

[STORMRGS]



But How Do It Know?

The Basic Principles of Computers for Everyone

J. Clark Scott

MAIS COMMENT LE SAVOIR ?

Les principes de base des ordinateurs

Pour tout le monde

Par

J. Clark Scott

Copyright © 2009 par John Clark Scott

Conception informatique incorporée ici
Copyright © 2009 par John Clark Scott

Tous les droits sont réservés

Publié par 34677

John C. Scott, Oldsmar, Floride

ISBN 978-0-615-30376-5

buthowdoitknow.com

Couverture, photographie et conception par Alexander C. Scott III artbyalexscoff.com

Imprimé aux États-Unis d'Amérique

Première édition : Juillet 2009

10 9 8 7 6 5 4 3 2 1

Table des matières

Table des matières

Introduction Juste les faits

Madame Speed Language Juste

un peu Qu'est-ce que... ?

Diagrammes de variations

simples Rappelez-vous

quand Que pouvons-nous

faire avec un mors ?

Une rose par n'importe quel autre nom

Huit, c'est assez de codes Retour à

l'octet Le bus magique Plus de

combinaisons de portes La première

moitié de l'ordinateur Numéros

d'adresses L'autre moitié de l'ordinateur

Plus de portes jouant avec les octets Les

manettes gauche et droite Le NOTter

L'ANDer L'ORer Le OR exclusif

L'additionneur Le comparateur et la logique zéro

L'unité arithmétique et logique Plus du processeur

L'horloge Faire quelque chose d'utile pas à pas

Tout est sous contrôle Faire quelque chose

d'utile, revisité Et ensuite ?

Les instructions de la première grande invention L'instruction d'arithmétique ou de logique Les instructions de chargement et de stockage
L'instruction de données La deuxième grande invention
Une autre façon de sauter La troisième grande invention
L'instruction Clear Flags Ta Daa!

Quelques mots de plus sur l'arithmétique Le monde extérieur Le clavier L'écran d'affichage
Un autre code Le dernier mot sur les codes Le disque Excusez-moi madame C'est tout le matériel et les programmes logiciels Les langues du système d'exploitation Les erreurs du système de fichiers Maladies informatiques ?

Firmware
Boots Digital
vs Analog Il lied - Sorte de philosophie de divulgation complète

Introduction

Le titre de ce livre est la réplique d'une vieille blague qui se résume ainsi : Joe est un type très gentil, mais il a toujours été un peu lent. Il entre dans un magasin où un vendeur se tient debout sur une caisse à savon devant un groupe de personnes. Le vendeur lance la nouvelle invention miracle, la bouteille Thermos. Il dit: "Il garde les aliments chauds au chaud et les aliments froids au froid..." Joe y réfléchit une minute, étonné par cette nouvelle invention qui est capable de décider laquelle de deux choses différentes elle est censée faire. selon le type de nourriture que vous y mettez. Il ne peut contenir sa curiosité, il saute, agite son bras en l'air, disant « mais, mais, mais, mais... » Finalement, il lâche sa question brûlante « Mais comment sait-il ?

Vous avez peut-être ri ou non de la blague, mais le fait est que Joe a examiné ce que cette bouteille thermos pouvait faire et a décidé qu'elle devait être capable de détecter quelque chose à propos de son contenu, puis d'effectuer une opération de chauffage ou de refroidissement en conséquence. Il pensait qu'il devait contenir un radiateur et un réfrigérateur. Il n'avait aucune idée du principe beaucoup plus simple sur lequel il fonctionne réellement, à savoir que la chaleur tende toujours de se déplacer d'une zone plus chaude vers une zone plus froide, et tout ce que fait le Thermos est de ralentir ce mouvement. Avec un contenu froid, la chaleur extérieure est ralentie à l'entrée et avec un contenu chaud, la chaleur est ralentie à la sortie. La bouteille n'a pas à "savoir" pour remplir sa mission et ne chauffe ni ne refroidit rien. Et finalement, le contenu, chaud ou froid, se retrouve à température ambiante.

Mais le concept de Joe sur le fonctionnement de la bouteille était bien plus compliqué que la vérité.

Donc, la raison du titre du livre, c'est que lorsqu'il s'agit d'ordinateurs, les gens les regardent, voient ce qu'ils peuvent faire et imaginent toutes sortes de choses qui doivent être dans ces machines. Ou ils imaginent toutes sortes de principes sur lesquels ils doivent se baser, et donc ce dont ils peuvent être capables. Les gens peuvent attribuer des qualités humaines à la machine. Et plus que quelques-uns se retrouvent dans des situations où ils se sentent embarrassés, comme notre ami dans la blague, Joe.

Mais les ordinateurs sont en fait assez faciles à comprendre. Bien sûr, les ordinateurs ont un plus grand nombre de pièces qu'une bouteille thermos, mais chaque pièce est extrêmement simple, et elles fonctionnent toutes sur un principe très simple et très facile à comprendre.

Avec le Thermos, le principe est celui du mouvement de la chaleur. C'est quelque chose que nous pouvons observer dans la vie. Nous voyons les glaçons fondre lorsqu'ils sont retirés du congélateur, et nous voyons le repas chaud refroidir sur la table lorsque la famille est en retard pour le dîner.

Dans l'ordinateur, le principe de fonctionnement est lié à l'électricité, mais cela ne veut pas dire qu'il est difficile à comprendre. Si vous avez observé le fait que lorsque vous allumez un interrupteur, une ampoule s'allume et que lorsque vous éteignez l'interrupteur, la lumière s'éteint, alors vous avez observé le principe de fonctionnement des ordinateurs. C'est à peu près tout ce que vous devez savoir sur l'électricité pour comprendre les ordinateurs.

Juste les faits Madame

Ce livre n'est pas avant tout destiné à être un manuel.

Il n'y a pas de problème à faire à la fin de chaque chapitre.

Son intention est simplement de démystifier le sujet des ordinateurs pour quiconque s'est déjà demandé ce qui se passe à l'intérieur de cette boîte. Bien sûr, cela constitue également une parfaite introduction à l'informatique pour un jeune qui finira par obtenir un doctorat en informatique.

Mais il doit être facilement compréhensible par les femmes au foyer, les personnes âgées et les enfants qui savent bien lire. Il doit être compréhensible pour les plombiers et les balayeurs de rue. Il ne nécessite aucune formation technique préalable.

Il suffit que vous sachiez lire la langue, que vous puissiez allumer et éteindre une ampoule et que vous puissiez faire une addition très simple de l'ordre de $8 + 5 = 13$.

Ce livre présente les éléments essentiels complets qui composent un ordinateur. Il présente chaque pièce et chaque partie, dans le bon ordre afin que chacune ait un sens et puisse être comprise. Chaque partie est expliquée en détail et chaque nouveau mot est défini en détail lors de sa première utilisation.

Toute tentative de simplifier davantage le sujet laisserait des lacunes dans la vue d'ensemble où quelqu'un devrait encore deviner comment les pièces fonctionnent ensemble, et vous n'auriez jamais ce "Aha, je comprends!" moment que je pense que vous aurez bientôt.

Ce livre n'est pas une version simplifiée d'un manuel universitaire. C'est une explication complète des principes de base des ordinateurs. C'est un livre technique, mais c'est aussi un livre de cuisine et c'est aussi un manuel d'éducation du conducteur.

Ce livre commence juste au début et définit chaque élément nécessaire pour comprendre la machine. Peu importe ce que quelqu'un sait déjà sur les ordinateurs, cela comblera toutes les pièces manquantes et les rassemblera en quelque chose qui a du sens.

Même notre ami, Joe, pourrait comprendre ce livre avec une étude diligente. Il y a des milliers de mots et d'idées associés au domaine des ordinateurs qui donnent l'impression que tout le sujet est un gâchis. Mais les concepts de base qui les sous-tendent sont simples.

Dans ce livre, il n'y aura pas de volume d'anecdotes sur

la construction ou l'histoire des ordinateurs, juste l'essentiel, ni plus ni moins. Chaque partie de l'ordinateur a une fonction simple, et lorsqu'elles sont connectées ensemble, vous vous retrouvez avec une machine utile appelée ordinateur.

Il n'y a rien à mémoriser dans ce livre. Chaque chapitre est conçu pour vous donner une nouvelle idée que vous n'aviez pas auparavant, ou si c'est quelque chose dont vous aviez entendu parler auparavant, cela a toujours semblé déroutant. Chaque idée est très simple, et une chose mène à la suivante. Chaque chapitre présente une idée. Chaque idée est simple et facile à comprendre. Les chapitres suivants présentent des idées qui s'appuient sur les idées des chapitres précédents.

Si quelqu'un écrivait un livre sur la construction d'une maison, il pourrait y avoir différents niveaux de détail. Le livre le plus simple dirait : « posez les fondations, montez les murs, couvrez d'un toit, installez la plomberie et l'électricité, et le tour est joué ». Ce ne serait pas assez détaillé pour quelqu'un qui n'a pas déjà une certaine expérience de l'utilisation d'un marteau et d'une scie et de l'installation d'un robinet et du câblage d'un interrupteur d'éclairage.

À l'autre extrémité du spectre se trouverait un livre contenant des chapitres distincts pour chaque type de fondation possible, les différents types de saleté que vous pourriez avoir à creuser, des formules pour une douzaine de types de béton différents, des tableaux des conditions météorologiques optimales pour poser des fondations, etc. Ce serait beaucoup trop d'informations. Il y aurait tellement de détails, que ce qui était vraiment important serait perdu.

Ce livre tente de donner juste assez de détails pour voir ce que chaque ordinateur a en commun et comment ils fonctionnent, pas comment construire le plus grand ou le meilleur ordinateur jamais conçu. Il ne s'agit pas d'une marque spécifique d'ordinateur. Il ne s'agit pas de savoir comment utiliser un ordinateur. S'il s'agissait d'un livre sur la construction d'une maison, il décrirait probablement un plan simple pour un abri de jardin solide avec un évier et une ampoule nue, montrant la taille et la forme de chaque morceau de bois, où mettre tous les clous, comment pour accrocher la porte et comment assembler les conduites d'eau pour qu'elles ne fuient pas. Cela ne montrerait pas comment construire quelque chose d'aussi compliqué qu'un escalier en chêne incurvé de fantaisie.

Nous allons montrer la partie simple que les ordinateurs

sont constitués, puis connectez-en plusieurs ensemble jusqu'à ce que nous ayons construit un ordinateur complet. Cela va être beaucoup plus simple que vous ne l'auriez jamais imaginé.

Les

ordinateurs de vitesse semblent mystérieux et magiques. Comment peuvent-ils faire ce qu'ils font ? Ils jouent à des jeux, ils dessinent des images, ils « connaissent » votre cote de crédit. Ces machines sont capables de faire toutes sortes de choses étranges et merveilleuses. Pourtant, ils sont simples. Ils ne peuvent faire que très peu de choses très simples. Et, ils ne peuvent faire qu'une seule de ces choses simples à la fois. Ils semblent faire des choses complexes, uniquement parce qu'ils font un grand nombre de choses simples les unes après les autres en peu de temps. Le résultat, comme dans un jeu vidéo, est très complexe en apparence, mais en réalité, il est très simple, juste très très rapide.

Les ordinateurs sont conçus pour faire un petit nombre de choses simples spécifiques, et pour faire ces choses rapidement, l'une après l'autre. Quelles tâches simples sont effectuées et dans quel ordre déterminent le type de tâche que l'ordinateur accomplit à un moment donné, mais tout ce que fait l'ordinateur ne consiste en rien en dehors de ses capacités limitées.

Une fois que vous aurez vu de quoi est composé un ordinateur, vous vous rendrez compte qu'il peut faire ce qu'il fait, exactement de quel genre de choses il est capable et aussi de quoi il n'est pas capable.

Ainsi, le secret des ordinateurs n'est pas qu'ils sont complexes, c'est plutôt leur rapidité. Regardons exactement à quelle vitesse leur vitesse est.

Puisque les ordinateurs fonctionnent à l'électricité, leur vitesse est liée à la vitesse de l'électricité. Vous vous souvenez peut-être d'avoir entendu dire que la vitesse de la lumière était de 186 000 miles par seconde. C'est sacrément rapide. La lumière peut faire le tour de la Terre entière sept fois en une seconde, ou de la Terre à la Lune en environ une seconde et demie. Selon les physiciens, l'électricité a de nombreuses propriétés en commun avec la lumière, et sa vitesse, lorsqu'elle se déplace dans un fil, est ralentie à environ la moitié de la vitesse de la lumière. Mais encore, faire tout le tour de la Terre trois fois et demi en une seconde est extrêmement rapide.

À titre de comparaison, imaginez qu'il fait chaud et que vous avez un ventilateur électrique assis sur la table qui souffle

air frais sur vous. Le ventilateur tourne si vite que les pales sont floues, mais il ne tourne qu'environ 40 fois par seconde. Un point sur le bord de l'une de ces pales ne parcourra qu'environ 150 pieds dans cette seconde, il faudra 35 secondes pour que ce point parcoure seulement un mile.

Étant donné que les pales du ventilateur sont déjà floues, il peut être difficile de les imaginer aller dix fois plus vite. Si c'était le cas, ce ventilateur éteindrait un jeu d'enfant. Et si vous pouviez le faire aller cent fois plus vite, il s'autodétruirait presque certainement, les pales du ventilateur se brisant et se coinçant dans le plafond. Mais l'électricité voyageant dans le même cercle ferait environ cent millions de fois en une seconde, soit deux millions et demi de fois plus vite que les pales du ventilateur. C'est rapide.

Un million est un très grand nombre. Si vous preniez une grande feuille de papier de 40 pouces carrés et que vous preniez une règle et que vous la placiez sur le bord supérieur, et que vous dessiniez 25 points par pouce le long du bord supérieur du papier, vous devriez dessiner mille points pour traverser cette feuille de papier. Si vous déplaciez ensuite la règle vers le bas de la page de 1/25 de pouce et que vous dessiniez mille autres points, et que vous continuiez à le faire, vous devriez déplacer la règle vers le bas de la page mille fois, en dessinant à chaque fois mille points. Si vous pouviez accomplir une tâche aussi ennuyeuse, vous vous retrouveriez avec un morceau de papier avec un million de points dessus. C'est beaucoup de points ou beaucoup de n'importe quoi. Et juste pour finir la pensée, si vous pouviez trouver un millier de personnes qui dessinaient chacune une de ces millions de feuilles à points, et empilaient ces milliers de feuilles en une pile, vous auriez alors un milliard de points.

Supposons maintenant que l'électricité se déplaçant à l'intérieur de l'ordinateur puisse accomplir une tâche simple en parcourant un pied. Cela signifie que l'ordinateur pourrait faire 500 millions de choses simples en une seconde. Encore une fois à titre de comparaison, le ventilateur sur la table tournera pendant 7 heures pour faire le tour d'un million de fois et il faudra six mois complets pour qu'il tourne environ 500 millions de fois.

Lorsque vous parlez de la vitesse à laquelle l'électricité peut se déplacer entre les pièces à l'intérieur de l'ordinateur, certaines des pièces que vous pouvez voir sont distantes d'un pied, d'autres plus proches, d'un pouce, d'un

dixième de pouce. Et à l'intérieur de ces pièces se trouvent une multitude d'autres pièces qui sont très proches les unes des autres, à quelques millièmes de pouce d'intervalle. Et plus la distance parcourue par l'électricité est courte, plus elle y parvient rapidement.

Il est inutile de dire combien de choses les ordinateurs d'aujourd'hui font en une seule seconde, car cela daterait ce livre. Les fabricants d'ordinateurs continuent de produire de nouveaux ordinateurs qui vont deux fois plus vite que les ordinateurs les plus rapides d'il y a seulement deux ou trois ans. Il y a une limite théorique à la vitesse à laquelle ils peuvent aller, mais les ingénieurs continuent de trouver des moyens pratiques de contourner les théories et de fabriquer des machines qui vont de plus en plus vite.

Pendant tout ce temps où les ordinateurs sont devenus plus rapides, plus petits et moins chers, les choses que font les ordinateurs n'ont vraiment pas changé depuis leur invention dans les années 1940. Ils font toujours les mêmes petites choses simples, juste plus rapides, moins chères, plus fiables et dans un boîtier plus petit.

Il n'y a que quelques sections dans un ordinateur, et elles sont toutes constituées du même type de pièces. Chaque section a une mission spécifique, et la combinaison de ces pièces dans une machine était une invention vraiment merveilleuse. Mais ce n'est pas difficile à comprendre.

Langue

Dans ce livre, nous allons devoir définir quelques mots qui sont utilisés pour décrire les pièces à l'intérieur d'un ordinateur.

Dans certaines professions, notamment la médecine et la justice, on a tendance à inventer beaucoup de nouveaux mots, et à les prendre des langues grecques et latines anciennes, et à les rendre longs et difficiles à prononcer.

Dans le monde des ordinateurs, il semble que les inventeurs pionniers étaient des personnes moins formelles. La plupart des mots qu'ils ont utilisés sont des mots simples du langage courant, des mots qui existaient déjà, mais qui sont utilisés d'une manière nouvelle.

Certains des nouveaux mots sont des mots que nous connaissons déjà, utilisés comme une partie différente du discours, comme un nom existant maintenant utilisé comme verbe. Certains mots sont des acronymes, les premières lettres des mots d'une phrase.

Chaque mot sera décrit en détail lors de sa première utilisation. Et bien qu'il y ait des milliers de mots et d'acronymes en usage si l'on considère l'ensemble de l'industrie informatique, il n'y a qu'une douzaine ou deux mots nécessaires pour comprendre l'ordinateur lui-même. Vous avez probablement déjà entendu certains de ces mots et compris ce qu'ils signifiaient à partir de la façon dont ils étaient utilisés, mais vous obtiendrez maintenant les définitions appropriées et complètes. Dans de nombreux cas, vous constaterez peut-être qu'ils sont plus simples que vous ne le pensiez.

Juste un petit peu

Qu'y a-t-il dans un ordinateur ? Il vous montre des images fixes, des images animées, de la musique, votre échiquier, les lettres que vous avez écrites, il joue à des jeux vidéo, communique partout dans le monde, et bien plus encore. Mais y a-t-il des images à l'intérieur de l'ordinateur ? Si vous sortez un microscope et saviez où chercher, pourriez-vous trouver de petites images quelque part à l'intérieur de l'ordinateur ? Voudriez-vous voir des « A » et des « B » et des « 8 » et des « 12 » se déplacer là-dedans quelque part ?

La réponse est non, il n'y a pas d'images, de chiffres ou de lettres dans un ordinateur. Il n'y a qu'un seul type de chose dans un ordinateur. Il y a un grand nombre de ce genre de choses, mais il n'y a qu'un seul genre de chose là-dedans. Ça s'appelle un peu.

Lorsque vous lancez une pièce en l'air et que vous la laissez tomber sur le sol, elle se retrouvera sur le sol dans l'un des deux états possibles - avec soit la tête visible, soit la

queue.

La lumière de votre salon (en supposant que vous ayez un interrupteur et non un gradateur) peut être allumée ou éteinte.

La serrure de votre porte d'entrée peut être verrouillée ou déverrouillée.

Qu'est ce que toutes ces choses ont en commun? Ce sont tous des endroits qui contiennent une chose qui peut être dans l'un des deux états possibles. C'est la définition d'un peu.

Un bit est une sorte d'objet physique qui a une taille et un emplacement dans l'espace, et il a une certaine qualité sur lui-même, qui à tout moment peut être dans l'un des deux états possibles, et peut être amené à revenir en arrière et

entre ces deux États.

Un morceau d'argile n'est pas un morceau. Il peut être moulé en une boule, un cube, une crêpe, un anneau, une bûche, un visage ou toute autre chose à laquelle vous pouvez penser. Il a une taille et un emplacement dans l'espace, mais il y a trop d'états dans lesquels il peut être pour qu'on l'appelle un peu. Si vous preniez ce morceau d'argile, l'aplatissiez, grattiez "oui" d'un côté et "non" de l'autre côté, puis le mettiez dans un four et le cuisez jusqu'à ce qu'il soit dur, alors vous pourriez

pouvoir l'appeler un peu. Il pourrait s'asseoir sur une table avec le "oui" ou le "non" affiché. Il n'aurait alors que deux états.

Vous avez probablement déjà entendu parler de bits en relation avec les ordinateurs, et maintenant vous savez ce que c'est. Dans un ordinateur, les bits ne sont pas comme la pièce de monnaie ou la serrure, ils ressemblent plutôt à la lumière. Autrement dit, les bits d'un ordinateur sont des endroits qui ont de l'électricité ou qui n'en ont pas. Dans un ordinateur, les bits sont très, très petits et il y a un très grand nombre de bits, mais c'est tout ce qu'il y a dedans.

Comme la lumière dans le salon, le mors est allumé ou éteint. Dans le salon, il y a de l'électricité dans le mur qui arrive dans l'interrupteur. Lorsque vous allumez l'interrupteur, l'électricité passe de l'interrupteur, à travers les fils dans le mur et le plafond, dans la douille de lumière, puis dans l'ampoule. Donc, ce morceau dans le salon fait plusieurs pieds de long, il comprend l'interrupteur, les fils, la prise et l'ampoule. Dans un ordinateur, les bits sont pour la plupart minuscules, en fait microscopiques. De plus, le bit d'ordinateur n'a pas d'interrupteur mécanique à une extrémité ni d'ampoule à l'autre. Si vous retiriez l'ampoule de la prise dans le salon, l'interrupteur enverrait toujours de l'électricité à la prise lorsqu'elle était allumée, et ce serait encore un peu - vous ne pourriez tout simplement pas voir si elle était allumée ou éteint en regardant une ampoule.

Votre ordinateur a quelque chose qui ressemble à des commutateurs, comme les touches du clavier, et quelque chose qui ressemble à des ampoules, comme les minuscules points sur l'écran, mais la plupart des bits sont à l'intérieur et invisibles.

