les systèmes d'exploitation résoudre plus tard avec leurs ordonnanceurs. Les diagrammes temporels enseignent la leçon la plus profonde du matériel : la correction n'est pas seulement une affaire de logique, c'est la logique plus le temps. Chaque lecture et chaque écriture est une séquence chorégraphiée de changements de signaux, et un seul temps de setup violé ruine tout.

À essayer: Trouvez en ligne le brochage d'un ancien processeur et classez ses broches à la main en groupes d'adresse, de données et de contrôle.

Microprogrammation : écrire la machine dans la machine

Au niveau de la microarchitecture, le livre construit le MIC-1 (Mic-1), une machine d'enseignement simple microprogrammée avec un chemin de données 32 bits (registres MAR, MDR, PC, SP, LV, CPP, TOS, OPC), une UAL plus un décaleur, et une mémoire de contrôle. Sa tâche est d'implémenter le jeu d'instructions IJVM (Integer Java Virtual Machine), qui comprend ILOAD, ISTORE, IADD, ISUB, BIPUSH, DUP, SWAP et IF_ICMPEQ, exécutées sur un modèle à pile. Le microprogramme implémente l'interpréteur IJVM via un cycle fetch-execute au niveau micro, écrit en MAL (Micro Assembly Language), avec des micro-instructions séquencées via MBR et MPC. Le format de micro-instruction comprend les champs : Addr (adresse suivante), JAM (condition de saut), ALU (opération de l'UAL), C (chargement de registre), Mem et B (source du bus).

C'est la démonstration la plus concrète de la thèse des couches : un jeu d'instructions peut être une interprétation s'exécutant sur un autre. L'IJVM ressemble à une machine réelle pour le programmeur, mais chacune de ses instructions est une courte séquence de micro-instructions lues dans la mémoire de contrôle. L'exécution à pile signifie que l'essentiel du travail consiste à empiler, dépiler et observer TOS, ce que le registre dédié du chemin de données anticipe. Une fois que vous avez suivi IADD jusqu'au niveau micro, l'expression « machine virtuelle » cesse d'être abstraite.

À essayer: Suivez IADD à la main : listez quels registres changent et quelle mémoire, le cas échéant, touche chaque micro-instruction.

x86 : quarante ans de bagages, et toujours en marche

Le niveau architecture du jeu d'instructions prend la famille Intel x86 (du 8086 au Pentium, IA-32) comme exemple fil rouge d'une conception CISC. Vous rencontrez son jeu de registres, son modèle de mémoire segmentée (registres de segment comme CS, DS, SS) et son encodage d'instructions à longueur variable, plus les modes d'adressage (immédiat, direct, par registre, indirect par registre, indexé, pile). Les types d'instructions (déplacement de données, arithmétique dyadique, opérations monadiques, comparaisons et branchements conditionnels,

appel et retour de procédure, E/S, manipulation de chaînes), avec l'utilisation de la pile pour sauvegarder les adresses de retour et les variables locales, et les procédures récursives. Le chapitre retrace aussi les idées RISC jusqu'aux travaux de David Patterson à Berkeley et John Hennessy à Stanford, notant que les processeurs x86 modernes traduisent en interne les instructions CISC en micro-opérations de type RISC.

L'ISA est le contrat entre le matériel et le logiciel, et le x86 montre ce qui arrive quand un contrat ne peut jamais être rompu : il s'encroûte. L'encodage à longueur variable et la segmentation sont une histoire que vous transportez dans chaque binaire moderne. L'histoire du RISC explique la torsion au cœur de chaque puce moderne : les instructions CISC que vous voyez sont une façade, décodées en dessous en micro-opérations simples de type RISC. La compatibilité a gagné le marché ; la simplicité a gagné le silicium.

À essayer: Désassemblez un petit programme C et identifiez une instruction de chaque famille de modes d'adressage listée par le chapitre.