C'est fondamentalement tout ce qu'il y a dans un ordinateur - des bits. Il y en a beaucoup, beaucoup, et ils sont arrangés et connectés de diverses manières, que nous examinerons en détail au fur et à mesure que le livre progresse, mais c'est ce qu'il y a à l'intérieur de tous les ordinateurs - des bits. Un bit est toujours dans l'un de ses deux états possibles, éteint ou allumé, et ils changent entre allumé et éteint quand on leur dit de le faire.

Les bits d'ordinateur ne sont pas comme la pièce de monnaie qui doit physiquement se retourner pour passer d'un état à l'autre. Les bits ne changent pas de forme ou d'emplacement, ils n'ont pas l'air différents, ils ne bougent pas, ne tournent pas, ne deviennent pas plus grands ou plus petits. Un bit d'ordinateur n'est qu'un endroit, s'il n'y a pas d'électricité à cet endroit, alors le bit est

à l'arrêt. Lorsque l'électricité est présente, le bit est allumé.

Si vous voulez changer une pièce de monnaie montrant des têtes en montrant des queues, vous devez la déplacer physiquement pour la retourner, ce qui prend un certain temps. Parce que la seule chose qui doit bouger dans un bit d'ordinateur est l'électricité, changer son état d'arrêt à marche ou de marche à arrêt peut se produire beaucoup plus rapidement que tout ce qui doit être déplacé physiquement.

Comme autre exemple, vous souvenez-vous du Far West américain dans les films ? Il y avait de petites villes séparées par de vastes distances. Les grandes villes auraient un bureau télégraphique. Dans ce bureau, il y avait un gars portant un drôle de chapeau qui avait un interrupteur à ressort appelé clé, et il envoyait des messages en appuyant sur cette touche et en l'éteignant dans certains motifs qui représentaient les lettres de l'alphabet.

Cette clé était connectée à une batterie (oui, ils avaient des batteries à l'époque) et à un fil qui était suspendu le long de poteaux jusqu'à ce qu'il atteigne une autre ville. La clé connectait simplement la batterie au fil lorsqu'elle était enfoncée et déconnectait la batterie lorsque la touche n'était pas enfoncée. Dans l'autre ville, il y avait un autre bureau de télégraphe, le fil entrait dans ce bureau, son extrémité était enroulée autour d'une tige de fer (qui se transforme en aimant lorsqu'il y a de l'électricité dans le fil), la tige magnétisée attirait une petite barre de fer tenu à proximité avec un ressort, et faisait un bruit de cliquetis chaque fois que l'électricité arrivait. Le gars dans le bureau a écouté le motif du clic et a écrit les lettres du message. Ils auraient pu utiliser une ampoule au lieu du clicker, sauf que les ampoules n'avaient pas encore été inventées.

L'intérêt d'aborder ce sujet, c'est que tout cet appareil télégraphique, de la touche qui est enfoncée dans une ville, en passant par le long fil qui se rend dans une autre ville à plusieurs kilomètres de distance, jusqu'au cliqueur, tout cet appareil ne comprend qu'un seul bit . C'est un endroit qui peut avoir ou non de l'électricité, et qui s'allume et s'éteint comme on le dit. Et cette méthode de communication a révolutionné le monde à bien des égards. Mais cette invention très importante des années 1840 ne consistait en rien de plus qu'un morceau.

J'espère donc que cela commence à simplifier le sujet de

ordinateurs pour vous. Il n'y a qu'une seule chose à l'intérieur des ordinateurs, des bits. Beaucoup d'entre eux pour être sûr, mais quand vous comprenez des morceaux, vous comprenez ce qu'il y a dedans.

Qu'est-ce que...?

Imaginez que c'est une journée ensoleillée et que vous entrez dans une pièce avec de nombreuses fenêtres ouvertes. Vous remarquez que le plafonnier est allumé. Vous décidez que c'est du gâchis et vous allez éteindre la lumière. Vous regardez le mur à côté de la porte et voyez une plaque d'interrupteur avec deux interrupteurs. Donc, vous supposez que celui le plus proche de la porte est pour le plafonnier. Mais ensuite, vous remarquez que l'interrupteur est déjà éteint. Et l'autre interrupteur est éteint aussi.

Alors vous pensez "eh bien, peut-être que quelqu'un a installé l'interrupteur à l'envers", alors vous décidez quand même d'actionner l'interrupteur. Vous l'allumez et l'éteignez mais rien ne se passe, le plafonnier reste allumé. Alors vous décidez que ce doit être l'autre interrupteur, et vous l'allumez, l'éteignez, l'allumez, l'éteignez.

Encore une fois, rien ne se passe, ce plafonnier continue de briller vers vous. Vous regardez autour de vous, il n'y a pas d'autre porte, il n'y a pas d'autres interrupteurs, aucun moyen apparent d'éteindre cette maudite lumière. Ça doit juste être l'un de ces deux interrupteurs, qui a construit cette maison folle de toute façon ? Alors vous attrapez un interrupteur avec chaque main et commencez à les retourner sauvagement. Puis soudain, vous remarquez que le plafonnier s'éteint brièvement. Ainsi, vous ralentissez le basculement de votre interrupteur et vous vous arrêtez lorsque le plafonnier est éteint. Les deux interrupteurs disent "on", et la lumière est maintenant éteinte. Vous éteignez, puis rallumez un interrupteur, et la lumière s'allume, puis s'éteint. C'est à l'envers. Un éteindre équivaut à allumer la lumière ? Ensuite, vous éteignez l'autre interrupteur, puis vous le rallumez, la même chose, la lumière s'allume, puis s'éteint. Que diable? Quoi qu'il en soit, vous comprenez enfin comment cela fonctionne. Si les deux interrupteurs sont allumés, la lumière s'éteint. Si l'un ou l'autre ou les deux interrupteurs sont éteints, alors le plafonnier est allumé. Un peu maladroit, mais vous accomplissez ce que vous vouliez, vous allumez les deux interrupteurs, la lumière s'éteint et vous sortez de cette pièce folle.

Maintenant, quel est le but de cette petite histoire sur les interrupteurs étranges ? La réponse est que, dans ce chapitre, nous allons présenter la partie la plus élémentaire dont sont faits les ordinateurs. Cette partie fonctionne exactement comme le système d'éclairage de cette pièce étrange.

Cette partie informatique est un appareil simple qui a trois connexions où il peut y avoir ou non des

électricité. Deux de ces connexions sont des endroits où l'électricité peut être introduite dans l'appareil, et la troisième connexion est un endroit où l'électricité peut sortir de l'appareil.

Sur les trois connexions, deux d'entre elles sont appelées "entrées", car l'électricité peut leur être envoyée d'ailleurs. La troisième connexion est appelée la "sortie" car l'électricité peut en sortir et ensuite être envoyée ailleurs.

Cette partie informatique est un appareil qui fait quelque chose avec des bits. Si vous avez deux bits et que vous connectez ces deux bits aux entrées, cet appareil "regarde" ces deux bits et "décide" s'il faut activer ou désactiver le bit de sortie.

La façon dont il « décide » est très simple, et est toujours la même. Si les deux entrées sont activées, la sortie sera désactivée. Si une ou les deux entrées sont désactivées, la sortie sera activée. C'est juste la façon dont la pièce avec les interrupteurs d'éclairage étranges fonctionnait.

N'oubliez pas qu'il n'y a que des bits à l'intérieur de l'ordinateur. Ce dispositif simple est d'où viennent les bits et où ils vont. La « décision » que prend cet appareil concerne la manière dont les bits sont activés et désactivés dans un ordinateur.

Deux bits entrent dans l'appareil et un bit en sort. Deux bits viennent d'ailleurs, sont examinés par l'appareil, et un nouveau troisième bit est généré pour qu'il puisse aller ailleurs.

Si vous avez été très observateur, vous vous êtes peut-être posé cette question : "lorsque les deux entrées sont désactivées, la sortie est activée, alors.... comment obtenez-vous de l'électricité à la sortie si les deux entrées sont éteintes ? " Eh bien, c'est une excellente question, et l'excellente réponse est que chacun de ces appareils est également connecté à l'alimentation.

Comme chaque appareil ou lampe de table de votre maison, où chacun a une prise à deux broches, cet appareil a une paire de fils, dont l'un est connecté à un endroit où l'électricité est toujours allumée, et l'autre est connecté à un endroit où l'électricité est toujours coupée. C'est de là que vient l'électricité pour la sortie. Quand quelqu'un construit un ordinateur, il doit établir toutes ces connexions électriques à chacune de ces pièces pour qu'il fonctionne, mais quand nous dessinons des schémas

des pièces, comment elles sont connectées et ce qu'elles feront, nous ne prendrons pas la peine de dessiner les fils d'alimentation - ils ne feraient qu'encombrer le dessin. Il est entendu que chaque partie a sa connexion électrique, et nous ne nous en soucions pas. Comprenez simplement qu'il est là, et nous n'en parlerons plus pour le reste du livre. Je ne l'aurais même pas mentionné ici, sauf que je me suis dit que vous vous poseriez probablement cette question tôt ou tard.

Maintenant, je sais que j'ai dit qu'il n'est pas nécessaire de comprendre grand-chose à l'électricité pour comprendre les ordinateurs. Ici, c'est aussi compliqué que possible. Il y a en fait une demi-douzaine de composants électroniques à l'intérieur de cet appareil qui le font fonctionner, mais nous n'allons pas examiner ces composants dans ce livre. Quelqu'un qui a une formation en électronique pourrait regarder ce qu'il y a dedans, et en environ 30 secondes dirait "Oh oui, si les deux entrées sont activées, la sortie sera désactivée, et pour toute autre combinaison, la sortie sera activée, tout comme le dit le livre. Et puis cette personne pourrait aller de l'avant et lire ce livre sans jamais avoir à penser à nouveau à ce qu'il contient. Quelqu'un qui ne connaît pas l'électronique passe à côté de ces quelques secondes de compréhension, mais ce livre est le même pour tout le monde.

Dans le câblage domestique normal, un interrupteur allume et éteint une lumière. Dans l'ordinateur, il faut deux interrupteurs, et c'est en quelque sorte à l'envers en ce sens qu'ils doivent tous les deux être allumés pour éteindre la lumière. Mais si vous acceptez le fait que quelque chose pourrait être fait qui fonctionne de cette façon, vous pouvez alors comprendre comment tout fonctionne dans l'ordinateur.

Ce type de pièce d'ordinateur est en fait le SEUL type de pièce nécessaire pour construire un ordinateur. Bien sûr, il en faut beaucoup pour construire un ordinateur complet, mais avec suffisamment d'entre eux, vous pouvez fabriquer n'importe quel type d'ordinateur. Alors voilà, vous voyez à quel point un ordinateur est simple ? C'est juste plein de ce genre de petites choses - beaucoup d'entre elles bien sûr, mais c'est tout ce qu'il y a.

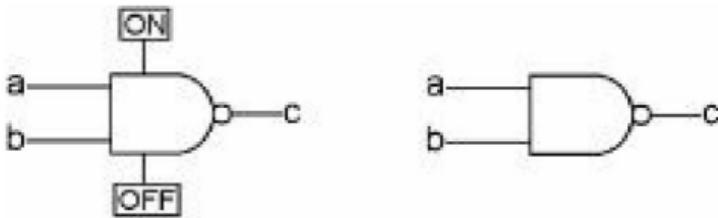
Maintenant, nous devons donner un nom à cet appareil, cette chose à l'intérieur de l'ordinateur dont sont faits les bits, cela s'appelle une "porte". Je ne trouve pas de bonne raison pour laquelle cela s'appelle une porte, une porte dans une clôture laisse passer les gens lorsqu'elle est ouverte et arrête les gens lorsqu'elle est fermée. Un ordinateur

gate génère un troisième bit à partir de deux autres bits, il ne s'ouvre pas et ne se ferme pas, ne s'arrête pas et ne laisse rien passer.

La signification de ce terme informatique "porte" ne semble pas correspondre au sens commun du mot, mais désolé, je n'ai pas inventé le nom, c'est juste comme ça qu'il s'appelle.

Tu t'y habitueras. Au moins, ce n'est pas un long mot du grec ancien.

Dans les prochains chapitres, nous allons montrer comment nous pouvons faire quelque chose d'utille en connectant plusieurs portes ensemble. Nous utiliserons des dessins comme celui-ci. La forme en « D » avec le petit cercle à son extrémité représente l'appareil que nous avons décrit, et les lignes représentent les fils entrant et sortant de celui-ci qui sont attachés à d'autres parties de l'ordinateur. L'image de gauche montre une porte complète avec ses fils électriques, mais comme promis, nous ne nous en occuperons pas pour le reste de ce livre. Le dessin à droite montre tout ce dont nous avons besoin :



Ceci est une représentation d'une porte. Les deux fils de gauche (a et b) sont les entrées et le fil de droite (c) est la sortie. Les trois fils sont des bits, ce qui signifie qu'ils sont allumés ou éteints. Chaque bit d'entrée provient d'un autre endroit de l'ordinateur et est activé ou désactivé en fonction de ce qui se passe d'où il vient, puis cette porte active ou désactive sa sortie en fonction de l'état de ses deux entrées.

Parfois, il est utile de faire un petit tableau qui montre comment les différentes combinaisons d'entrée créent la sortie, comme ceci :

un	b	c
----	---	---

À l'arrêt	À l'arrêt	Sur
À l'arrêt	Sur	Sur
Sur	À l'arrêt	Sur
Sur	Sur	À l'arrêt

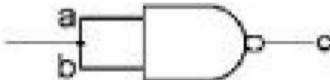
Chaque ligne montre une combinaison possible des entrées,
et quelle sera la sortie dans ces circonstances.

Comparez ce petit tableau avec l'expérience avec le
pièce étrange avec les deux interrupteurs d'éclairage. Si un interrupteur est
appelé 'a', l'autre interrupteur s'appelle 'b' et le
plafonnier s'appelle 'c', alors ce petit tableau
décrit complètement et exactement comment l'équipement
cette pièce fonctionne. La seule façon d'éteindre cette lumière
est d'avoir à la fois l'interrupteur 'a' et l'interrupteur 'b'.

Variantes simples

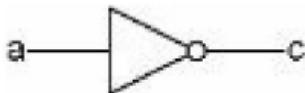
Comme mentionné, cette porte est la seule chose dont vous avez besoin pour construire un ordinateur, mais vous en avez besoin de beaucoup, et ils doivent être câblés ensemble de manière intelligente dans afin de pouvoir leur faire faire quelque chose d'utile. Quoi nous allons faire ici est de montrer deux choses simples qui se font plusieurs fois à l'intérieur de n'importe quel ordinateur.

Ce premier est très simple. Prenez la porte au-dessus, et prenez les deux fils d'entrée, 'a' et 'b', et attachez-les ensemble. Ainsi 'a' et 'b' seront toujours les mêmes. Ils peuvent toujours être activés et désactivés, mais 'a' et 'b' peuvent jamais être différent. 'A' et 'b' peuvent être activés tous les deux, ou les deux soient éteints. Ainsi le graphique de cette combinaison uniquement a deux lignes dessus, deux possibilités:



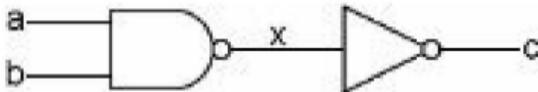
un	b	c
À l'arrêt	À l'arrêt	Sur
Sur	Sur	À l'arrêt

En fait, puisque les colonnes 'a' et 'b' sont identiques, il y a est vraiment une seule entrée et il peut être dessiné simplement comme ceci avec un triangle au lieu de la forme en « D ». Son tableau est aussi très simple :



un	c
À l'arrêt	Sur
Sur	À l'arrêt

Pour notre deuxième variation, combinons l'un de nos type de portail d'origine avec le nouveau portail que nous venons inventé, comme ceci :



Et nous combinerons les graphiques de leur fonctionnement. Le 'A', 'b' et 'x' sont comme la première porte, le 'x' et 'c' sont comme la deuxième porte.

un	b	x	c
À l'arrêt	À l'arrêt	Sur	À l'arrêt
À l'arrêt	Sur	Sur	À l'arrêt
Sur	À l'arrêt	Sur	À l'arrêt
Sur	Sur	À l'arrêt	Sur

Cette combinaison est si souvent utilisée à l'intérieur des ordinateurs, que il est construit comme une seule unité, et le bit 'x' n'est pas disponible pour se connecter. Alors pour simplifier comprendre, il est dessiné comme une seule unité comme ceci:



La seule différence entre cette photo et la photo de notre porte d'origine est que le petit cercle après le le grand 'D' manque.

Puisque 'x' n'est pas utilisé, le graphique peut également être simplifié, et ça ressemble à ça :

a	b	c
À l'arrêt	À l'arrêt	À l'arrêt
À l'arrêt	Sur	À l'arrêt
Sur	À l'arrêt	À l'arrêt
Sur	Sur	Sur

La seule différence entre ce graphique et le graphique de notre porte d'origine est que chaque élément de la colonne 'c' est l'opposé de ce qu'il était dans le tableau original.

Imaginez que cette combinaison de portes ait été installée dans cette pièce avec les deux interrupteurs et le plafond lumière. La seule façon dont la lumière pourrait être allumée est si les deux les interrupteurs étaient allumés. Donc, si vous êtes entré là-bas et que vous avez vu le lumière allumée, puis regardé les interrupteurs, vous verriez qu'ils étaient tous les deux allumés. Peu importe quel commutateur vous décidé était pour la lumière, et vous l'avez éteint, le la lumière s'éteindrait. Vous ne remarquerez peut-être pas que si vous éteint les deux, puis a voulu rallumer la lumière allumé, vous ne pourriez pas le faire en retournant simplement un changer. Tu devrais passer par le même expérience, en basculant les deux interrupteurs jusqu'à ce que la lumière apparaisse allumé, et vous constaterez qu'un interrupteur et l'autre l'interrupteur doit être allumé pour que la lumière s'allume.

Cette porte combinée pourrait être décrite de la manière suivante : pour que la sortie soit activée, une entrée ET l'autre entrée doivent toutes deux être activées. Ainsi ce type de porte a un nom, et dans la tradition de la terminologie informelle inventée par les informaticiens, car elle rappelle ce que signifie le mot ET, on l'appelle simplement « porte ET ».

Maintenant, pour compléter quelques détails délibérément omis ci-dessus, la porte d'origine que nous avons examinée fonctionne comme la porte ET sauf que la sortie est l'opposé, ou le négatif de la porte ET. Ainsi, on l'appelle une porte ET négative, ou simplement une «porte NAND» en abrégé.

La porte simple qui avait les deux entrées liées ensemble a également son propre nom. La sortie est toujours l'opposé d'une entrée, c'est-à-dire que si l'entrée est activée, la sortie n'est pas activée (désactivée). Si l'entrée est désactivée, la sortie n'est pas désactivée (activée). La sortie n'est toujours PAS ce l'entrée est donc appelée une "porte NON".

Remarquez la différence entre les schémas de la porte ET et de la porte NAND. Ce sont les mêmes sauf qu'il y a un petit cercle au début de la sortie de la porte NAND. La chose qui ressemble à une grande lettre 'D' signifie faire la fonction 'AND', ce qui signifie n'agir que si les deux entrées sont activées, et le petit cercle signifie passer à l'opposé. Ainsi, une porte ET est activée si les deux entrées sont activées, une porte NAND est désactivée si les deux entrées sont activées. La porte NOT commence par un triangle, ce qui signifie simplement prendre l'entrée et la transformer en sortie. Le cercle signifie alors basculer vers l'opposé.

La porte ET est beaucoup utilisée dans les ordinateurs, et c'est probablement la plus facile à comprendre, mais nous avons d'abord regardé la porte NAND pour deux raisons. La première raison, et la moins importante, est que la porte NAND est la porte la plus facile à construire. Lorsque vous devez construire un grand nombre de portails, il sera moins cher et plus fiable si vous pouvez utiliser le type de portail le plus facile à construire.

La deuxième raison, et très importante, pour laquelle nous avons d'abord examiné la porte NAND est la suivante : tout ce qui, dans un ordinateur, en fait un ordinateur, peut être constitué d'une ou plusieurs portes NAND. Nous avons déjà vu que la porte NON et la porte ET peuvent être constituées de portes NAND, et nous verrons quelques combinaisons plus intéressantes au fur et à mesure. Mais chacun d'eux est basé sur cela

petite chose idiote appelée une porte NAND.

Le problème dans ce chapitre a été que la porte NAND est le bloc de construction de base des ordinateurs, mais la porte ET est la première porte qui a un nom qui a du sens.

Nous avons donc d'abord regardé la porte NAND et la porte NOT sans leur donner de noms. Ensuite, nous avons construit une porte ET, lui avons donné son nom, et sommes revenus et avons nommé les deux premiers.

Comme note sur la langue ici, le mot «et» est une conjonction en anglais régulier. Il relie deux choses, comme dans "J'aime les pois et les carottes". En informatique, nous utilisons le mot de deux manières nouvelles. Premièrement, c'est un adjectif, un mot qui modifie un nom. Lorsque nous disons "c'est une porte ET", le mot "porte" est un nom, et le mot "ET" nous dit de quel type de porte il s'agit. C'est ainsi que "ET" a été utilisé dans ce chapitre. "ET" sera également utilisé comme verbe, comme dans "let us AND these two bits". Nous verrons ET utilisé de cette manière plus loin dans le livre.

Revenons donc au thème de la simplicité de ce livre, nous avons dit qu'il n'y a qu'une seule chose dans les ordinateurs, les bits.

Et maintenant, nous voyons que les bits sont construits à l'aide de portes, et toutes les portes se résument à la porte NAND. Donc, tout ce que vous devez savoir pour comprendre les ordinateurs, c'est ce dispositif très simple, la porte NAND. Sans blague! Pouvez-vous comprendre cette chose? Ensuite, vous pouvez comprendre l'ensemble de l'ordinateur.

Diagrammes

Si vous voulez voir comment fonctionne une machine mécanique, la meilleure façon de le faire est de regarder à l'intérieur de celle-ci, de regarder les pièces bouger pendant qu'elle fonctionne, de la démonter, etc. La deuxième meilleure façon est de l'étudier à partir d'un livre qui a beaucoup de photos montrant les pièces et comment elles interagissent.

Un ordinateur est aussi une machine, mais la seule chose qui bouge à l'intérieur est l'électricité invisible et silencieuse. C'est très ennuyeux de regarder l'intérieur d'un ordinateur, on dirait que rien ne se passe du tout.

La construction proprement dite des différentes parties d'un ordinateur est un sujet très intéressant, mais nous n'allons pas le couvrir plus loin que de dire ce qui suit : la technique commence avec une fine tranche de cristal, et en une série d'étapes, il est soumis à divers produits chimiques, procédés photographiques, chaleur et métal vaporisé. Le résultat est quelque chose appelé une « puce », qui a des millions de composants électroniques construits sur sa surface. Le processus comprend la connexion des pièces en portes et la connexion des portes en sections informatiques complètes.

La puce est ensuite enfermée dans un morceau de plastique d'où sortent des broches. Plusieurs d'entre eux sont branchés sur une carte, et là vous avez un ordinateur. L'ordinateur que nous allons "construire" dans ce livre pourrait facilement tenir sur une puce de moins d'un quart de pouce carré.

Mais le fait est que, contrairement à une machine mécanique, la structure réelle d'une puce est très encombrée et difficile à suivre, et vous ne pouvez de toute façon pas voir l'électricité. Les schémas que nous avons vus dans le chapitre précédent sont le meilleur moyen de montrer comment fonctionne un ordinateur, nous ferions donc mieux de bien les lire.

Dans la suite de ce livre, nous allons construire de nouvelles pièces en connectant plusieurs portes entre elles. Nous allons décrire ce que fait la nouvelle pièce, puis lui donner un nom et son propre symbole. Ensuite, nous pouvons connecter plusieurs de ces nouvelles parties en quelque chose d'autre qui reçoit également un nom et un symbole. Avant que vous ne le sachiez, nous aurons assemblé un ordinateur complet.

Chaque fois qu'il y a un nouveau diagramme, le texte expliquera

quel est son but et comment les pièces y parviennent, mais le lecteur doit vraiment regarder le diagramme jusqu'à ce qu'il puisse voir que les portes font réellement ce que le livre dit qu'elles feront. Si cela est fait fidèlement avec chacun, vous verrez très bientôt exactement comment fonctionne un ordinateur.

Il n'y a que deux choses dans nos dessins, il y a des pièces qui ont des entrées et des sorties, et il y a des lignes, ou des fils, qui relient les sorties et les entrées ensemble.

Lorsque l'électricité sort de la sortie d'une porte, l'électricité parcourt tout le fil aussi vite qu'elle peut aller. Si la sortie d'une porte est allumée, alors l'électricité est allumée dans le fil qui y est connecté, aussi loin qu'il aille. Si la sortie d'une porte est désactivée, tout le fil est désactivé. Je suppose que vous pourriez considérer que le morceau qui sort de la porte comprend également tout le fil.

Les entrées des portes n'utilisent pas l'électricité dans le fil, donc une sortie peut être connectée à l'entrée d'une ou plusieurs portes.

Lorsque les fils sont connectés ensemble, cela est indiqué par un point où ils se rencontrent sur le schéma, et tous les fils qui sont connectés ensemble reçoivent de l'électricité comme s'ils n'étaient qu'un seul fil. Lorsque des fils se croisent sur un schéma sans point, cela signifie qu'il n'y a pas de connexion entre eux, ils ne se touchent pas, les deux bits sont séparés.

Chaque fois qu'il y a un choix, les diagrammes montreront le chemin de l'électricité se déplaçant de gauche à droite, ou du haut de la page vers le bas. Cependant, il y aura de nombreuses exceptions à cela, surtout plus tard dans le livre. Mais vous pouvez toujours dire dans quel sens l'électricité se déplace dans un fil en commençant par une sortie et en la suivant jusqu'à une entrée.

La plupart des schémas du livre sont très faciles à suivre. Dans quelques cas, il y aura également un de ces graphiques qui montre ce que sera la sortie pour chaque combinaison possible d'entrées. Si vous avez du mal à suivre un diagramme, vous pouvez écrire les tenants et les aboutissants directement sur la page, ou placer des pièces sur la page et les retourner de sorte que pile signifie activé et pile signifie désactivé.

Malheureusement, le schéma du chapitre suivant est

probablement le plus difficile à suivre de tout le livre, mais une fois que vous le maîtriserez, vous serez un lecteur de diagramme expert.

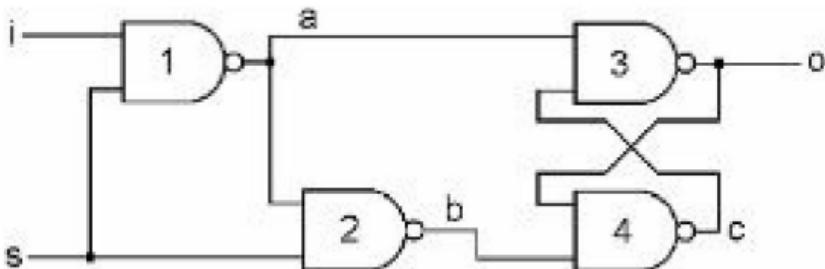
Rappelez-vous quand

Vous avez probablement entendu parler de la mémoire informatique, et maintenant nous allons voir exactement ce que c'est. Étant donné que la seule chose à l'intérieur des ordinateurs, ce sont des bits, et que la seule chose qui arrive aux bits est qu'ils s'allument ou s'éteignent, il s'ensuit que la seule chose qu'un ordinateur peut "se souvenir" est de savoir si un bit était allumé ou éteint. Nous allons maintenant voir comment cela se fait.

Le schéma suivant montre un bit de mémoire d'ordinateur.

Il se trouve que c'est l'une des astuces les plus intéressantes que vous puissiez faire avec quelques portes. Nous examinerons longuement comment cela fonctionne ici, et après l'avoir compris, nous le remplacerons par son propre symbole et l'utiliserons comme élément de base pour des choses plus grandes et meilleures.

Il est composé de seulement quatre portes NAND, mais son câblage est un peu spécial. C'est ici:



Cette combinaison dans son ensemble a deux entrées et une sortie. 'i' est l'endroit où nous entrons le bit dont nous voulons nous souvenir, et 'o' est la sortie du bit mémorisé.

'S' est une entrée qui indique à ces portes quand 'définir' la mémoire. Il y a aussi trois fils internes étiquetés « a », « b » et « c » que nous devrons examiner pour voir comment ces pièces fonctionnent ensemble. Essayez de suivre cela attentivement, une fois que vous verrez que cela fonctionne, vous comprendrez l'une des choses les plus importantes et les plus couramment utilisées dans un ordinateur.

Pour voir comment cela fonctionne, commencez par 's' activé et 'i' désactivé.

Puisque 'i' et 's' vont dans la porte 1, une entrée est désactivée, donc 'a' sera activé. Puisque 'a' et 's' vont à la porte 2, les deux

entrées sont activées, et donc 'b' sera désactivé. En regardant la porte 4, puisque 'b' est désactivé, la sortie de la porte 4, 'c' sera activée. Puisque 'c' et 'a' sont tous les deux activés, la sortie de la porte 3, 'o' sera désactivée. « O » redescend à la porte 4 fournissant une deuxième entrée désactivée, laissant « c » toujours allumé. La chose importante à noter ici est qu'avec 's' activé, 'o' finit par être le même que 'l'. Maintenant, avec 's' toujours activé, changeons 'l' en activé. Puisque 'l' et 's' vont dans la porte 1, 'a' sera éteint. 'A' va d'un côté à la fois de la porte 2 et de la porte 3, donc leurs sorties 'o' et 'b' doivent toutes les deux être allumées. 'O' et 'b' vont tous les deux dans la porte 4 et éteignent 'c', qui remonte à la porte 3 en lui fournissant une seconde entrée éteinte, laissant 'o' toujours allumé. La chose importante à noter ici est la même chose que nous avons notée dans le paragraphe précédent - qu'avec 's' activé, 'o' finit par être le même que 'l'. Jusqu'à présent, nous avons vu que lorsque 's' est activé, vous pouvez activer et désactiver 'l', et 'o' changera avec lui. 'O' s'allumera et s'éteindra exactement comme 'l'. Avec 's' activé, cette combinaison n'est pas plus utile qu'un fil reliant 'l' à 'o'.

Voyons maintenant ce qui se passe lorsque nous désactivons "s". Regardez la porte 1. Quand 's' est éteint, 'a' sera allumé quoi que vous fassiez avec 'l'. Maintenant, vous pouvez activer et désactiver 'l' et rien ne se passera. Il en va de même pour la porte 2. 'A' peut être allumé, mais 's' est éteint, donc 'b' ne peut être que allumé. 'a' et 'b' sont activés, et changer 'l' ne fait rien. Maintenant, la seule chose qui compte, la grande question est, que sera le "o" ?

Si 'l' et 'o' étaient activés avant que 's' ne soit désactivé, la porte 3 avait les deux entrées désactivées et la porte 4 avait les deux entrées activées. Lorsque 's' s'éteint, 'a' s'allume, ce qui est une entrée de la porte 3. Mais l'autre entrée est éteinte, donc rien ne change, 'o' reste allumé.

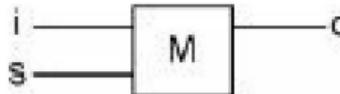
Si 'l' et 'o' étaient désactivés avant que 's' ne soit désactivé, la porte 3 avait les deux entrées activées et la porte 4 avait les deux entrées désactivées. Lorsque 's' s'éteint, 'b' s'allume, ce qui est une entrée de la porte 4. Mais l'autre entrée est éteinte, donc rien ne change, 'c' reste allumé et 'o' reste éteint.

Ainsi, la réponse à la question de savoir ce qui arrive à 'o' lorsque 's' est désactivé, est qu'il reste tel qu'il était et qu'il n'est plus affecté par 'l'.

Maintenant, qu'avons-nous ici? Avec 's' activé, 'o' fait tout ce que 'i' fait. Avec 's' désactivé, 'o' reste tel quel et 'i' étaient, au dernier instant juste avant que 's' ne disparaisse. Maintenant, 'i' peut changer, mais 'o' reste tel qu'il était. Cette combinaison de portes verrouille la façon dont « je » était à une époque antérieure. C'est ainsi qu'une combinaison de quatre portes NAND peut "se souvenir". Ce n'est qu'un bit de mémoire, mais c'est le bloc de construction de base de toute mémoire informatique. Tout ce que la mémoire informatique est, c'est un moyen de préserver la façon dont un bit a été défini à un moment donné.

J'espère que vous avez suivi les fils et les marches et arrêts dans ce chapitre. Une fois que vous aurez vu exactement comment cela fonctionne, vous saurez que ces simples portes NAND peuvent créer un bit de mémoire, et je vous assure que vous ne vous poserez plus jamais de questions à ce sujet.

Maintenant que nous savons comment cette chose fonctionne, nous n'avons plus besoin de regarder ce câblage interne délicat de cette combinaison. Nous avons vu comment cela fonctionne, et à partir de maintenant, nous nous contenterons d'utiliser ce schéma pour le représenter :



'I' est le bit d'entrée que vous souhaitez enregistrer. 'S' est l'entrée qui autorise 'i' dans le bit de mémoire lorsque 's' est activé, et le verrouille en place ou le 'définit' lorsque 's' s'éteint. 'O' est la sortie des données actuelles ou sauvegardées. 'M' signifie Mémoire. Assez simple, hein ?

Revenons à notre chambre avec les drôles d'interrupteurs.

Il y avait une porte NAND connectée. Retirons la porte NAND et remplaçons-la par ce nouveau bit de mémoire. Nous connecterons l'interrupteur de gauche au fil « i », l'interrupteur de droite au fil « s » et le plafonnier au fil « o ». Nous pourrions commencer avec tout se ressemblant, c'est-à-dire que la lumière est allumée, mais les deux interrupteurs sont éteints. Cela signifierait qu'à un moment donné dans le passé, 'i' et 's' étaient activés, et 's' a été désactivé en premier, verrouillant l'état de 'i' dans notre bit de mémoire, qui sort ensuite à 'o.' Alors 'i' aurait pu être éteint sans rien affecter. Donc, si nous entrions et décidions que nous voulions éteindre la lumière, nous essaierions d'abord l'interrupteur "i", l'allumerions et l'éteindrions, et

rien ne se passerait. Ensuite, nous essaierions le commutateur 's'.

Lorsque nous l'allumions, la lumière s'éteignait. Aha on dit, l'interrupteur en 's' contrôle la lumière, mais il est installé à l'envers ! Ensuite, nous éteignons l'interrupteur "s", en nous attendant à ce que la lumière se rallume, mais la lumière reste éteinte. Maintenant, les interrupteurs sont dans la même position que lorsque nous sommes entrés dans la pièce, ils sont tous les deux éteints, mais maintenant la lumière est également éteinte, c'est déroutant.

Maintenant, je ne veux pas spéculer sur la quantité de jurons qui se passeraient avant que quelqu'un ne comprenne cela, mais à la fin, ils découvriraient que lorsque 's' était allumé, la lumière s'allumait et s'éteignait avec 'l', et quand ' s'était éteint, la lumière restait telle qu'elle était juste avant que 's' ne s'éteigne.

Que pouvons-nous faire avec un bit ?

Maintenant, nous avons décris un peu, nous avons montré comment en construire un, comment se rappeler au fil du temps dans quel état se trouvait un bit à un instant antérieur, et maintenant ? Qu'est-ce qu'on en fait ?

Puisqu'un bit n'est en fait rien de plus que l'électricité allumée ou éteinte, la seule chose réelle et réelle que nous puissions faire avec un bit est d'allumer ou d'éteindre les lumières, les grille-pain ou autre chose.

Mais nous pouvons aussi utiliser un peu pour représenter quelque chose d'autre dans nos vies. Nous pouvons en prendre un peu, et le connecter à un feu rouge, et dire que quand ce bit est allumé, cela signifie s'arrêter, et quand ce bit est éteint, vous pouvez partir. Ou si un morceau particulier est allumé, vous voulez des frites avec votre hamburger ; s'il est éteint, vous ne voulez que le hamburger.

C'est l'action d'utiliser un code. Qu'est-ce qu'un code ? Un code est quelque chose qui vous dit ce que quelque chose d'autre signifie. Quand quelque chose est censé signifier quelque chose, quelque part quelqu'un doit faire une liste de tous les états de la « chose », et les significations associées à chacun de ces états. Quand il s'agit d'un bit, puisqu'il ne peut être que dans deux états différents, alors un bit ne peut signifier qu'une des deux choses. Un code pour un bit n'aurait besoin que de deux significations, et l'une de ces significations serait associée au bit désactivé, et l'autre signification serait associée au bit activé.

C'est ainsi que vous attribuez un sens à un bit. Le bit ne contient aucune signification en soi ; il n'y a pas de place dans un bit pour autre chose que la présence ou l'absence d'électricité. La signification est attribuée à un bit par quelque chose d'extérieur au bit. Il n'y a rien sur la circulation ou les frites dans un bout, on dit juste que pour ce bout à cet endroit, relié à un feu rouge suspendu au-dessus d'une intersection, quand c'est allumé, il faut s'arrêter, quand c'est éteint, on peut aller. Un autre bit, dans une caisse enregistreuse d'un fast-food, signifie mettre des frites dans le sac quand le bit est allumé, ou pas de frites quand il est éteint.

Ce sont deux cas où quelqu'un a inventé un simple code à deux éléments. Dans un cas, le code est : bit on veut dire frites,

un peu éteint signifie pas de frites, dans l'autre cas, un peu éteint signifie aller, un peu allumé signifie s'arrêter. Ces deux bits sont identiques, ils sont simplement utilisés à des fins différentes, et quelqu'un décide de la signification de ces deux bits. Le code est écrit quelque part dans les livres de droit ou dans le manuel du gérant du restaurant, mais le code n'est pas dans le bit. L'état du bit indique simplement à quelqu'un quelle ligne du code il est censé croire vraie à l'instant présent. C'est ce qu'est un code.

Comme les espions qui transmettent des messages en utilisant un code secret, le message peut être vu par d'autres personnes, mais ces autres personnes n'ont pas le code, donc elles ne savent pas ce que signifie le message. Peut-être qu'un espion a un pot de fleurs posé sur le rebord de la fenêtre de son appartement. Lorsque le pot est sur le côté gauche du seuil, cela signifie "Rejoignez-moi à la gare à 1h30". Et quand le pot de fleurs est sur le côté droit du seuil, cela signifie "Pas de réunion aujourd'hui". Chaque jour, l'autre espion marche dans la rue et lève les yeux vers cette fenêtre pour voir s'il doit aller à la gare aujourd'hui. Tous les autres qui marchent dans cette rue peuvent tout aussi bien voir ce message, mais ils n'ont pas le code, donc cela ne signifie rien pour eux. Ensuite, lorsque les deux espions se rencontrent, ils peuvent passer un morceau de papier qui est écrit dans un autre code secret.

Ils encodent et décodent le message à l'aide d'un livre de codes qu'ils ne portent pas lors de leur rencontre. Donc, si leur message est intercepté par quelqu'un d'autre, cela ne signifiera rien pour ce quelqu'un d'autre. Quelqu'un qui n'a pas le livre de codes n'aura pas la bonne signification des symboles sur la feuille de papier.

Un bit d'ordinateur n'est encore et ne sera toujours rien de plus qu'un endroit où il y a ou non de l'électricité, mais quand nous, en tant que société d'êtres humains, utilisons un bit dans un certain but, nous donnons un sens au bit. Lorsque nous connectons un morceau à un feu rouge et que nous l'accrochons au-dessus d'une intersection, et que nous demandons aux gens d'étudier les manuels du conducteur avant de leur donner des permis de conduire, nous avons donné un sens à ce morceau. Le rouge signifie « arrêter », non pas parce que le mors est capable de faire quoi que ce soit à un véhicule circulant sur la route, mais parce que nous, en tant que personnes, convenons que le rouge signifie arrêter, et nous, voyant ce mors allumé, arrêterons notre voiture afin d'éviter être renversé par une voiture circulant dans la rue transversale, et nous espérons que tout le monde le fera

de même afin que nous soyons assurés que personne ne nous frappera lorsque ce sera à notre tour de traverser l'intersection.

Il y a donc beaucoup de choses qui peuvent être faites avec un peu. Il peut indiquer vrai ou faux, aller ou s'arrêter. Un simple oui ou non peut être une chose majeure, comme dans la réponse à "Veux-tu m'épouser?" ou une question de tous les jours comme "Voulez-vous des frites avec ça?"

Mais encore, il y a beaucoup de choses qui ne peuvent pas être faites avec un bit, ou qui semblent totalement incompatibles avec l'idée de bits. Il peut y avoir de nombreux exemples de choses oui/non dans la vie de tous les jours, mais il y a beaucoup plus de choses qui ne sont pas un simple oui ou non.

Dans le cas du télégraphe, qui n'était incontestablement qu'un bit, comment peut-il y avoir plus de deux éléments dans le code Morse ? La réponse est que la capacité d'envoyer et de recevoir des messages dépendait des compétences et de la mémoire des opérateurs aux deux extrémités du fil. Dans le code Morse, si la touche était enfoncée pendant très peu de temps, cela s'appelait un « point (.) », et si elle était enfoncée pendant un temps légèrement plus long, cela s'appelait un « tiret (-) ».

Chaque lettre de l'alphabet se voyait attribuer une combinaison unique de points et/ou de tirets, et les deux opérateurs étudiaient le code, le mémorisaient et s'entraînaient à l'utiliser.

Par exemple, le code de la lettre 'N' était tiret point (-.) et le code de la lettre 'C' était tiret point tiret point (-.-.). La longueur des temps d'activation était différente pour faire des points et des tirets, et la durée des temps d'arrêt était différente pour faire la distinction entre le temps qui sépare les points et les tirets dans une lettre, le temps qui sépare les lettres et le temps qui sépare les mots.

Vous avez besoin d'un temps d'arrêt plus long pour éviter de confondre un 'C' avec deux 'N'. La personne qui les recevait devait les reconnaître comme des modèles - c'est-à-dire qu'elle devait entendre et se souvenir de la durée de plusieurs heures d'allumage et d'extinction jusqu'à ce qu'elle reconnaisse une lettre. L'appareil télégraphique n'avait aucune mémoire, il n'y avait même jamais une lettre entière sur le fil à la fois, les morceaux de lettres descendaient le fil, pour être assemblés en points et en tirets dans l'esprit de l'opérateur, puis en lettres, puis en mots et phrases écrits sur une feuille de papier. Ainsi, le bit télégraphique atteint plus de deux significations en ayant plusieurs moments individuels où il peut y avoir des ons ou

offs.

Si un ordinateur était construit sur les principes du code Morse, il n'y aurait qu'une ampoule au-dessus qui nous ferait clignoter le code. Puisque nous préférions voir des lettres entières, des mots et des phrases à l'écran simultanément, nous avons besoin de quelque chose de plus qu'un simple bit et de cet ancien code.

Même dans les exemples utilisés dans ce chapitre, les vrais feux de circulation ont en fait trois bits, un pour le rouge, un pour le jaune et un pour le vert. Si vous n'aviez qu'un seul bit, vous pourriez simplement avoir un feu rouge à l'intersection, et quand il était allumé, cela signifierait s'arrêter, et quand il était éteint, cela signifierait partir. Mais quand il était éteint, vous pourriez vous demander s'il était vraiment éteint ou si l'ampoule venait de griller. Donc, utiliser trois bits est beaucoup plus utile dans ce cas.

Dans le monde réel, nous avons déjà vu que les ordinateurs peuvent contenir des lettres, des mots, des phrases, des livres entiers, ainsi que des chiffres, des images, des sons et plus encore. Et pourtant, tout cela ne se résume qu'à des morceaux.