La mémoire virtuelle : l'illusion dans laquelle vit chaque programme

Au niveau machine du système d'exploitation, la mémoire cesse d'être physique. Le chapitre explique la pagination (découper la mémoire en pages de taille fixe), les tables de pages (traduire les adresses virtuelles en adresses physiques) et les algorithmes de remplacement de pages comme LRU, FIFO et l'algorithme de la seconde chance. Un défaut de page bloque le programme pendant que le système va chercher la donnée sur disque ; le modèle de l'ensemble de travail, et le thrashing quand la mémoire disponible est trop faible. Il décrit aussi la segmentation comme alternative ou complément à la pagination, y compris la pagination segmentée dans l'architecture Intel Pentium. Sous tout cela, l'MMU (unité de gestion de la mémoire) traduit les adresses virtuelles en adresses physiques. Un processus est décrit comme un programme en cours d'exécution avec son propre espace d'adressage virtuel et ses ressources, avec ses états (Running, Ready, Blocked) et le rôle de l'ordonnanceur et du dispatcher dans la commutation de contexte.

La mémoire virtuelle est la plus grande escroquerie de l'informatique : chaque programme croit posséder la machine entière. Le mécanisme est une table de traduction par processus plus du matériel, c'est pourquoi l'MMU compte autant que le CPU. Le remplacement de pages est un problème de prédiction, et le modèle de l'ensemble de travail en est la théorie ; le thrashing est ce qui arrive quand la prédiction échoue à l'échelle du système. Le cycle Running-Ready-Blocked est le battement de cœur de chaque machine multitâche que vous avez jamais utilisée.

À essayer: Ouvrez le moniteur de mémoire de votre système, trouvez le nombre de défauts de page d'un processus en cours, et regardez-les s'envoler quand vous basculez entre applications lourdes.

Les sémaphores de Dijkstra apprennent aux processus à attendre leur tour

Les programmes à ce niveau demandent des services au système d'exploitation par des instructions d'E/S virtuelles au niveau machine du SE, y compris les opérations sur fichiers, la création de processus et la communication interprocessus, et comment le SE les implémente via des traps. Il couvre les pilotes de périphériques, les gestionnaires d'interruptions et le DMA (accès direct à la mémoire) pour un transfert de données efficace. Quand les processus doivent partager des ressources, le chapitre introduit les sémaphores, introduits par Edsger Dijkstra, pour résoudre les problèmes de section critique et de producteur-consommateur. Le niveau machine, il s'avère, est l'endroit où le système d'exploitation maintient la paix.

Les traps sont la charnière de tout le niveau : la même instruction qui ressemble à un appel de procédure franchit en réalité une frontière de protection vers le noyau. Le DMA compte parce que copier des données avec le CPU est du gâchis ; laisser le périphérique parler directement à la mémoire libère le processeur pour du vrai travail. Les sémaphores résolvent un problème qui paraît trivial, deux processus et une ressource, et dont toutes les solutions naïves échouent dans les interstices entre les instructions. L'intuition de Dijkstra fut de faire de l'attente elle-même un objet de première classe, dénombrable.

À essayer: Écrivez deux threads incrémentant un compteur partagé sans verrou, observez les mises à jour perdues, puis corrigez avec un sémaphore ou un mutex.

L'assembleur est un traducteur en deux passes avec de la mémoire

Le langage d'assemblage est défini comme une représentation symbolique du langage machine au niveau ISA, en correspondance un-à-un avec les instructions machine. Un assembleur travaille en deux passes, construisant une table de symboles à la première passe et générant le code à la seconde, traduisant les mnémoniques en opcodes et résolvant les étiquettes en adresses. Il couvre les macros et l'expansion de macros au moment de l'assemblage, distinguant les macros des procédures, une distinction qui surprend ceux qui pensent qu'elles sont interchangeables. Enfin, l'édition de liens et le chargement, y compris la relocalisation, les chargeurs absolus et avec édition de liens, et les formats de fichiers objets, la plomberie qui transforme des compilations séparées en un seul programme en marche.

La conception en deux passes existe à cause des références avant : on ne peut pas remplir un saut vers une étiquette qu'on n'a pas encore vue, donc la première passe collecte, la seconde résout. Macros contre procédures est en réalité un choix entre vitesse et espace, décidé à des moments différents, temps d'assemblage contre temps d'exécution. La relocalisation est la raison pour laquelle les fichiers objets existent : du code écrit sans connaître son adresse finale peut quand même être combiné. Chaque système de build que vous utilisez aujourd'hui hérite de ce pipeline.

À essayer: Écrivez un fichier assembleur de deux lignes avec un saut avant, puis simulez à la main la première et la seconde passe pour voir la table de symboles se remplir.