Si nous voulons que la mémoire de notre ordinateur puisse contenir plus d'un on ou off, ou oui ou non, nous devrons avoir quelque chose de plus qu'un simple bit. Heureusement, nous pouvons faire quelque chose de beaucoup plus utile simplement en utilisant plusieurs bits ensemble, puis en créant un code (ou peut-être plusieurs codes) pour leur attribuer une signification utile.

Une rose sous un autre nom

Avant de poursuivre, nous allons introduire un changement dans ce que nous appelons quelque chose. Comme nous le savons, tous les bits de l'ordinateur sont des endroits où il y a ou n'y a pas, certains électicité. Nous appelons ces états, "on" et "off", et c'est exactement ce qu'ils sont. Même si ce sont des mots courts, il y a des endroits où c'est beaucoup plus facile, plus clair et plus simple d'utiliser un seul symbole pour décrire ces états. Heureusement, nous n'allons pas inventer quelque chose de délicat, nous allons juste utiliser deux symboles que vous connaissez déjà bien, les nombres zéro et un. D'ici sur out, nous annulerons 0, et nous appellerons 1. Et parfois nous utiliserons toujours on et off.

Ainsi, le graphique de notre porte NAND ressemblera à ceci :

a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

C'est très facile à comprendre, bien sûr, mais le point qui doit être fait ici, c'est que l'ordinateur les pièces n'ont pas changé, la seule chose qui a changé est ce que nous, en tant que personnes regardant la machine, sommes l'appelant. Juste parce que nous appelons un peu un zéro ou un, cela ne veut pas dire que des nombres sont soudainement apparus et courrent à l'intérieur de l'ordinateur. Il y a toujours pas de chiffres (ni de mots, ni de sons, ni d'images) dans un ordinateur, seulement des bits, exactement comme décrit précédemment. Nous

aurait pu les appeler plus et moins, oui et non, vrai et faux, pile et face, quelque chose et rien, nord et sud, ou même Bert et Ernie. Mais zéro et un le feront. Il s'agit d'un simple code à deux éléments. Allumé signifie 1 et éteint signifie 0.

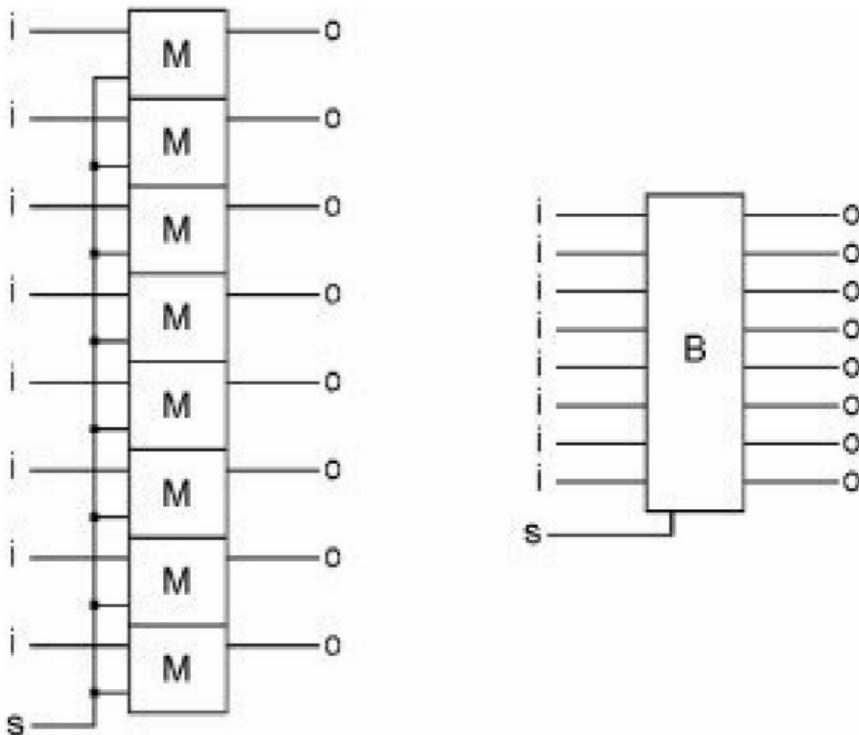
En guise de commentaire ici, il semble y avoir une tendance parmi les fabricants d'appareils électroménagers du monde à remplacer les termes obsolètes et démodés de marche et d'arrêt par les 0 et 1 modernes. Sur de nombreux interrupteurs d'alimentation, ils mettent un 0 à la position d'arrêt , et un 1 par la position on. Le premier endroit où j'ai vu cela était sur un ordinateur personnel, et j'ai pensé que c'était une jolie nouveauté, être sur un ordinateur, mais maintenant cette pratique s'est étendue aux téléphones portables, aux cafetières et aux tableaux de bord des automobiles. Mais je pense que c'est une erreur. Comprenez-vous que le code aurait tout aussi bien pu être défini comme "éteint signifie 1 et allumé signifie 0 ?" L'ordinateur fonctionnerait exactement de la même manière, seule l'impression dans les manuels techniques décrivant ce qui se passe à l'intérieur de l'ordinateur changerait.

Lorsque vous voyez l'un de ces commutateurs 0/1, vous devez le traduire à partir de ce code informatique très couramment utilisé en ce qu'il signifie réellement, activé ou désactivé. Alors pourquoi s'embêter ? Vous ne voulez pas allumer votre machine à café '1', vous voulez qu'elle soit allumée pour pouvoir obtenir votre java et vous réveiller déjà. Imaginez mettre ces symboles sur un gaufrier en 1935. Personne n'aurait eu la moindre idée de ce que cela signifiait. C'est probablement juste pour que les fabricants n'aient pas à faire imprimer des interrupteurs dans différentes langues. Ou peut-être que cette tendance vient d'un désir altruiste d'éduquer le public sur le « fait » moderne qu'un 1 est le même que sur, mais ce n'est pas un fait, c'est un code arbitraire.

Huit c'est assez

Afin de pouvoir représenter quelque chose de plus que de simples questions oui/non, ce que nous allons faire est d'empiler huit bits dans un seul paquet et de les utiliser comme une seule unité. Voici un schéma de la façon dont cela est fait. Nous avons pris huit de nos bits de mémoire, chacun a toujours sa propre entrée de données 'i' et sa propre sortie 'o', mais nous avons câblé les huit entrées 's' ensemble.

Ainsi, lorsque le « s » unique est allumé puis éteint, ces huit « M » capturent les états de leurs « i » correspondants en même temps. L'image de gauche montre les huit 'M', celle de droite est la même chose, juste un peu plus simple.



Cet assemblage a un nom ; on l'appelle un octet, donc le "B" dans le diagramme. Il existe plusieurs explications contradictoires sur l'origine exacte de ce mot, mais comme il sonne comme le mot «mordre», vous pouvez simplement le considérer comme une bouchée entière par rapport à une unité plus petite, un peu. Juste pour vous montrer que les concepteurs d'ordinateurs ont le sens de l'humour, lorsqu'ils utilisent quatre bits comme unité, ils appellent cela un grignotage. Ainsi, vous pouvez manger un tout petit morceau de tarte aux cerises, ou grignoter ou prendre un octet entier.

Quand nous en avions un peu, nous dirions simplement que son état était soit 0 soit 1. Maintenant que nous avons un octet, nous allons écrire le contenu de l'octet comme ceci : 0000 0000, et vous pouvez voir pourquoi nous avons cessé d'utiliser off /sur 0/1. Ce

montre le contenu de chacun des huit bits, dans ce cas ce sont tous des zéros. L'espace au milieu est juste là pour faciliter la lecture. La main gauche 0 ou 1 correspondrait au bit supérieur de notre octet, et le 0 ou 1 le plus à droite représenterait le bit inférieur.

Comme vous le savez mieux maintenant, un bit a deux états possibles dans lesquels il peut être - activé ou désactivé. Si vous avez deux bits, il y a quatre états possibles dans lesquels ces deux bits peuvent se trouver. Vous souvenez-vous du tableau que nous avons dessiné pour les entrées de la porte NAND ? Il y avait quatre lignes sur le graphique, une pour chaque combinaison possible des deux bits d'entrée de la porte, 0-0, 0-1, 1-0 et 1-1.

Notez que l'ordre des bits est important - c'est-à-dire que si vous regardez deux bits et demandez seulement combien de bits sont activés, il n'y a que trois possibilités : aucun bit activé, un bit activé ou deux bits activés. Ce serait appeler les combinaisons 1-0 et 0-1 la même chose. Dans le but d'utiliser plusieurs bits pour implémenter un code, nous nous soucions définitivement de l'ordre des bits dans un octet. Lorsqu'il y a deux bits, nous voulons utiliser les quatre possibilités, nous devons donc garder les bits dans l'ordre.

Combien y a-t-il de possibilités différentes lorsque vous utilisez huit bits ? Si tout ce que vous avez est un bit, il peut être dans l'un des deux états. Si vous ajoutez un deuxième bit, la paire a deux fois plus d'états qu'auparavant car l'ancien bit a ses deux états tandis que le nouveau bit est à sens unique, puis l'ancien bit a ses deux états tandis que le nouveau bit est le

autrement. Donc deux bits ont quatre états. Lorsque vous ajoutez un troisième bit, les deux premiers ont quatre états avec le nouveau bit désactivé et quatre états avec le nouveau bit activé, pour un total de huit états. Chaque fois que vous ajoutez un bit, vous doublez simplement le nombre d'états possibles. Quatre bits ont 16 états, cinq en ont 32, six en ont 64, sept en ont 128, huit en ont 256, neuf en ont 512, etc.

Nous allons prendre huit bits et l'appeler un octet.

Puisqu'un bit est une chose qui a un emplacement dans l'espace, qui peut être dans l'un des deux états, alors un octet est une chose qui a huit emplacements distincts dans l'espace, chacun pouvant être activé ou désactivé, qui sont conservés dans le même ordre.

L'octet, pris dans son ensemble, est un emplacement dans l'espace qui peut être dans l'un des 256 états à un moment donné, et peut être amené à changer d'état au fil du temps.

Codes

Un peu ne pouvait représenter que des types de choses oui/non, mais maintenant que nous avons 256 possibilités, nous pouvons rechercher des choses dans nos vies qui sont légèrement plus compliquées.

L'une des premières choses qui pourraient correspondre au projet de loi est la langue écrite. Si vous regardez dans un livre et que vous voyez tous les différents types de symboles utilisés pour imprimer le livre, vous verrez les 26 lettres de l'alphabet en majuscules et en minuscules. Ensuite, il y a les chiffres de 0 à 9, et il y a des signes de ponctuation comme des points, des virgules, des guillemets, des points d'interrogation, des parenthèses et plusieurs autres. Ensuite, il y a des symboles spéciaux comme le signe "arobase" (@.) la devise (\$.) et plus encore.

Si vous additionnez ces 52 lettres, 10 chiffres, quelques dizaines pour la ponctuation et les symboles, vous obtenez quelque chose comme 100 symboles différents qui peuvent apparaître imprimés sur les pages d'un livre moyen.

À partir de maintenant, nous utiliserons le mot « caractère » pour désigner l'une de ces sortes de choses, l'une des lettres, des chiffres ou d'autres symboles qui sont utilisés dans la langue écrite. Un caractère peut être une lettre, un chiffre, un signe de ponctuation ou tout autre type de symbole.

Nous avons donc un langage écrit avec environ 100 caractères différents, et notre octet avec 256 possibilités, peut-être pouvons-nous représenter le langage avec des octets. Voyons, comment mettez-vous un 'A' dans un octet ? Il n'y a rien d'inherent à un octet qui l'associerait à un caractère, et il n'y a rien d'inherent à un caractère qui ait quoi que ce soit à voir avec des bits ou des octets. L'octet ne contient pas de formes ou d'images. Diviser un caractère en huit parties ne trouve aucun bit.

La réponse, comme précédemment, consiste à utiliser un code pour associer l'un des états possibles de l'octet à quelque chose qui existe dans le monde réel. La lettre « A » sera représentée par un modèle particulier de 1 et de 0 dans les bits d'un octet. L'octet a 256 états possibles différents, donc quelqu'un doit s'asseoir avec un crayon et du papier et énumérer les 256 de ces combinaisons, et à côté de chacune, mettre l'un des caractères qu'il veut que ce modèle représente. Bien sûr, au moment

Ainsi, au tout début de l'informatique, chaque fabricant s'est assis et a inventé un code pour représenter langue écrite. À un moment donné, les différentes sociétés rendu compte qu'il serait avantageux qu'ils utilisent tous le même code, au cas où ils voudraient un jour celui de leur entreprise ordinateurs pour pouvoir communiquer avec une autre marque. Ils ont donc formé des comités, tenu des réunions et fait tout ce qu'ils devaient faire d'autre pour trouver un code sur lesquels ils pourraient tous s'entendre.

Il existe plusieurs versions de ce code conçu pour objectifs différents, et ils tiennent encore des réunions aujourd'hui pour conclure des accords sur divers détails ésotériques de choses. Mais nous n'avons pas besoin de nous préoccuper de tout pour voir comment fonctionne un ordinateur. Le code de base qu'il est encore en usage aujourd'hui, et je ne connais pas aucune raison pour laquelle il aurait jamais besoin d'être changé.

Le code a un nom, c'est le : American Standard Code pour l'échange d'informations. Ceci est généralement abrégé en ASCII, prononcé "aass-key". Nous n'avons pas besoin d'imprimer tout le code ici, mais voici un exemple. Ce sont 20 des codes qu'ils ont trouvés, les 10 premiers lettres de l'alphabet en majuscules et minuscules :

PARTIE DU TABLEAU DES CODES ASCII

UN	0100 0001	un	0110 0001
B	0100 0010	b	0110 0010
C	0100 0011	c	0110 0011
ré	0100 0100	ré	0110 0100
et	0100 0101	et	0110 0101
F	0100 0110	F	0110 0110
g	0100 0111	g	0110 0111
H	0100 1000	h	0110 1000

je	0100 1001	je	0110 1001
J	0100 1010	j	0110 1010

Chaque code est unique. Il est intéressant de noter la façon dont ils ont arrangé les codes de sorte que les codes pour les majuscules et les minuscules d'une même lettre utilisent le même code à l'exception d'un bit. Le troisième bit à partir de la gauche est désactivé pour toutes les lettres majuscules et activé pour toutes les lettres minuscules.

Si vous vouliez mettre un message sur l'écran de votre ordinateur disant "Bonjour Joe", vous auriez besoin de neuf octets. Le premier octet aurait le code pour la majuscule "H", le deuxième octet aurait le code pour la minuscule "e", les troisième et quatrième octets auraient le code pour la minuscule "l", le cinquième octet aurait le code pour la minuscule "o", le sixième octet aurait le code d'un espace vide, et les octets sept, huit et neuf contiendraient les codes pour "J", "o" et "e".

Notez qu'il existe même un code pour un espace vide (c'est 0010 0000 en passant.) Vous vous demandez peut-être pourquoi il doit y avoir un code pour un espace vide, mais cela vous montre à quel point les ordinateurs sont stupides. Ils ne contiennent pas vraiment de phrases ou de mots, il y a juste un certain nombre d'octets définis avec les codes de la table de codes ASCII qui représentent les symboles individuels que nous utilisons dans le langage écrit. Et l'un de ces « symboles » est l'absence de tout symbole, appelé espace, que nous utilisons pour séparer les mots.

Cet espace nous dit, le lecteur, que c'est la fin d'un mot et le début d'un autre. L'ordinateur n'a que des octets, chacun pouvant être dans l'un de ses 256 états. Dans quel état se trouve actuellement un octet, cela ne signifie rien pour l'ordinateur.

Prenons donc un octet de mémoire et réglons les bits sur 0100 0101. Cela signifie que nous avons mis la lettre E dans l'octet, n'est-ce pas ? Eh bien pas vraiment. Nous avons défini le modèle qui apparaît à côté de la lettre E dans la table des codes ASCII, mais il n'y a rien d'inherent dans l'octet qui a à voir avec un "E". Si Thomas Edison avait testé huit de ses nouvelles ampoules expérimentales et les avait placées en rangée sur une étagère, et que les première, troisième, quatrième, cinquième et septième ampoules avaient grillé, les ampoules restantes seraient un octet avec ça

motif. Mais il n'y avait pas une seule personne sur la face de la Terre qui aurait regardé cette rangée d'ampoules et pensé à la lettre « E », car l'ASCII n'avait pas encore été inventé. La lettre est représentée par le code.

La seule chose dans l'octet est le code.

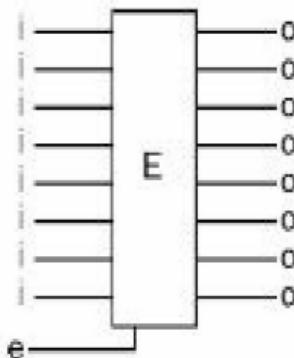
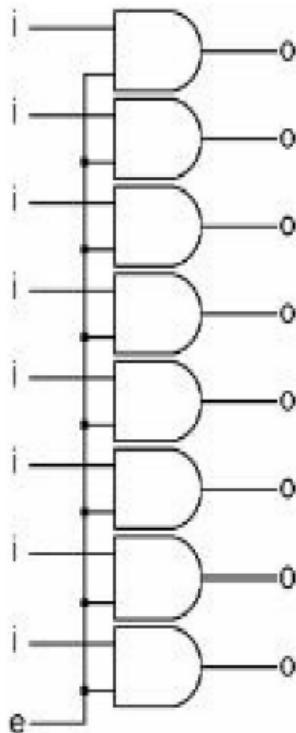
Voilà le sujet des codes. Un code informatique est quelque chose qui vous permet d'associer chacun des 256 motifs possibles d'un octet à quelque chose d'autre.

Une autre note de langage ici, parfois le mot code fait référence à toute la liste des modèles et à ce qu'ils représentent, comme dans "Ce message a été écrit avec un code secret". Parfois, le code fait simplement référence à l'un des modèles, comme dans « Quel code contient cet octet ? » Il sera assez évident d'après le contexte de quelle manière il est utilisé.

Retour à l'octet

Vous souvenez-vous de l'octet de mémoire que nous avons dessiné il y a quelques chapitres ? C'était huit bits de mémoire avec leurs fils 's tous connectés ensemble. Presque chaque fois que nous devons nous souvenir d'un octet à l'intérieur d'un ordinateur, nous avons également besoin d'une partie supplémentaire qui se connecte à la sortie de l'octet. Cette partie supplémentaire se compose de huit portes ET.

Ces huit portes ET, ensemble, sont appelées un « facilitateur ». Le dessin de gauche montre toutes les pièces, le dessin de droite est une façon plus simple de le dessiner.



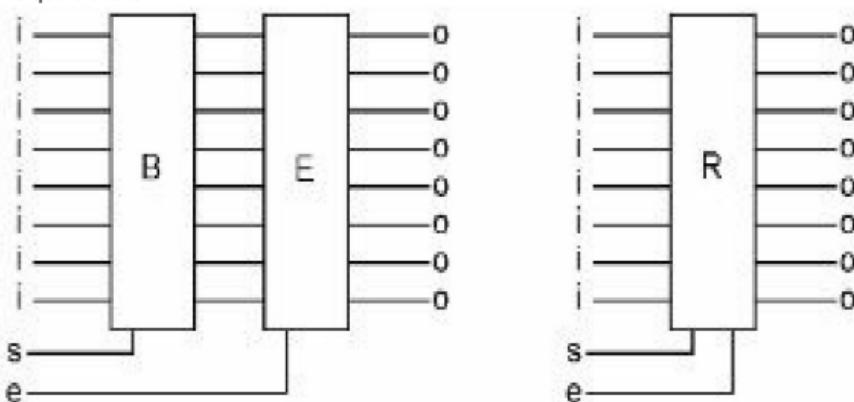
La deuxième entrée des huit portes ET est connectée ensemble et porte le nom « activer » ou « e » en abrégé.

Lorsque 'e' est désactivé, tout ce qui entre dans le Enabler ne va pas plus loin, car l'autre côté de chaque porte ET est désactivé, ainsi les sorties de ces portes vont toutes être désactivées.

Lorsque 'e' est activé, les entrées passent par le Enabler sans changement vers les sorties, 'o.' Au fait, lorsque des portes sont utilisées pour quelque

chose comme ça, le nom "porte" commence à avoir un sens. Un Enabler laisse passer un octet lorsque le bit 'e' est à 1 et arrête l'octet lorsqu'il est à 0. Ainsi, 'e' étant activé, c'est comme ouvrir une porte, et 'e' étant désactivé, c'est comme fermer une porte.

Nous allons prendre notre octet et le connecter à un activateur, comme indiqué dans le dessin de gauche. Pour simplifier encore une fois, nous pouvons le dessiner comme indiqué à droite.



Nous avons maintenant une combinaison qui peut stocker huit bits. Il les capture tous en même temps, et il peut soit les garder pour lui, soit les laisser sortir pour les utiliser ailleurs. Cette combinaison d'un Byte et d'un Enabler, a un nom, on l'appelle un Register, donc le 'R' dans le dessin.

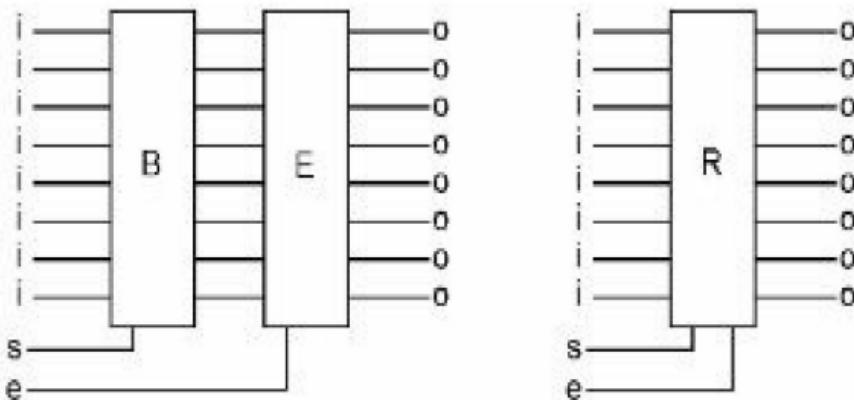
Il y aura quelques endroits dans ce livre où il y a des registres dont les sorties n'ont jamais besoin d'être désactivées. Dans ces cas, nous dessinerons un registre qui n'a qu'un bit 'set' et pas de bit 'enable'. Nous devrions probablement faire référence à ces périphériques en tant qu'"octets", mais nous les appellerons néanmoins des registres.

Le registre signifie simplement un endroit pour enregistrer une sorte d'information, comme un registre d'hôtel où tous les clients se connectent, ou un registre de chèques où vous écrivez chaque chèque qui est écrit. Dans le cas de cette partie informatique, vous enregistrez l'état des huit bits d'entrée. Ce registre est cependant très limité, en ce sens qu'il ne peut contenir qu'un seul ensemble de valeurs ; dans un registre d'hôtel, il y a une nouvelle ligne pour chaque client. Chaque fois que vous stockez un nouvel état dans un registre informatique, l'état précédent des huit bits de mémoire est perdu. La seule chose qui s'y trouve est la valeur la plus récemment enregistrée.

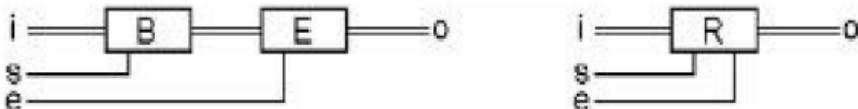
Le bus magique

Il existe de nombreux endroits dans un ordinateur où huit fils sont nécessaires pour connecter les registres ensemble. Notre registre, par exemple, a huit bits de mémoire, dont chacun a une entrée et une sortie. Pour simplifier nos schémas, nous allons remplacer nos huit fils par un double fil.

Ainsi, notre registre peut ressembler à l'un de ceux-ci :

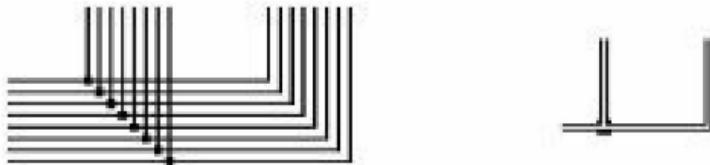


Ou, nous pouvons simplifier et le remplacer par l'un de ceux-ci :



C'est exactement la même chose, on va juste économiser beaucoup d'encre dans nos dessins, et ils seront plus faciles à comprendre.

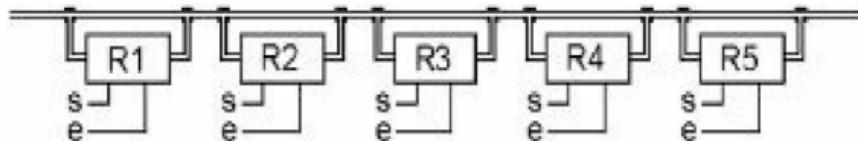
Lorsqu'il y a une connexion entre deux de ces faisceaux de fils, un fil de chaque faisceau est connecté à un fil de l'autre faisceau comme indiqué sur le schéma de gauche. Mais nous allons le simplifier et le dessiner simplement comme le schéma de droite.



Maintenant, ce regroupement de huit fils est si commun à l'intérieur des ordinateurs qu'il porte un nom. Cela s'appelle un autobus. Pourquoi s'appelle-t-il un bus ? Eh bien, cela a probablement à voir avec le vieux terme électrique "bus", qui signifie une barre de métal utilisée comme un très gros fil dans des endroits comme les centrales électriques. Mais il existe également une similitude intéressante avec le type d'autobus que les gens utilisent pour se déplacer.

Un bus est un véhicule qui se déplace généralement le long d'un itinéraire prédéterminé et fait de nombreux arrêts où les gens montent ou descendent. Ils commencent quelque part, et le bus les emmène à un autre endroit où ils doivent être. Dans le monde des ordinateurs, un bus est simplement un ensemble de huit fils qui vont à divers endroits à l'intérieur de l'ordinateur. Bien sûr, huit est le nombre de fils nécessaires pour transporter un octet d'information. À l'intérieur de l'ordinateur, le contenu des octets doit aller d'où ils se trouvent vers d'autres endroits, donc le bus va à tous ces endroits, et la conception du registre permet au contenu de n'importe quel octet sélectionné d'entrer dans le bus et d'en descendre à une destination choisie.

Dans l'exemple suivant, nous avons un bus et il y a cinq registres, dont chacun a son entrée et sa sortie connectées au même bus.



Si tous les bits 's' et 'e' sont désactivés, chaque registre sera défini comme il est et le restera. Si vous souhaitez copier les informations de R1 dans R4, activez d'abord le bit 'e' de R1. Les données de R1 seront désormais sur le bus et disponibles aux entrées des cinq registres. Si vous tournez ensuite brièvement le bit 's'

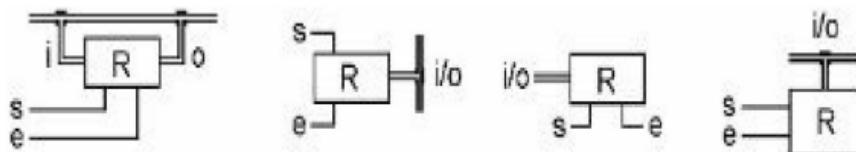
de R4 activé et désactivé, les données sur le bus seront capturées dans R4. L'octet a été copié. Donc un bus informatique c'est un peu comme le bus qui transporte les gens. Il y a un certain nombre d'arrêts et les octets peuvent arriver là où ils doivent aller.

Notez que nous pouvons copier n'importe quel octet dans n'importe quel autre octet. Vous pouvez copier R2 dans R5 ou R4 dans R1. Le bus fonctionne dans les deux sens. L'électricité mise sur le bus lorsque vous activez n'importe quel registre va aussi vite que possible aux entrées de tout le reste sur le bus. Vous pouvez même activer un registre sur le bus et le définir dans deux ou plusieurs autres registres en même temps. La seule chose que vous ne voulez pas faire est d'activer les sorties de deux registres sur le bus en même temps.

En termes de tailles de bits, vous pouvez le voir de cette façon : lorsque le bit 'e' de R1 est activé, les bits de R1 s'allongent maintenant, ils occupent un espace plus grand car ils sont maintenant connectés au bus, donc ces 8 bits incluent maintenant R1 et l'ensemble du bus. Lorsque le bit "s" de R4 est activé, les bits R1 deviennent encore plus gros car ils incluent désormais R1, le bus et R4. Si quelque chose dans R1 devait changer d'une manière ou d'une autre à ce moment-là, le bus et R4 changeraient immédiatement avec lui. Lorsque le bit « s » de R4 est désactivé, R4 retrouve son statut d'octet séparé, et lorsque le bit « e » de R1 est désactivé, le bus cesse de faire partie de R1.

C'est donc un bus. C'est un faisceau de huit fils qui va généralement à de nombreux endroits.

Une dernière chose à propos des registres : il existe de nombreux endroits où nous allons connecter l'entrée et la sortie d'un registre au même bus, donc pour simplifier encore plus, nous pouvons simplement montrer un faisceau de fils étiquetés " i/o ", ce qui signifie entrée et sortie. Tous les éléments suivants sont exactement équivalents en ce qui concerne leur fonctionnement. Le placement des fils sur le dessin peut être ajusté pour le rendre le plus épuré possible.



Autre note de langue : un octet est un emplacement qui peut se trouver dans l'un des 256 états. Parfois, nous parlons de déplacer un octet d'ici à là. Par définition, les octets ne se déplacent pas à l'intérieur de l'ordinateur. L'octet ne fait référence qu'à l'emplacement, mais parfois, lorsque quelqu'un veut se référer au réglage actuel de l'octet, et qu'il doit dire "permet de copier le contenu de R1 dans R4", il simplifie et dit "déplacer R1 vers R4" ou "déplacez cet octet là-bas." Ils utilisent le mot octet pour désigner le contenu de l'octet. Encore une fois, le contexte rend généralement cela très clair. Dans l'exemple ci-dessus de copie du contenu de R1 dans R4, vous pouvez l'entendre décrit comme "déplacement d'un octet de R1 à R4". Techniquement, R1 et R4 sont les octets, qui ne bougent pas, seul le contenu va d'un endroit à l'autre.

De plus, le contenu ne quitte pas l'endroit d'où il vient. Lorsque vous avez fini de « déplacer » un octet, l'octet « de » n'a pas changé, il ne perd pas ce qu'il avait. À l'autre extrémité, le motif qui était à l'origine dans l'octet « to » est maintenant « parti », il n'est allé nulle part, il a simplement été écrasé par la nouvelle information. L'ancien modèle cesse tout simplement d'exister. La nouvelle information est exactement la même que celle qui se trouve encore dans le premier octet. L'octet n'a pas bougé, il y a encore deux octets à deux endroits, mais les informations du premier octet ont été copiées dans le second octet.

Plus de combinaisons de portes

Maintenant, nous allons montrer seulement deux autres combinaisons, et alors nous pourrons rassembler ce que nous savons jusqu'à présent, faire la première moitié d'un ordinateur. Alors ne comprenez pas déçus, juste un peu plus loin et nous serons à mi-chemin domicile.

La première combinaison est très simple. C'est juste un ET porte avec plus de deux entrées. Si vous connectez deux ET portes comme ce schéma à gauche, vous voyez que pour 'd' doit être activé, les trois entrées, 'a', 'b' et 'c' doivent être sur. Donc, cette combinaison peut simplement être dessinée comme ceci schéma à droite :



Et le graphique qui montre comment cela fonctionne ressemble à cette:

a	b	c	ré
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0

1	1	0	0
1	1	1	1

Imaginez que vous remplacez l'entrée 'c' par une autre porte ET, vous auriez alors une porte ET à quatre entrées. Vous pouvez alors remplacer n'importe laquelle des quatre entrées par une autre porte ET et avoir une porte ET à cinq entrées. Cela peut être fait autant de fois que nécessaire pour ce que vous faites.

Au fur et à mesure que vous ajoutez des entrées, le graphique aura besoin de plus en plus de lignes. Chaque fois que vous ajoutez une autre entrée, vous doublez le nombre de combinaisons que les entrées peuvent avoir. Le graphique que nous avons vu pour la porte ET à deux entrées d'origine comportait quatre lignes, une pour chaque possibilité. Les trois entrées, directement au-dessus, ont huit lignes. Une porte ET à quatre entrées aura 16 lignes, une entrée à cinq en aura 32, etc. Dans tous les cas cependant, pour une porte ET, une seule combinaison entraînera l'activation de la sortie, c'est-à-dire la ligne où toutes les entrées sont activées.

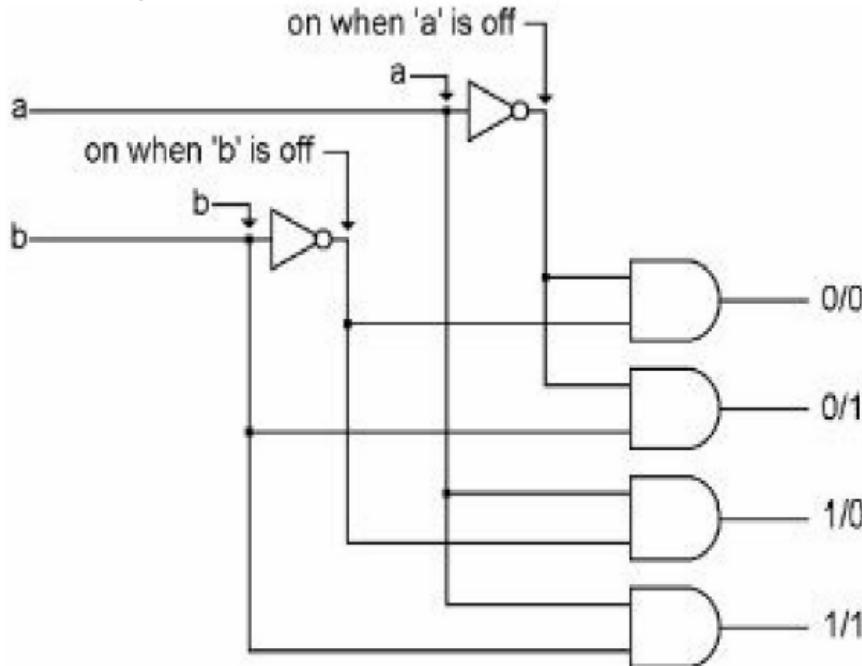
Voici la dernière combinaison dont nous avons besoin pour créer la première moitié d'un ordinateur. Cette combinaison est différente de tout ce que nous avons vu jusqu'à présent, en ce sens qu'elle a plus de sorties que d'entrées. Notre premier exemple a deux entrées et quatre sorties. Ce n'est pas très compliqué, il a juste deux portes NOT et quatre portes ET.

Dans le diagramme ci-dessous, 'a' et 'b' sont les entrées venant de la gauche. Les deux sont connectés à des portes NOT. Les portes NOT génèrent l'opposé de leurs entrées. Il y a quatre fils verticaux descendant la page qui viennent de 'a' et 'b' et les opposés de 'a' et 'b'. Ainsi, pour chaque 'a' et 'b', il y a deux fils descendant la page, où l'un d'eux sera allumé si son entrée est allumée, et l'autre sera allumé si son entrée est éteinte. Maintenant, nous mettons quatre portes ET sur la droite et connectons chacune à une paire différente de fils verticaux de sorte que chaque porte ET s'allume pour une autre des quatre combinaisons possibles de 'a' et 'b'. Haut

La porte ET, étiquetée "0/0" est connectée au fil qui est allumé quand 'a' est éteint, et le fil qui est allumé quand 'b' est off, et s'allume donc quand 'a' et 'b' sont tous les deux 0. La porte ET suivante, "0/1" est connecté au fil qui est sur lorsque 'a' est désactivé et 'b', il s'allume donc lorsque 'a' vaut 0 et 'b' vaut 1, etc.

Les entrées peuvent être activées dans n'importe quelle combinaison, les deux bits désactivés, l'un allumé, l'autre allumé ou les deux allumés. Aucun, un ou deux.

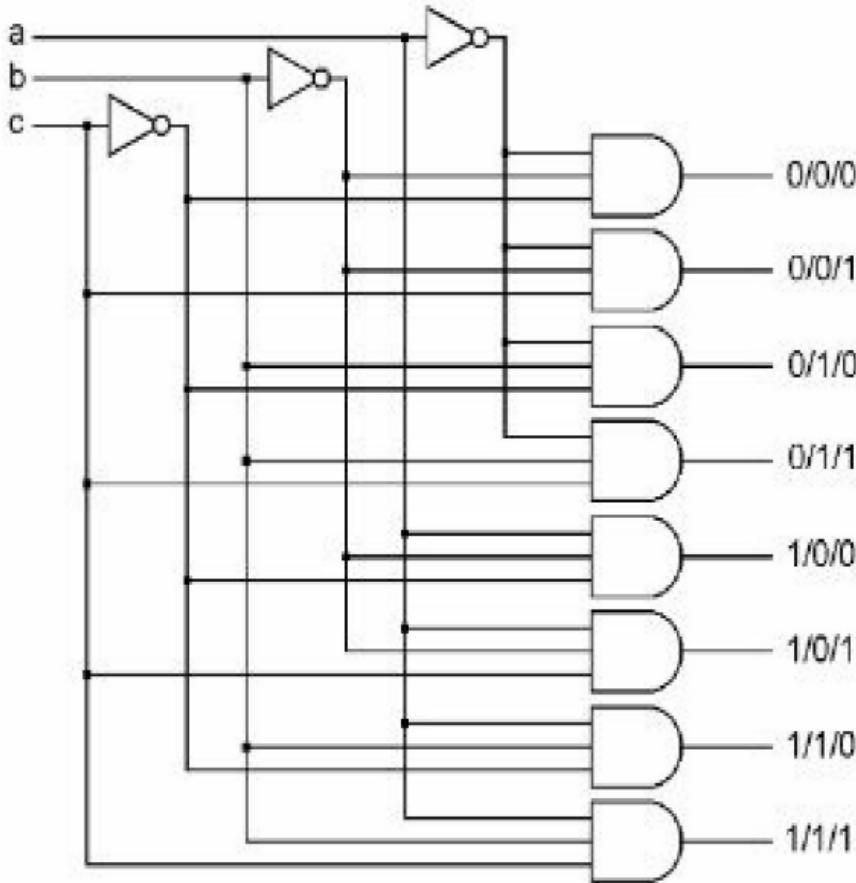
Les sorties, cependant, auront toujours un et un seul sortie activée et les trois autres désactivées. Celui qui est allumé est déterminé par les états actuels de 'a' et 'b.'



0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Cette combinaison s'appelle un décodeur. Le nom signifie que si vous considérez les quatre états possibles des deux entrées sous forme de code, la sortie vous indique laquelle des codes est actuellement sur l'entrée. Peut-être que ce n'est pas génial nom, mais c'est ce que cela signifiait pour quelqu'un une fois, et le nom coincé. Ce décodeur a deux entrées, ce qui signifie qu'il peut y avoir quatre combinaisons d'états entrées, et il y a quatre sorties, une correspondant à chacune des combinaisons d'entrées possibles.

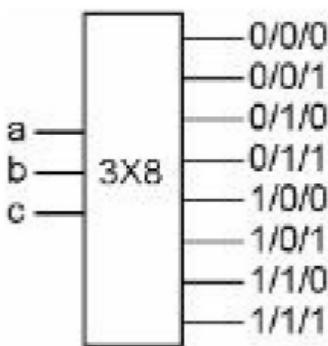
Cela peut être prolongé. Si nous ajoutons une troisième entrée, il y a serait alors huit combinaisons d'entrées possibles, et si nous avons utilisé huit, trois portes ET d'entrée, nous pourrions construire un Décodeur à trois entrées et huit sorties. De même, nous pourrions construire un décodeur à quatre entrées et 16 sorties. Les décodeurs sont nommés par le nombre d'entrées "X" le nombre de sorties. Comme 2X4, 3X8, 4X16, 5X32, 6X64, etc.



Encore une fois, nous allons simplifier nos dessins, nous ne montrerons aucune des pièces internes ou du câblage, nous aurons juste une boîte avec un nom et les entrées et sorties qui nous intéressent. Nous avons vu comment les portes NAND font NOT les portes et les portes ET, puis les portes NON et les portes ET forment un décodeur. C'est une boîte pleine de portes NAND câblées pour faire quelque chose d'utile. Nous savons ce qu'il fait, un et

une seule des sorties est toujours allumée, et laquelle est déterminée par l'état des trois entrées.

C'est tout ce qu'il fait.



Première moitié de l'ordinateur

Construisons quelque chose avec les pièces que nous avons jusqu'à présent. En fait, nous pouvons maintenant construire la moitié de ce qu'il y a dans un ordinateur.

Tout d'abord, construisons quelque chose de similaire en bois (dans notre esprit), puis nous reviendrons et montrerons comment construire une version informatique qui fait à peu près la même chose.

Vous savez, dans un hôtel, à la réception, sur le mur derrière le greffier, il y a une série de petits cagibis en bois, un pour chaque chambre de l'hôtel. C'est là qu'ils gardent les clés de chambre supplémentaires et les messages ou le courrier pour les invités. Ou vous avez peut-être vu un vieux film où quelqu'un dans un ancien bureau de poste triait le courrier. Il est assis à une table avec une série de casiers à l'arrière. Il a une pile de courrier non trié sur la table, en prend un à la fois, lit l'adresse et place la lettre dans le casier approprié.

Nous allons donc construire des cagibis. Le nôtre aura trois pouces carrés, et il y aura seize cagibi de haut et seize cagibi de large. C'est une taille totale de quatre pieds sur quatre pieds, avec un total de deux cent cinquante-six cubes.

Maintenant, nous allons ajouter quelque chose qu'ils n'ont pas à la poste ou à l'hôtel. Nous allons placer un grand panneau de bois juste devant les casiers qui est deux fois plus large que l'ensemble, et au milieu il y a une fente verticale qui est juste assez grande pour exposer une colonne de 16 casiers. Le panneau aura des roues sur le fond afin qu'il puisse glisser à gauche et à droite pour exposer l'une des colonnes verticales de seize casiers à la fois et couvrir toutes les autres colonnes.

Prenons un autre panneau de bois comme le premier, mais tournez-le sur le côté pour qu'il soit deux fois plus haut que nos cagibis, et la fente au milieu va d'un côté à l'autre. Ce deuxième panneau sera monté juste en face du premier, dans quelque chose comme un cadre de fenêtre, de sorte qu'il puisse glisser de haut en bas, exposant une seule rangée de seize casiers à la fois.

Nous avons donc maintenant une série de 256 cagibis, et deux panneaux de bois à fentes devant eux qui ne permettent que

un casier à la fois pour être visible. Dans chacun de ces casiers, nous placerons un seul bout de papier sur lequel nous écrirons une des combinaisons possibles de huit zéros et uns.

Ce dispositif de cagibi dispose de 256 emplacements pour stocker quelque chose.