Les quatre cases de Flynn classent toutes les machines parallèles

Quand un seul processeur ne suffit plus, le chapitre fait appel à la taxonomie de Flynn : SISD (une seule instruction, une seule donnée), SIMD (une seule instruction, plusieurs données), MISD et MIMD, en discutant des machines SIMD, MIMD et vectorielles. Il explique les multiprocesseurs à mémoire partagée (SMP), y compris les SMP à bus, les commutateurs matriciels (crossbar) et les réseaux d'interconnexion multiniveaux comme le réseau omega. Puis vient le problème qui définit tout le domaine : le problème de la cohérence du cache. Sont présentées des solutions telles que les bus à écoute (snooping) et le protocole MESI (Modified, Exclusive, Shared, Invalid) pour maintenir la cohérence des caches de plusieurs processeurs.

Les quatre cases de Flynn sont le tableau périodique du domaine : presque toutes les machines que vous rencontrerez y trouvent leur place. Le vrai drame, c'est la cohérence, car les caches existent pour masquer la lenteur de la mémoire, et la masquer à plusieurs processeurs à la fois crée des mensonges. MESI fonctionne en faisant de chaque cache une commère sur le bus partagé, annonçant les changements d'état pour qu'aucun processeur n'agisse sur des données périmées. La hiérarchie des réseaux d'interconnexion, du bus au commutateur matriciel puis au réseau omega, est un compromis coût-contre-contention qui se répète à toutes les échelles.

À essayer: Trouvez la taille de ligne de cache et le nombre de cœurs de votre CPU, puis raisonnez sur l'état MESI dans lequel vivrait une table de correspondance partagée en lecture seule.

Le Cosmic Cube : les multiordinateurs apprennent à se passer des mots

L'autre voie vers le parallélisme abandonne entièrement la mémoire partagée. Dans les multiordinateurs à passage de messages, les processeurs disposent d'une mémoire privée et communiquent par des messages explicites via un réseau d'interconnexion. Le jalon historique fut le Cosmic Cube, construit à Caltech par Charles Seitz comme l'un des premiers multiordinateurs hypercube. Le chapitre discute des topologies de réseaux d'interconnexion telles que la grille, l'hypercube et les réseaux à commutation comme les commutateurs matriciels (crossbar). Là où le SMP se dispute une seule mémoire, le multiordinateur est une fédération de petites machines qui négocient par courrier.

Le passage de messages échange la commodité contre l'échelle : pas de problème de cohérence, puisque rien n'est partagé, mais chaque échange doit être explicite. L'hypercube est une topologie élégante, chaque nœud étant connecté à ses voisins binaires, ce qui maintient un diamètre logarithmique à mesure que la machine grandit. Le Cosmic Cube de Seitz a prouvé qu'un groupe de processeurs modestes pouvait se comporter comme une seule grande machine. C'est la lignée des centres de données d'aujourd'hui, où votre instance cloud est un nœud qui passe des messages à ses voisins.

À essayer: Dessinez un hypercube à 4 dimensions, étiquetez ses 16 nœuds avec des adresses binaires et vérifiez que chaque nœud est connecté à exactement 4 voisins.

Les meilleures citations

- "Génération Zéro" - Andrew S. Tanenbaum

- "moulin et magasin" - Andrew S. Tanenbaum
- "Modifié, Exclusif, Partagé, Invalide" - Andrew S. Tanenbaum
- "une seule instruction, plusieurs données" - Andrew S. Tanenbaum

Limites

The book's age shows at the edges: hardware examples and capacity numbers reflect the editions' era, so readers wanting current GPUs, NVMe storage, or modern multicore cache hierarchies will need supplements. The layered model, while pedagogically clean, smooths over how blurry the boundaries are in real designs, where compilers, microcode, and operating systems leak into each other. Some chapters are dense with detail, timing diagrams and protocol tables, that reward patience more than excitement. Absolute beginners may want a gentler on-ramp before tackling the microprogram chapters.

En une phrase

A computer is a tower of levels, each one a machine pretending to be the one above it, and you can climb every rung.

Pour qui

Computer science students facing architecture courses, and self-taught programmers who suspect their high-level language is hiding a machine they should meet. Also ideal for curious engineers who want the whole stack, gates to grids, in one coherent frame.