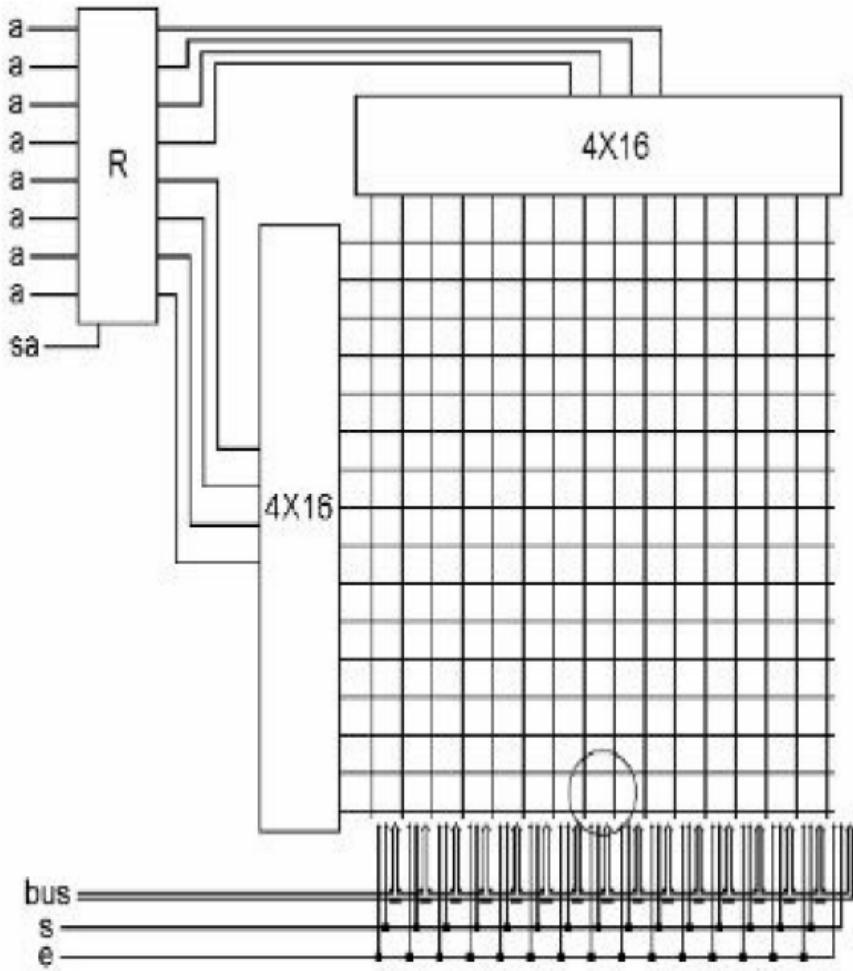
À tout moment, nous pouvons sélectionner un et un seul de ces endroits en faisant glisser les panneaux de bois d'un côté à l'autre ou de haut en bas. Au cagibi sélectionné, nous pouvons atteindre et obtenir le bout de papier et le lire, ou le remplacer par un autre.

Maintenant, nous allons prendre les portes, les registres et les décodeurs que nous avons décrits et en faire quelque chose qui fait à peu près la même chose que notre dispositif de cagibi.

Cette chose aura 256 emplacements dans lesquels stocker quelque chose, et nous pourrons sélectionner un et un seul de ces emplacements à un moment donné.

En faisant référence au schéma ci-dessous, nous commençons avec un seul registre. Son entrée "a" est un bus qui vient d'un autre endroit de l'ordinateur. Une combinaison de bits est placée sur le bus et le bit 'sa' (set a) passe à 1 puis à 0. Ce modèle de bit est maintenant stocké dans ce registre, qui est l'un de ces registres dont la sortie est toujours activée. Les quatre premiers bits de sortie sont connectés à un décodeur 4X16, et les quatre autres bits de sortie sont connectés à un autre décodeur 4X16. Les sorties des deux décodeurs sont disposées selon un motif de grille. Les fils ne se touchent pas, mais il y a ici 16 par 16, soit 256 intersections que nous utiliserons bientôt. Un décodeur, comme indiqué, a une et une seule de ses sorties allumées à tout moment, et les autres sont éteintes. Puisque nous avons deux décodeurs ici, il y aura un fil de grille horizontal et un fil de grille vertical. Par conséquent, sur ces 256 intersections, il n'y aura qu'une seule intersection où les fils horizontaux et verticaux sont allumés. Quelle intersection changera chaque fois que la valeur dans R est modifiée, mais il y en aura toujours une où les deux fils sont allumés tandis que les 255 autres n'en auront qu'un seul

ou aucun activé.



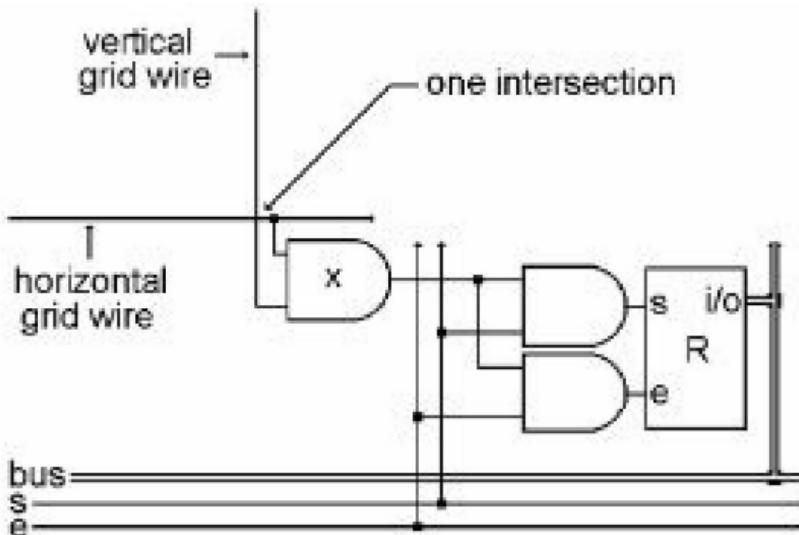
Au bas de ce diagramme se trouve un bus et un bit « s » et « e », tout comme les connexions qui vont à un registre. Comme vous pouvez le voir, ils vont vers le haut et dans la grille. Le diagramme ne le montre pas, mais ils montent sous

la grille jusqu'en haut, de sorte que chacune des 256 intersections ait un bus et un 's' et 'e' à proximité.

Il y a un cercle sur le schéma ci-dessus, autour d'une des intersections de la grille.

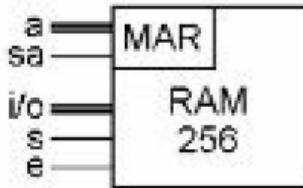
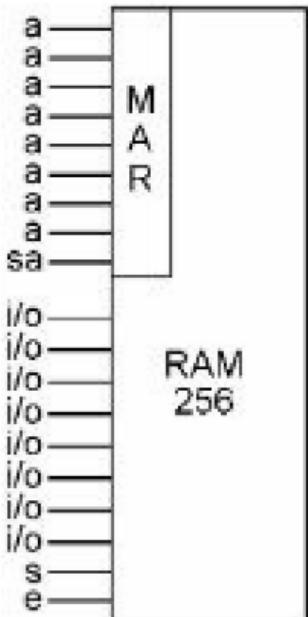
Ce qui se trouve dans ce cercle est agrandi dans le diagramme ci-dessous, montrant qu'il y a trois portes ET et un registre à chacune des 256 intersections. Comme nous pouvons le voir, il y a une porte ET 'x' connectée au fil de grille vertical et au fil de grille horizontal à cette intersection. Ces portes 'x' sont les seules choses connectées au réseau. Le reste des connexions descend vers le bus et les bits "s" et "e" au bas du diagramme. N'oubliez pas qu'il n'y a qu'une seule intersection où les deux fils du réseau sont allumés. Par conséquent, il y a 256 de ces portes "x", mais une seule d'entre elles a sa sortie à un moment donné.

La sortie de cette porte 'x' va d'un côté de chacune des deux autres portes ET. Ces deux portes contrôlent l'accès à l'ensemble et autorisent les entrées du registre à cette intersection. Ainsi, lorsqu'une porte 'x' est désactivée, les bits 's' et 'e' de ce registre ne peuvent pas être activés. Ce sera le cas pour 255 de ces registres, ceux où la porte 'x' est désactivée. Mais une intersection a sa porte 'x' activée, et son registre peut être défini à partir du bus, ou son contenu peut être activé sur le bus et envoyé ailleurs en utilisant les bits 's' et 'e' au bas du diagramme .



Ce qui précède est la mémoire principale de l'ordinateur. C'est la moitié de ce qui est nécessaire pour construire un ordinateur. Il est parfois appelé par des noms différents, mais le nom le plus correct vient du fait que vous pouvez sélectionner n'importe lequel des 256 octets à un moment donné, puis vous pouvez immédiatement sélectionner n'importe quel autre des 256 octets, et peu importe où le dernier était, ou où se trouve le suivant, il n'y a aucun avantage ou inconvénient de vitesse à l'ordre dans lequel vous sélectionnez les octets. En raison de cette qualité, il s'agit d'un bon type de mémoire à utiliser si vous souhaitez pouvoir accéder aux octets de mémoire dans un ordre aléatoire. Ce type de mémoire est donc appelé "Random Access Memory" ou "RAM" en abrégé.

C'est la RAM. Il utilise 257 registres. 256 registres sont des emplacements de stockage de mémoire, un registre est utilisé pour sélectionner l'un des emplacements de stockage et est appelé "Registre d'adresse mémoire" ou "MAR" en abrégé. Maintenant que nous savons ce qu'il contient, nous pouvons créer un schéma simplifié comme celui-ci, et une version de bus encore plus simple :



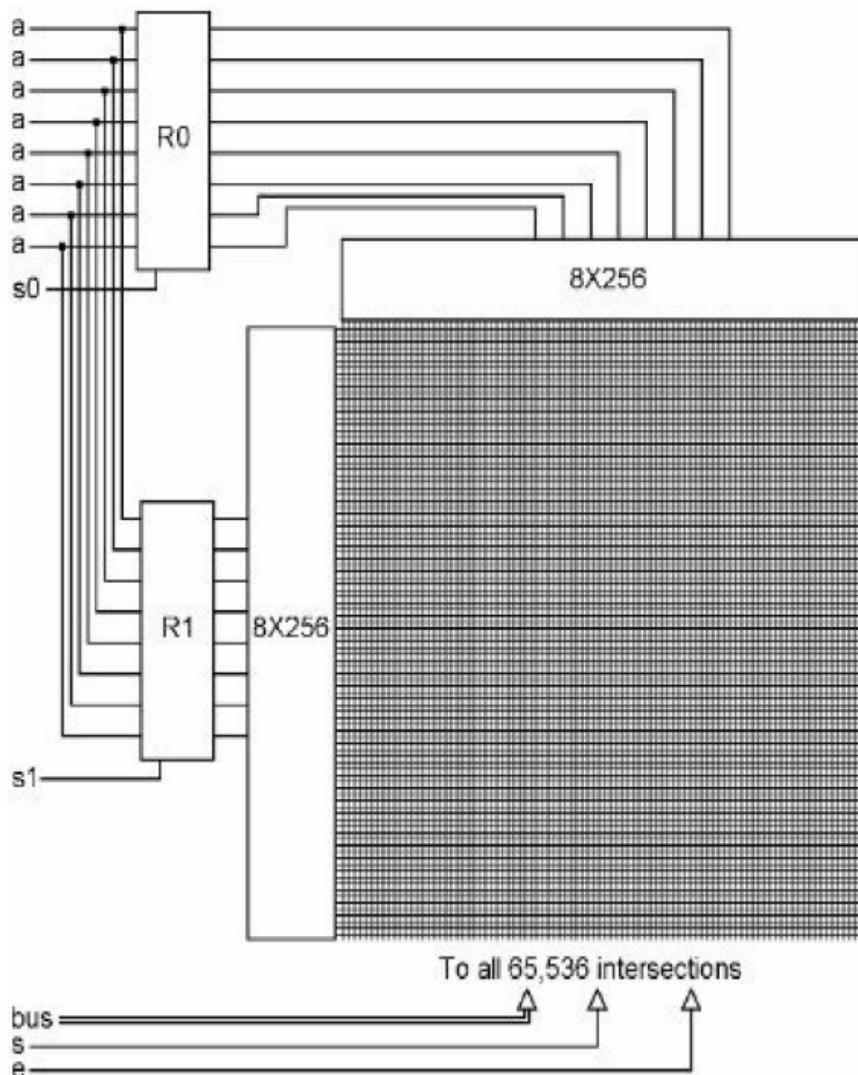
C'est la moitié d'un ordinateur. Un ordinateur n'a que deux parties, et celle-ci en est une. Alors maintenant, vous savez la moitié de ce qu'il y a à l'intérieur d'un ordinateur. Chaque pièce est constituée de portes NAND. Ce n'était pas très difficile n'est-ce pas ?

Il y a un problème ici, et c'est que 256 octets est une très petite taille pour la RAM d'un ordinateur. Nous pourrons peut-être nous en sortir dans ce livre, mais si vous voulez un vrai ordinateur, il vous faudra une RAM avec beaucoup plus d'octets parmi lesquels choisir.

Une RAM plus grande peut être construite en fournissant deux registres qui sont utilisés pour sélectionner un emplacement de stockage de mémoire. Cela permet l'utilisation de décodeurs 8X256 et se traduit par une grille avec 65 536 intersections, et donc une RAM avec 65 536 emplacements différents dans lesquels stocker quelque chose.

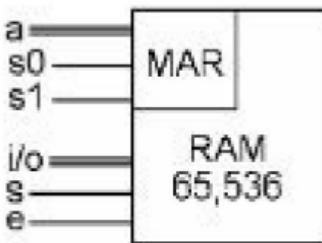
Voici une idée de ce à quoi cela ressemblerait : (Ne vous embêtez pas

en essayant de compter les lignes de la grille, il n'a été possible d'en insérer qu'environ la moitié sur la page imprimée.)



Un bus transporte un octet à la fois, donc la sélection de l'un des 65 536 emplacements de mémoire de cette RAM serait un processus en deux étapes. Tout d'abord, un octet devrait être placé sur le bus 'a' et mis dans R0, puis le deuxième octet devrait être placé sur le bus 'a' et mis dans R1. Vous pouvez maintenant accéder à l'emplacement mémoire souhaité avec le bus et les bits "s" et "e" en bas du dessin.

En simplifiant encore, nous avons quelque chose qui ressemble beaucoup à notre RAM de 256 octets, il a juste un bit d'entrée de plus.



Pour le reste de ce livre, nous utiliserons la RAM de 256 octets juste pour garder les choses simples. Si vous voulez imaginer un ordinateur avec une plus grande RAM, chaque fois que nous envoyons un octet au registre d'adresses mémoire, tout ce que vous avez à faire est d'imaginer envoyer deux octets à la place.

Nombres

Nous allons revenir sur le sujet des codes pour une moment. Auparavant, nous avons examiné un code appelé ASCII qui est utilisé pour représenter le langage écrit. Eh bien, les chiffres sont utilisés aussi dans la langue écrite, donc il y a des codes ASCII pour les chiffres de zéro à neuf. Plus tôt, nous avons vu 20 de les codes ASCII d'une partie de l'alphabet, en voici 10 de plus, les codes des nombres en langage écrit :

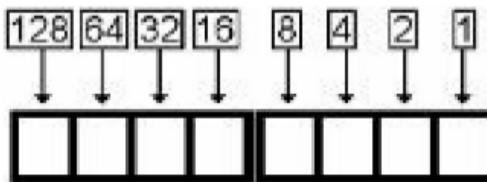
0	0011 0000
1	0011 0001
2	0011 0010
3	0011 0011
4	0011 0100
5	0011 0101
6	0011 0110
-	0011 0111
8	0011 1000
9	0011 1001

C'est un code très utile, mais pas tout ce qui les ordinateurs ont quelque chose à voir avec le langage écrit. Pour les autres tâches, il existe d'autres codes qui conviennent mieux à ces tâches. En ce qui concerne les nombres, si vous utilisez ASCII, un octet peut être l'un des 10 chiffres de 0 à 9. Mais parfois il y a un octet qui est toujours utilisé pour stocker un numéro, et ce numéro ne sera jamais imprimé ou affiché à l'écran. Dans ce cas, nous pouvons utiliser un code différent qui ne gaspille aucun de ses possibles états sur les lettres de l'alphabet ou quoi que ce soit d'autre que Nombres. Comme un octet a 256 états possibles, vous pouvez

que ce code représente 256 nombres différents. Puisque nous voulons inclure zéro, ce code commence à zéro et monte jusqu'à 255.

Maintenant, comment ce code est-il arrangé ? L'ASCII ci-dessus n'est pas du tout utilisé ; c'est un code complètement différent. Ce code n'a nécessité aucune réunion de comité pour être inventé car c'est le code le plus simple et le plus évident que les ordinateurs utilisent. C'est ce qui se rapproche le plus d'un code informatique « naturel ».

Comme il s'agit d'un long chapitre, voici un aperçu de ce code. Elle consiste à attribuer une valeur numérique à chaque bit de l'octet. Pour utiliser ce code, activez simplement les bits qui s'ajoutent au nombre que vous souhaitez représenter.



Pour voir comment ce code fonctionne, pourquoi il est utilisé dans les ordinateurs et comment ces valeurs de bits ont été choisies, nous examinerons le sujet des nombres en dehors des ordinateurs.

Il existe trois systèmes de numération que vous connaissez probablement et que nous pouvons analyser. Selon moi, ces trois systèmes sont chacun constitués de deux idées ou éléments - premièrement, une liste de symboles, et deuxièmement, une méthode pour utiliser ces symboles.

Le système de numération le plus ancien est probablement une chose simple appelée Tally Marks. Il a deux symboles, "|" et "/" . La méthode d'utilisation de ces symboles consiste à écrire un "|" pour chacune des quatre premières choses que vous comptez, puis pour la cinquième marque, vous écrivez un "/" sur les quatre premières. Vous répétez cela encore et encore aussi longtemps que nécessaire, puis lorsque vous avez terminé, vous comptez les marques par groupes de cinq - 5, 10, 15, 20, etc. Ce système est très bon pour compter les choses au fur et à mesure qu'elles passent, disons ton troupeau de moutons. Au fur et à mesure que chaque animal passe, vous n'avez qu'à rayer une marque de plus - vous n'avez pas besoin de barrer '6' et d'écrire '7'. Ce système a un autre avantage en ce sens qu'il y a en fait une note pour chaque

chose qui a été comptée. Plus tard dans le chapitre, nous allons faire des choses intéressantes avec des nombres qui peuvent prêter à confusion, donc afin de garder les choses claires, nous utiliserons cet ancien système.

Vous souvenez-vous des chiffres romains ? C'est un système de numération qui se compose également de deux éléments. Le premier élément est les symboles, juste des lettres sélectionnées de l'alphabet, 'I' pour un, 'V' pour cinq, 'X' pour dix, 'L' pour cinquante, 'C' pour cent, 'D' pour cinq cent, 'M' pour mille. Le deuxième élément est une méthode qui vous permet de représenter des nombres qui n'ont pas un seul symbole.

La méthode romaine dit que vous écrivez plusieurs symboles, les plus grands en premier, et que vous les additionnez, sauf lorsqu'un symbole plus petit est à gauche d'un plus grand, puis vous le soustrayez. Donc 'II' est deux (ajoutez un et un) et 'IV' est quatre (soustrayez un de cinq). L'année est devenue beaucoup plus simple. 1999 était "MCMXCIX", vous devez faire trois soustractions dans votre tête juste pour lire celle-là. 2000 était simplement 'MM.' Le système numérique normal que nous utilisons aujourd'hui se compose également de deux idées, mais ce sont deux idées très différentes qui nous sont venues par l'Arabie plutôt que par Rome. La première de ces idées concerne également les symboles, dans ce cas 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. Ces chiffres sont des symboles qui représentent une quantité. La deuxième idée est une méthode à laquelle nous sommes tellement habitués que nous l'utilisons instinctivement. Cette méthode dit que si vous écrivez un chiffre, cela signifie ce qu'il dit. Si vous écrivez deux chiffres l'un à côté de l'autre, celui de droite signifie ce qu'il dit, mais celui de gauche signifie dix fois ce qu'il dit. Si vous écrivez trois chiffres côté à côté, celui de droite signifie ce qu'il dit, celui du milieu signifie dix fois ce qu'il dit et celui de gauche signifie cent fois ce qu'il dit. Lorsque vous voulez exprimer un nombre supérieur à 9, vous le faites en utilisant plusieurs chiffres, et vous utilisez cette méthode qui dit que le nombre de positions à gauche du premier chiffre vous indique combien de fois vous le multipliez par dix avant de additionnez-les. Donc, si vous avez '246' pommes, cela signifie que vous avez deux cents pommes plus quarante pommes plus six pommes.

Donc comment ça fonctionne? Un nombre de n'importe quel montant peut être écrit avec les chiffres de zéro à neuf, mais lorsque vous dépassiez neuf, vous devez utiliser deux chiffres. Lorsque vous dépassiez quatre-vingt-dix-neuf, vous devez utiliser trois chiffres.

Au-dessus de neuf cent quatre vingt dix neuf, vous passez à quatre chiffres, etc. et chaque fois que vous revenez de neuf à zéro, vous augmentez le chiffre vers la gauche de 1. Vous n'avez donc que dix symboles, mais vous pouvez en utiliser plusieurs selon les besoins et leurs positions les unes par rapport aux autres précisent leur valeur complète .

Il y a quelque chose d'étrange à ce sujet dans la mesure où le système est basé sur dix, mais il n'y a pas de symbole unique pour dix. D'un autre côté, il y a quelque chose de juste à ce sujet - les symboles '0' à '9' constituent dix symboles différents. Si nous avions aussi un seul symbole pour dix, il y aurait en fait onze symboles différents. Donc celui qui a pensé à ça était assez intelligent.

L'une des nouvelles idées de ce système arabe était d'avoir un symbole pour zéro. C'est utile si vous voulez dire que vous n'avez aucune pomme, mais c'est aussi une chose nécessaire pour garder les positions des chiffres droites. Si vous avez 50 pommes ou 107 pommes, vous avez besoin des zéros dans les nombres pour savoir à quelle position se trouve réellement chaque chiffre, afin que vous puissiez multiplier par dix le nombre correct de fois.

Maintenant, ces deux idées dans le système numérique arabe (les chiffres et la méthode) ont une chose en commun. Ils ont tous deux le numéro dix qui leur est associé. Il y a dix chiffres différents, et lorsque vous ajoutez des chiffres à gauche d'un nombre, chaque position vaut dix fois plus que la précédente.

À l'école, lorsqu'ils enseignent les chiffres aux enfants pour la première fois, ils disent quelque chose sur le fait que notre système de numération est basé sur le nombre dix, car nous avons dix doigts. Voici donc une question étrange : et si ce système de numération avait été inventé par des paresseux à trois doigts ? Ils n'ont que trois doigts à chaque main et pas de pouce. Ils auraient inventé un système numérique avec seulement six chiffres - 0, 1, 2, 3, 4 et 5. Cela pourrait-il fonctionner ? Si vous aviez huit pommes, comment l'écririez-vous ? Il n'y a pas de chiffre '8' dans ce système. La réponse est que depuis la première idée, le

chiffres, a été modifié pour n'avoir que six chiffres, alors la deuxième idée, la méthode, devrait également être modifiée de sorte que lorsque vous ajoutez des positions à gauche, chacune devrait être multipliée par six au lieu de dizaines. Alors ce système fonctionnerait. En comptant vos pommes, vous diriez « 0, 1, 2, 3, 4, 5... » et puis quoi ? Il n'y a pas de '6' pour le numéro suivant. Eh bien, selon la méthode, lorsque vous voulez dépasser le chiffre le plus élevé, vous revenez à '0' et ajoutez un '1' à gauche. D'accord, "0, 1, 2, 3, 4, 5, 10, 11, 12." Maintenant, vous avez compté toutes vos pommes. Que signifierait ce '12' ? Ce serait ceci



. Je suppose que tu l'appellerais huit, mais tu écriras beaucoup : c'est '12'. Très étrange, mais ça marche - 1 fois six plus deux égale huit pommes, il suit la méthode arabe, mais il est basé sur six au lieu de dix. Si vous continuez à compter, lorsque vous arrivez à '15', qui est



(une fois six plus cinq), le nombre suivant serait « 20 », mais le « 2 »

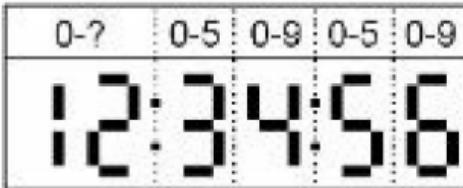
signifierait deux six, ou ce nombre :



. Et 55 serait suivi de 100. Le '1' dans ce la troisième position serait combien de '36 il y avait (six fois six)

C'est un système de nombres très impairs, mais devinez quoi, vous l'utilisez déjà dans votre vie de tous les jours. Oui, pensez à la façon dont nous écrivons l'heure, ou au genre d'horloge qui affiche les chiffres sur son cadran. Le bon chiffre des minutes et des secondes suit nos chiffres normaux, 0-9, 0-9, encore et encore. Mais le chiffre de gauche des minutes et des secondes ne va que de 0 à 5. Après 59 minutes, l'horloge passe à l'heure et 00 minutes suivantes. Il y a 60 minutes dans une heure, numérotées de 00 à 59. Cette position de gauche ne dépasse jamais 5. Cette position utilise le système de numération basé sur six symboles (0-5). La partie heure de l'horloge indique combien il y a de 60, bien que vous ne verrez jamais un 60 sur le cadran de l'horloge. Et vous êtes tellement habitué à cela que vous n'avez pas à y penser. Lorsque l'horloge indique 1h30, vous savez que c'est à mi-chemin entre 1h00 et 2h00, vous n'avez pas besoin de faire de calcul dans votre tête pour le comprendre. Avez-vous déjà dû ajouter du temps ? Si vous ajoutez 40 minutes et 40 minutes, vous obtenez 80 minutes, mais pour écrire cela en heures et minutes, vous devez déterminer combien de 60 il y a dans 80, dans ce cas 1, puis déterminer combien de minutes il y a au-delà de 60 ans, dans ce

cas 20. Vous écrivez donc 1:20. Le 1 représente 60 minutes, ajoutez 20 et vous avez à nouveau vos 80. C'est donc assez compliqué, deux systèmes de numération différents dans le même nombre ! Mais vous l'avez déjà utilisé toute votre vie.



Les positions horaires sont encore plus étranges. Sur une horloge de 12 heures, il saute zéro et passe de 1h à 12h, puis de 13h à 12h. Sur une horloge de 24 heures, cela va de 00 à 23. Nous n'essaierons pas de les analyser. Le point que nous voulions souligner est que vous connaissez déjà les systèmes de numération basés sur des nombres autres que dix.

Vous pouvez inventer un système de numération pour n'importe quel nombre de chiffres, 10 ou 6 comme nous l'avons vu ci-dessus, ou 3 ou 14 ou n'importe quel nombre que vous choisissez. Mais le plus simple serait si vous n'aviez que 2 chiffres, 0 et 1. Comment celui-ci fonctionnerait-il ?

Eh bien, vous comptez 0, 1... et puis vous êtes déjà à court de chiffres - alors revenez à 0 et ajoutez 1 à gauche, faisant le prochain numéro 10 puis 11, puis vous êtes à nouveau à court de chiffres, donc 100 puis 101, 110, 111 puis 1000. Ce système est basé sur deux, donc il n'y a que deux chiffres, et comme vous ajoutez des positions à gauche, chacune vaut deux fois plus que la précédente. La bonne position signifie ce qu'elle dit, la suivante à gauche signifie deux fois ce qu'elle dit, la suivante signifie quatre fois ce qu'elle dit, la suivante signifie huit fois, etc. Lorsque vous en arrivez à n'avoir que deux chiffres possibles, vous n'avez pas besoin de faire beaucoup de multiplications pour déterminer la valeur totale d'une position. Dans la position qui vaut 'huit', par exemple, il ne peut y avoir qu'un un, signifiant un 'huit', ou un zéro, signifiant 'pas de huit'. Tant qu'on y est, imaginons un animal très étrange avec huit doigts à chaque main. Cet animal aurait inventé des nombres basés sur seize. Dans leur système, ils seraient capables d'écrire de dix à quinze chacun avec un

symbole unique. Ce n'est qu'une fois arrivés à seize ans qu'ils revenaient à 0 et devaient mettre un 1 dans la position de gauche. Pour voir comment cela fonctionnerait, nous avons besoin de six nouveaux symboles, alors utilisons simplement les six premières lettres de l'alphabet. « A » signifie dix, « B » signifie onze, « C » signifie douze, « D » signifie treize, « E » signifie quatorze et « F » signifie quinze. Ce n'est qu'après avoir utilisé les seize symboles dans la bonne position que nous manquerons de symboles, et le prochain numéro sera seize, écrit «10» dans ce système. Si vous êtes familier avec le système de poids en livres et en onces, c'est un peu comme ce système. Il y a 16 onces dans une livre, vous savez donc que 8 onces valent une demi-livre. L'ajout de 9 onces et 9 onces donne 1 livre 2 onces.

Voici un tableau qui montre cinq systèmes de numération différents. La première colonne est l'ancien système de pointage pour le garder raisonnable.

Tally	0-9	0-5	0-1	0-F
	0	0	0	0
I	1	1	1	1
II	2	2	10	2
III	3	3	11	3
IV	4	4	100	4
V	5	5	101	5
VI	6	10	110	6
VII	7	11	111	7
VIII	8	12	1000	8
VIII I	9	13	1001	9
VIII II	10	14	1010	A
VIII III	11	15	1011	B
VIII IV	12	20	1100	C
VIII V	13	21	1101	D
VIII VI	14	22	1110	E
VIII VII	15	23	1111	F
VIII VIII	16	24	10000	10
VIII VIII I	17	25	10001	11
VIII VIII II	18	30	10010	12

Nos nombres normaux de 0 à 9 sont appelés le système décimal, car « dec » signifie dix dans une langue ancienne. Le système 0-5 serait appelé le système sénaire, car « sen » signifie six dans une autre langue ancienne. Ce nouveau

Le système avec juste 0 et 1 est appelé le système binaire parce que "bi" signifie deux, également à cause d'un langage ancien. Cet autre nouveau système, le système 0-F, sera appelé le système hexadécimal, car "hex" est un autre mot ancien qui signifie six et "dec" signifie toujours dix, c'est donc le système six plus dix.

Une autre méthode pour nommer différents systèmes de numération consiste à les appeler par le nombre sur lequel ils sont basés, comme "base 10" ou "base 2", etc. signifiant décimal ou binaire, etc.

Mais notez que le nombre après le mot 'base' est écrit dans le système décimal. '2' écrit en binaire est '10', donc 'base 10' signifierait binaire si le '10' était écrit en binaire. En fait, chaque système de numération serait en « base 10 » si le « 10 » était écrit dans les nombres de ce système ! On pourrait donc parler de base 2, base 6, base 10 et base 16 si on le voulait, à condition de se rappeler que ces nombres de base sont écrits en décimal. Si on parle de binaire, sénaire, décimal et hexadécimal, c'est la même chose, juste peut-être un peu moins déroutant.

Encore une fois, dans nos nombres décimaux normaux, la position la plus à droite est le nombre d'unités. La position suivante à gauche est le nombre de dizaines, etc. Chaque position vaut dix fois la précédente. Dans le système binaire, la position la plus à droite est également le nombre de uns, mais la position suivante à gauche est le nombre de « deux », la suivante à gauche est le nombre de « quatre », la suivante est le nombre de « huit ». Chaque position vaut deux fois le montant à sa droite. Étant donné que chaque position n'a que deux valeurs possibles, zéro ou un, c'est quelque chose que nous pourrions utiliser dans un octet.

C'est le propos de ce chapitre. Le système de numération binaire correspond « naturellement » aux capacités des composants informatiques. Nous pouvons l'utiliser comme un code, avec 0 représentant zéro et 1 représentant un, en suivant la méthode des nombres arabes avec seulement deux symboles. Dans un octet, nous avons huit bits. Quand on utilise ce code, le bit de droite vaudra 1 quand le bit est allumé, ou 0 quand il est éteint. Le bit suivant à gauche vaudra 2 lorsqu'il est activé, ou 0 lorsqu'il est désactivé. Le suivant à gauche est 4, et ainsi de suite avec 8, 16, 32, 64 et 128. Dans l'ordre dans lequel nous les voyons normalement, les valeurs des huit bits ressemblent à ceci : 128 64 32 16 8 4 2 1.

Dans ce code, 0000 0001 signifie un, 0001 0000 signifie seize, 0001 0001 signifie dix-sept (seize plus un), 1111 1111 signifie 255, etc. Dans un octet de huit bits, nous pouvons représenter un nombre n'importe où entre 0 et 255. Ce code est appelé le « code numérique binaire ».

L'ordinateur fonctionne très bien avec cet arrangement, mais c'est ennuyeux à utiliser. Le simple fait de dire ce qu'il y a dans un octet est un problème. Si vous avez 0000 0010, vous pouvez l'appeler "zéro zéro zéro zéro zéro un zéro binaire" ou vous pouvez le traduire mentalement en décimal et l'appeler "deux", et c'est généralement ce qui est fait. Dans ce livre, lorsqu'un nombre est épelé, tel que "douze", cela signifie 12 dans notre système décimal. Un binaire 0000 0100 serait appelé "quatre", car c'est ce que cela donne en décimal.

En fait, dans l'industrie informatique, les gens utilisent souvent l'hexadécimal (et ils l'appellent simplement "hex"). Si vous regardez le tableau ci-dessus, vous pouvez voir que quatre chiffres binaires peuvent être exprimés par un chiffre hexadécimal. Si vous avez un octet contenant 0011 1100, vous pouvez le traduire en 60 décimal, ou simplement l'appeler "3C hex". Ne vous inquiétez pas, nous n'utiliserons pas l'hexagone dans ce livre, mais vous avez peut-être vu ce type de nombres quelque part, et maintenant vous savez de quoi il s'agit.

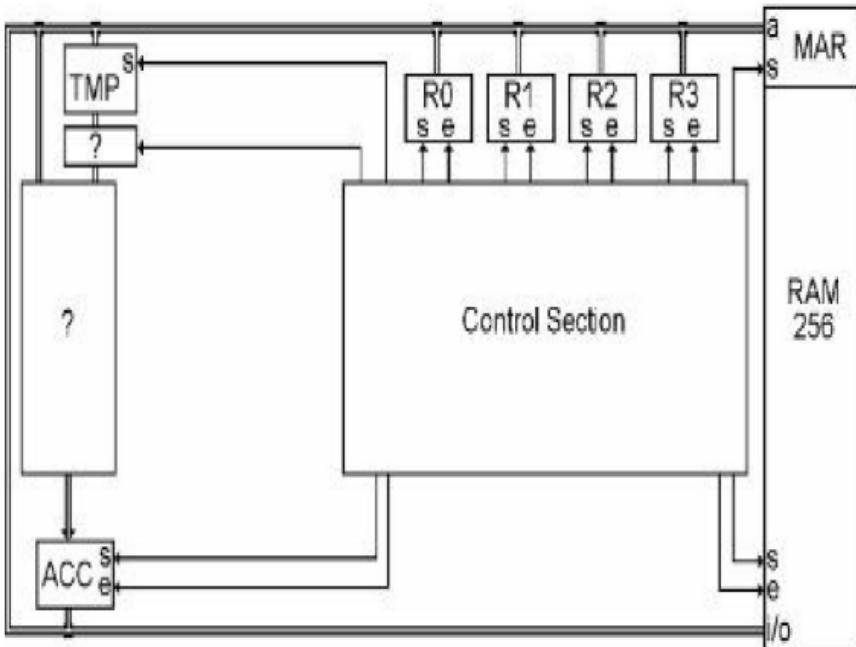
Adresses

Maintenant que nous avons le code binaire, nous pouvons l'utiliser à diverses fins dans notre ordinateur. L'un des premiers endroits où nous l'utiliserons est dans le registre d'adresses mémoire. Le modèle de bits que nous mettons dans ce registre utilisera le code binaire. Les bits de ce nombre dans MAR sélectionnent alors l'un des 256 emplacements de stockage RAM. Le nombre dans MAR est considéré comme un nombre quelque part entre 0 et 255, et donc chacun des 256 octets de RAM peut être considéré comme ayant une adresse.

C'est assez simple, mais il faut préciser ici exactement ce que l'on entend par une adresse à l'intérieur d'un ordinateur. Dans un quartier résidentiel, chaque maison a une adresse, comme le 125, rue Maple. Il y a un panneau au coin qui dit "Maple St." et écrit sur la maison sont les chiffres "125". C'est ainsi que nous pensons normalement aux adresses. Le point à souligner ici est que les maisons et les rues portent des numéros ou des noms écrits dessus. Dans l'ordinateur, l'octet ne contient aucune information d'identification ou n'y est contenu. C'est simplement l'octet qui est sélectionné lorsque vous mettez ce numéro dans le registre d'adresses mémoire. L'octet est sélectionné en fonction de son emplacement, et non de tout autre facteur contenu à cet emplacement. Imaginez un quartier de maisons qui avait seize rues et seize maisons sur chaque rue. Imaginez que les rues n'ont pas de panneaux et que les maisons n'ont pas de numéros écrits dessus. Vous seriez toujours en mesure de trouver une maison spécifique si on vous disait, par exemple, d'aller à « la quatrième maison de la septième rue ». Cette maison a toujours une adresse, c'est-à-dire une méthode pour la localiser, elle n'a tout simplement aucune information d'identification à l'emplacement. Ainsi, une adresse d'ordinateur n'est qu'un nombre qui provoque la sélection d'un certain octet lorsque cette adresse est placée dans le registre d'adresses mémoire.

L'autre moitié de l'ordinateur

L'autre moitié de l'ordinateur n'est également constituée en fin de compte que de portes NAND, et elle contient probablement moins de pièces au total que la RAM que nous avons construite, mais elle n'est pas disposée de manière aussi régulière et répétitive, il faudra donc un peu plus de temps pour expliquer . Nous appellerons cette moitié de l'ordinateur «l'unité centrale de traitement», ou CPU en abrégé, car elle fait quelque chose avec et sur les octets de la RAM. Il les « traite », et nous verrons ce que cela signifie dans les prochains chapitres. La chose qui est commune aux deux côtés de l'ordinateur est le bus.



Voici les débuts du CPU. La RAM est affichée à droite et le bus fait une grande boucle entre les deux connexions de bus sur la RAM. Le CPU démarre avec six

registres connectés au bus. Ces six registres sont tous les endroits que le CPU utilisera pour "traiter" les octets. Ce n'est pas si compliqué, n'est-ce pas ?

La grande case intitulée "Section de contrôle" au milieu du diagramme sera examinée en détail plus tard. Il contrôle tous les bits 'set' et 'enable' dans le CPU et la RAM. Les cases avec des points d'interrogation seront expliquées immédiatement après ce chapitre. Pour l'instant, nous allons regarder où les octets peuvent aller dans le CPU.

R0, R1, R2 et R3 sont des registres utilisés comme stockage à court terme pour les octets nécessaires au CPU. Leurs entrées et sorties sont connectées au bus. Ils peuvent être utilisés à de nombreuses fins différentes, ils sont donc appelés « registres à usage général ».

Le registre appelé 'TMP' signifie temporaire. Son entrée provient du bus et sa sortie descend vers l'une puis l'autre des cases marquées d'une question. TMP a un bit "set", mais pas de bit "enable" car nous n'avons jamais de raison de désactiver sa sortie.

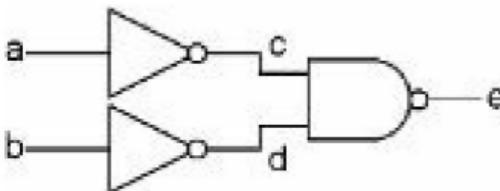
Le dernier registre est appelé l'accumulateur, ou ACC en abrégé. C'est un mot qui vient de l'époque des anciennes machines à additionner mécaniques (avant 1970). Je suppose que cela signifiait que lorsque vous additionniez une colonne de nombres, cela "accumulerait" un total cumulé. Dans un ordinateur, cela signifie simplement qu'il stocke temporairement un octet provenant de cette grande case marquée d'une question. La sortie de l'ACC est ensuite connectée à notre vieil ami, le bus, de sorte qu'elle peut être envoyée ailleurs au besoin.

Les registres du CPU et de la RAM sont les endroits d'où proviennent et vont le contenu des octets lorsque l'ordinateur fonctionne. Tous les mouvements impliquent l'activation d'un registre sur le bus et la définition du contenu du bus dans un autre registre.

Nous allons maintenant examiner ce qui se trouve dans ces cases avec les points d'interrogation.

Plus de portes

Nous avons utilisé les portes NAND, AND et NOT jusqu'à présent. Il y a deux autres portes de combinaison que nous devons définir. La premier est construit comme ceci:



Tout ce qu'il fait est de ne PAS les deux entrées à l'un de nos bons anciennes portes NAND. Voici le tableau pour cela, montrant le fils intermédiaires afin qu'il soit facile à suivre.

un	b	c	ré	et
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

Dans ce cas, lorsque les deux entrées sont désactivées, la sortie est éteint, mais si 'a' OU 'b' est allumé, ou les deux, alors le sortie sera activée. Donc, il a un autre nom très simple, c'est ce qu'on appelle la "porte OU". Au lieu de dessiner tous les parties, il a son propre diagramme en forme de quelque chose comme un bouclier. Le diagramme et le graphique ressemblent à ceci :



un	b	c
0	0	0
0	1	1

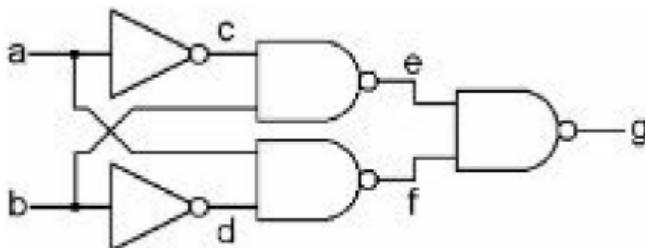
1	0		1
1	1		1

Comme la porte ET, vous pouvez construire des portes OU avec plus de deux entrées. Ajoutez simplement une autre porte OU à la place de l'une des les entrées, et vous aurez alors trois entrées, n'importe laquelle qui activera la sortie. Aussi comme le ET gate, chaque fois que vous ajoutez une entrée, le nombre de lignes sur le graphique va doubler. Avec la porte OU, seule la ligne dont toutes les entrées sont désactivées aura la sortie désactivée. Tout le reste des lignes montrera que la sortie est activée.

La dernière porte de combinaison dont nous avons besoin ici prend cinq portes à faire, mais ce qu'il fait finalement est assez simple.

Similaire à la porte OU, la sortie est activée lorsque l'un ou l'autre l'entrée est activée, mais dans cette version, la sortie revient éteint si les deux entrées sont allumées. C'est ce qu'on appelle une exclusivité Porte OU, ou porte XOR en abrégé. La sortie est activée si soit OU l'autre entrée est activée, exclusivement. Seulement s'il est OU, pas si c'est ET. Une autre façon de voir les choses est la sortie s'allume si une et une seule entrée est allumée.

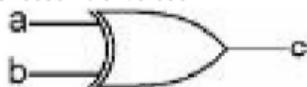
Encore une autre façon de voir les choses est que la sortie est désactivée si les entrées sont les mêmes, et allumé si les entrées sont différent.



un	b	c	ré	et	F	g
0	0	1	1	1	1	0

0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	1	1	0

Le schéma simplifié ressemble à une porte OU, mais il a une double ligne courbe du côté de l'entrée. La le diagramme et le graphique ressemblent à ceci :



a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Nous avons maintenant quatre types de portes qui prennent deux entrées et faire une sortie. Ce sont NAND, AND, OR et XOR. Voici un tableau qui le rend assez simple:

a	b	NAND	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0

Pour les quatre combinaisons d'entrées possibles de « a » et « b », chaque type de porte a son propre ensemble d'états de sortie, et les noms des portes peuvent vous aider à vous rappeler lequel est lequel.

Malgré le fait que tout dans l'ordinateur est fait de portes NAND, nous n'allons pas utiliser de portes NAND par elles-mêmes dans cet ordinateur ! Maintenant que nous les avons utilisés pour construire les portes ET, OU, XOR et NON, et le bit de mémoire, nous en avons fini avec la porte NAND. Merci M. NAND Gate, au revoir pour l'instant.

Jouer avec les octets

Les portes individuelles fonctionnent sur des bits. Deux bits dedans, un peu de moins. Mais la RAM stocke et récupère un octet à la fois.

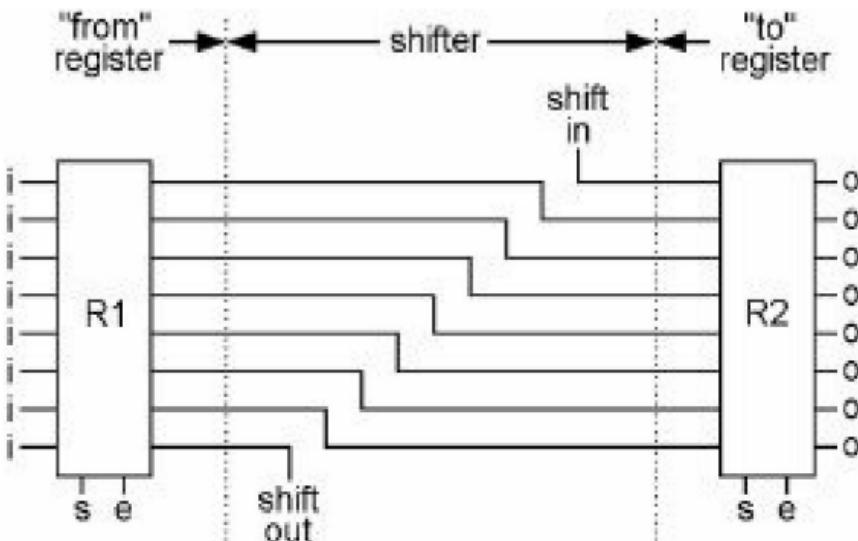
Et le bus se déplace d'un octet à la fois. Ici, dans le CPU, nous voulons pouvoir travailler sur un octet entier à la fois. Nous voulons des "portes" qui affectent un octet entier. Dans le chapitre sur le bus, nous avons vu comment le contenu d'un octet peut être copié d'un registre à l'autre. Ceci est généralement appelé déplacement d'un octet. Nous allons maintenant voir quelques variantes à ce sujet.

Nous allons d'abord voir trois manières de modifier le contenu d'un octet lorsqu'il passe d'un registre à un autre. Deuxièmement, nous verrons quatre façons de prendre le contenu de deux octets et de les faire interagir les uns avec les autres pour créer le contenu d'un troisième octet.

Ce sont toutes les choses que les ordinateurs font réellement aux octets. Tout se résume finalement à ces sept opérations.

Les manettes gauche et droite

Le levier de vitesses est très facile à construire. Il n'utilise aucune porte du tout, il câble juste le bus un peu bizarrement. Cela se fait comme ceci :



Cela montre deux registres reliés par un décalage à droite.

Le levier de vitesses n'est que les fils entre les deux registres.

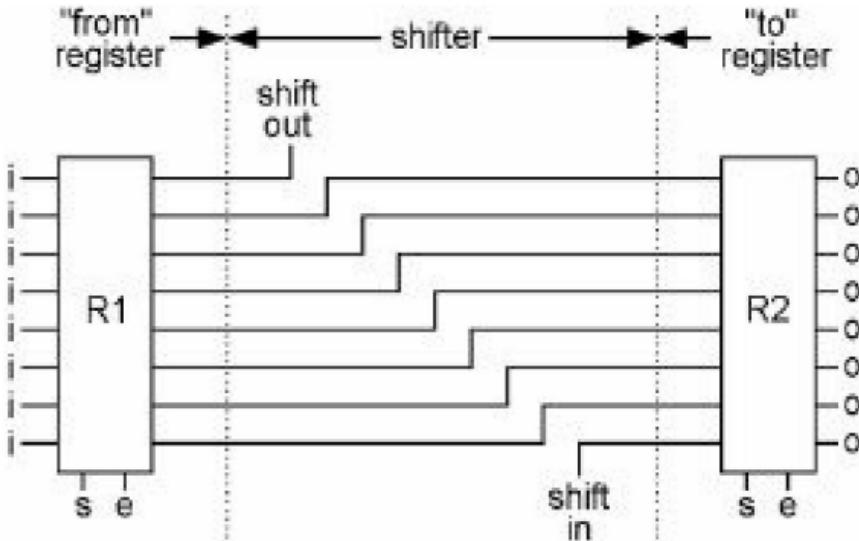
Lorsque le bit « e » de R1 est activé et que le bit « s » de R2 est activé puis désactivé, tous les bits de R1 sont copiés dans R2, mais ils sont décalés d'une position.

Celui du bas (shift out) peut être connecté à un autre bit de l'ordinateur, mais est souvent connecté à celui du haut (shift in) et lorsque cela est fait, le bit le plus à droite s'enroule vers le bit le plus à gauche à l'autre bout de l'octet.

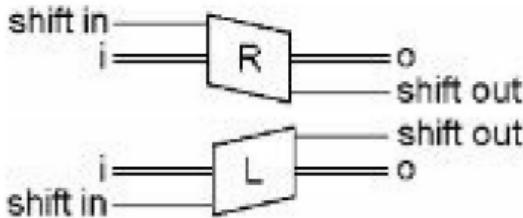
Un levier de vitesses droit changera 0100 0010 en 0010 0001.

Si 'shift out' est connecté à 'shift in', un shift droit changera 0001 1001 en 1000 1100 Un shifter gauche changera 0100 0010 en 1000 0100.

le levier de vitesses gauche est câblé comme ceci :



Encore une fois, nous avons des versions bus de ces dessins. Ils ont chacun un bus "i" et "o", ainsi qu'un bit d'entrée et de sortie, comme ceci :



Maintenant, à quoi cela sert-il ? L'esprit des programmeurs a imaginé toutes sortes de choses, mais en voici une intéressante. Supposons que vous utilisez le code binaire. Vous avez le numéro 0000 0110 en R1. Cela revient au nombre décimal 6. Maintenant, déplacez ce code vers la gauche dans R2. R2 sera alors 0000 1100. Cela revient au nombre décimal 12. Que savez-vous, nous venons de

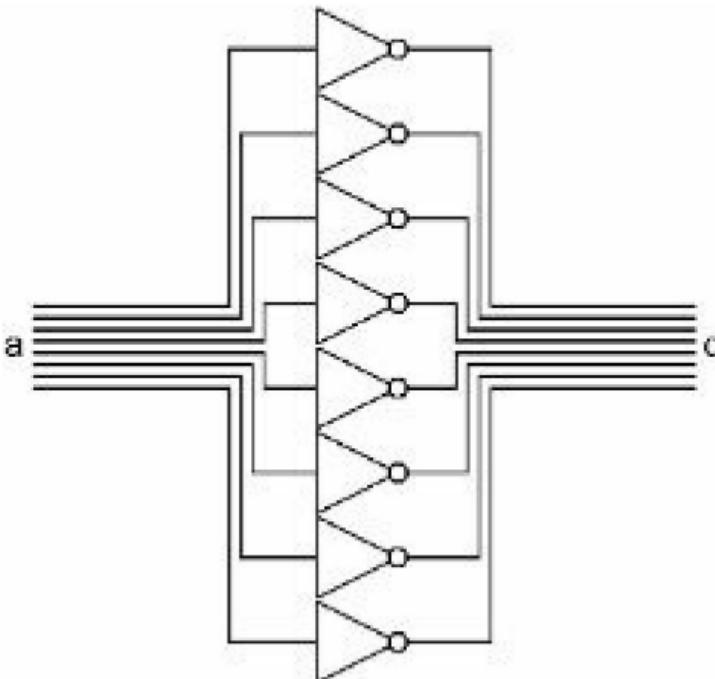
multiplié le nombre par 2. C'est la base de la façon dont la multiplication est effectuée dans notre ordinateur. On verra plus tard comment vous multipliez par autre chose que 2, mais c'est aussi simple que cela, il suffit de décaler les bits. Ceci est similaire à ce que nous faisons avec les nombres décimaux. Si vous voulez multiplier quelque chose par dix, il vous suffit d'ajouter un zéro sur le côté droit, décalant ainsi chaque chiffre d'une position vers la gauche. Dans le système binaire, cela ne fait que multiplier par deux car deux est ce sur quoi le système est basé.

C'est donc le levier de vitesses, pas de portes du tout.

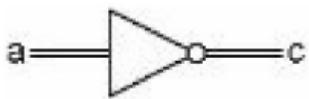
Le NOTter

Cet appareil connecte deux registres avec huit portes NOT.

Chaque bit sera remplacé par son opposé. Si vous commencez par 0110 1000, vous finirez par 1001 0111. Cette opération est utilisée à plusieurs fins, la première étant dans certaines fonctions arithmétiques. Nous verrons exactement comment cela fonctionne peu de temps après avoir décrit quelques autres choses. Un autre nom pour une porte NOT est un "inverseur", car il fait le contraire de ce que vous lui donnez, le retourne ou l'inverse.

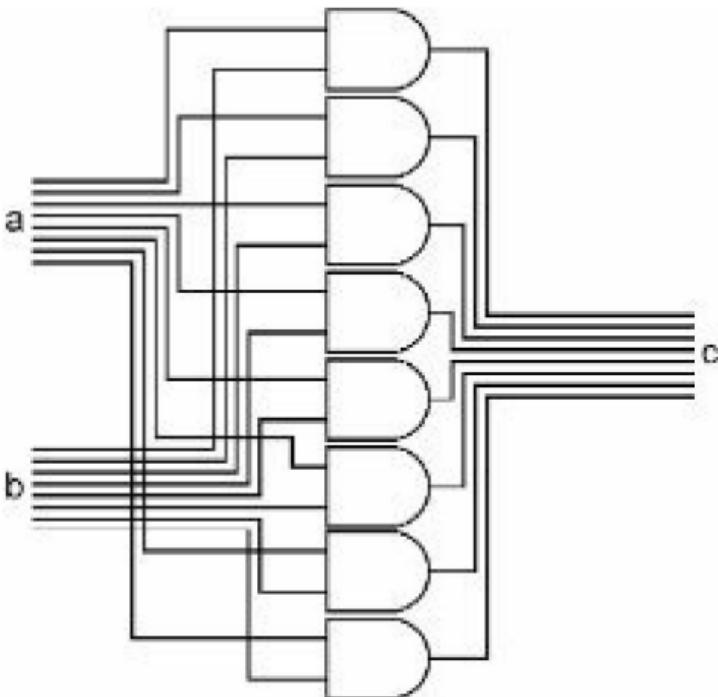


Étant donné que l'entrée et la sortie sont toutes deux à huit fils, nous allons simplifier en utilisant notre image de type bus.



L'ANDer

L'ANDer prend deux octets d'entrée, et ANDs chaque bit de ces deux dans un troisième octet. Comme vous pouvez le voir, les huit bits du bus d'entrée 'a' sont connectés au côté supérieur de huit portes ET. Les huit bits du bus d'entrée 'b' sont connectés au côté inférieur des mêmes huit portes ET. Les sorties des huit portes ET forment la sortie bus 'c' de cet ensemble. Avec cela, nous pouvons ET deux octets ensemble pour former un troisième octet.



Il existe de nombreuses utilisations pour cela. Par exemple, vous pouvez vous assurer qu'un code de lettre ASCII est en majuscule. Si vous avez le code pour la lettre 'e' dans R0, 0110 0101, vous pourriez

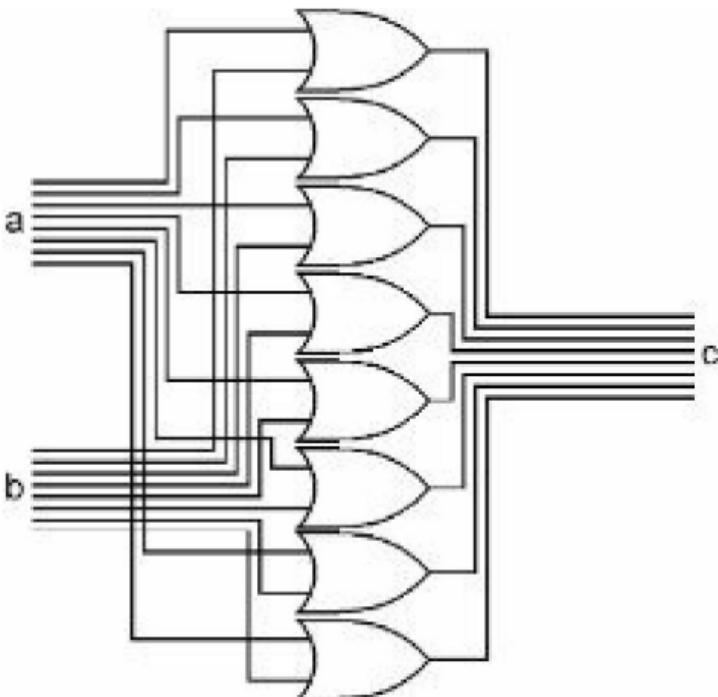
placez le motif binaire 1101 1111 dans R1 puis ET R1 et R0 et remettez la réponse dans R0. Tous les bits qui étaient activés dans R0 seront recopierés dans R0, à l'exception du troisième bit. Que le troisième bit ait été activé ou désactivé auparavant, il sera désormais désactivé. R0 contiendra désormais 0100 0101, le code ASCII pour 'E.' Cela fonctionne pour tous les codes de lettres ASCII en raison de la façon dont ASCII est conçu.

Voici une image de type bus plus simple pour l'ANDer.



L'OR

L'ORer prend deux octets d'entrée et ORs chaque bit de ces deux dans un troisième octet. Comme vous pouvez le voir, les huit bits du bus d'entrée 'a' sont connectés au côté supérieur de huit portes OU. Les huit bits du bus d'entrée 'b' sont connectés au côté inférieur des mêmes huit portes OU. Les sorties des huit portes OU sont la sortie du bus 'c' de cet ensemble. Avec cela, nous pouvons OR deux octets ensemble pour former un troisième octet.



Quelle est l'utilité de cela? Il y en a beaucoup, mais en voici un. Disons que vous avez le code ASCII pour la lettre 'E' dans R0, 0100 0101. Si vous voulez que cette lettre soit

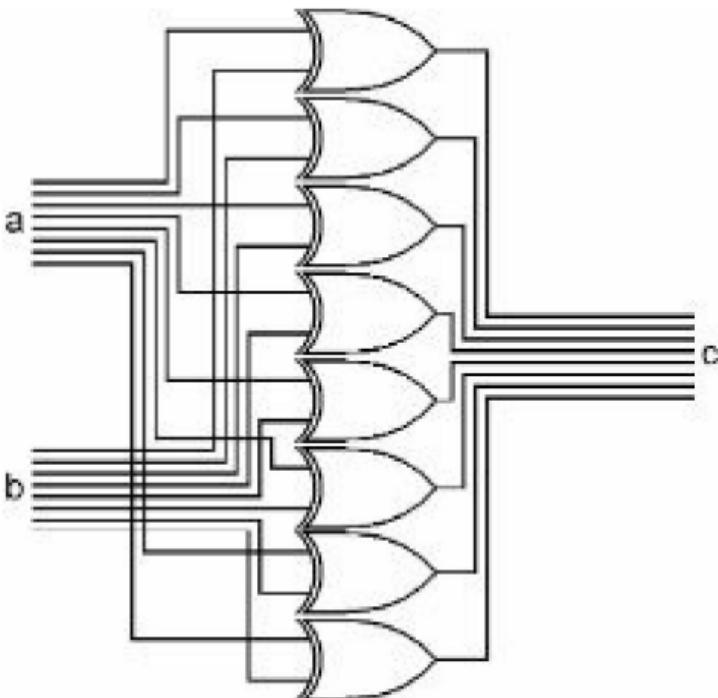
minuscules, vous pouvez mettre le modèle de bits 0010 0000 dans R1, puis OU R0 et R1 et remettre la réponse dans R0. Que va-t-il se passer ? Tous les bits de R0 seront recopierés dans R0 tels qu'ils étaient, sauf que le troisième bit sera désormais activé, peu importe ce qu'il était. R0 sera désormais 0110 0101, le code ASCII pour 'e.' Cela fonctionnera quel que soit le code de lettre ASCII dans R0 en raison de la façon dont ASCII a été conçu.

Voici une image de type de bus plus simple pour l'ORer.



L'OR exclusif

Le XORer prend deux octets d'entrée et XOR chaque bit de ces deux dans un troisième octet. Comme vous pouvez le voir, les huit bits du bus d'entrée 'a' sont connectés au côté supérieur de huit portes XOR. Les huit bits du bus d'entrée 'b' sont connectés au côté inférieur des mêmes huit portes XOR. Les sorties des huit portes XOR sont la sortie de bus 'c' de cet ensemble. Avec cela, nous pouvons XOR deux octets ensemble pour former un troisième octet.



Quelle est l'utilité de cela? Encore une fois, les personnes imaginatives ont trouvé de nombreuses utilisations. Mais en voici un. Disons que vous avez un code dans R1 et un autre code dans R2. Tu veux

pour savoir si ces deux codes sont identiques. Donc vous XOR R1 et R2 dans R1. Si R1 et R2 contenaient les mêmes modèles, alors R1 sera désormais tous des zéros. Peu importe quel modèle de 0 et de 1 était dans R1, si R2 contenait le même modèle, après un XOR, R1 sera tout

des zéros.

Étant donné que les entrées et la sortie sont toutes deux à huit fils, nous allons simplifier en utilisant notre image de type de bus.



L'additionneur

Il s'agit d'une combinaison de portes qui est étonnamment simple compte tenu de ce qu'elle fait. Dans notre système de numération binaire, nous avons des nombres compris entre 0 et 255 représentés dans un octet. Si vous pensez à la façon dont l'addition est faite avec deux de nos nombres décimaux réguliers, vous commencez par additionner les deux nombres dans la colonne de droite.

Étant donné que les deux nombres pourraient chacun être n'importe où entre 0 et 9, la somme de ces deux sera quelque part entre 0 et 18.

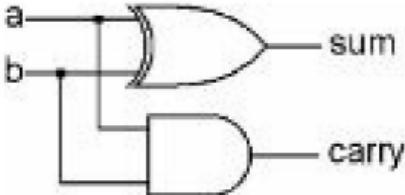
Si la réponse est comprise entre 0 et 9, vous l'écrivez sous les deux chiffres. Si la réponse est de 10 à 18, vous écrivez le bon chiffre, et portez le 1 à ajouter à la colonne suivante à gauche.

$$\begin{array}{r} 2 + 4 \\ \hline 6 \end{array} \qquad \begin{array}{r} 5 \\ + 7 \\ \hline 12 \end{array}$$

Dans le système de numération binaire, c'est en fait beaucoup plus simple. Si vous faites le même type d'addition en binaire, les deux nombres de la colonne de droite ne pourront chacun être que 0 ou 1. Ainsi les seules réponses possibles pour additionner la colonne de droite de deux nombres binaires seront 0, 1 ou 10 (zéro, un ou deux). Si vous additionnez 0+0, vous obtenez 0, 1+0 ou 0+1 vous obtenez 1, 1+1 vous obtenez 0 dans la colonne de droite, et portez 1 dans la colonne de gauche.

$$\begin{array}{r} 1 + 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ + 1 \\ \hline 10 \end{array}$$

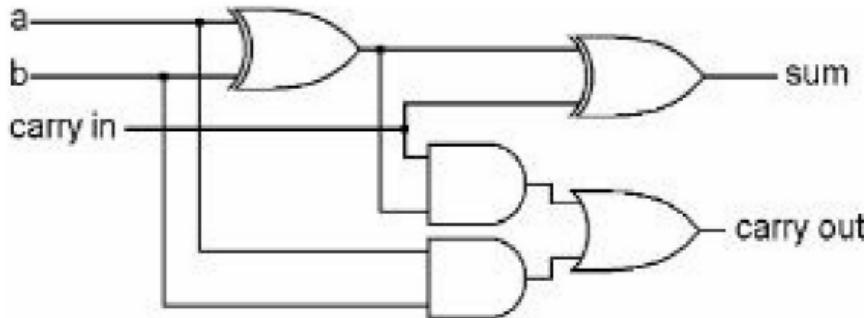
Les portes que nous avons décrites peuvent facilement le faire. Un XOR des deux bits nous dira quelle devrait être la réponse de la colonne de droite, et un AND des deux bits nous dira si nous devons porter un 1 à la colonne de gauche. Si un bit est activé et l'autre désactivé, c'est-à-dire que nous ajoutons un 1 et un 0, la réponse pour la colonne de droite sera 1. Si les deux nombres sont 1, ou si les deux nombres sont 0, la colonne de droite sera 0. La porte ET ne s'active que dans le cas où les deux numéros d'entrée sont 1.



Nous avons donc ajouté la colonne de droite facilement. Maintenant, nous voulons ajouter la colonne suivante à gauche. Ça devrait être pareil, droit? Il y a deux bits qui pourraient être 0 ou 1, mais ceci temps, nous avons également la possibilité d'un report de la colonne précédente. Donc ce n'est pas pareil, cette fois nous sommes ajoutant vraiment trois nombres, les deux bits dans ce colonne, plus le report possible de la précédente colonne.

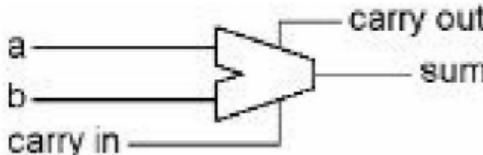
Porter->	0	1	0	1 1
	00	01	dix	011
	<u>+01</u>	<u>+01</u>	<u>+01</u>	<u>+011</u>
	01	dix	11	110

Lors de l'addition de trois bits, les réponses possibles sont 0, 1, 10 ou 11 (zéro, un, deux ou trois.) C'est plus complexe, mais pas impossible. Voici la combinaison qui fait ce:

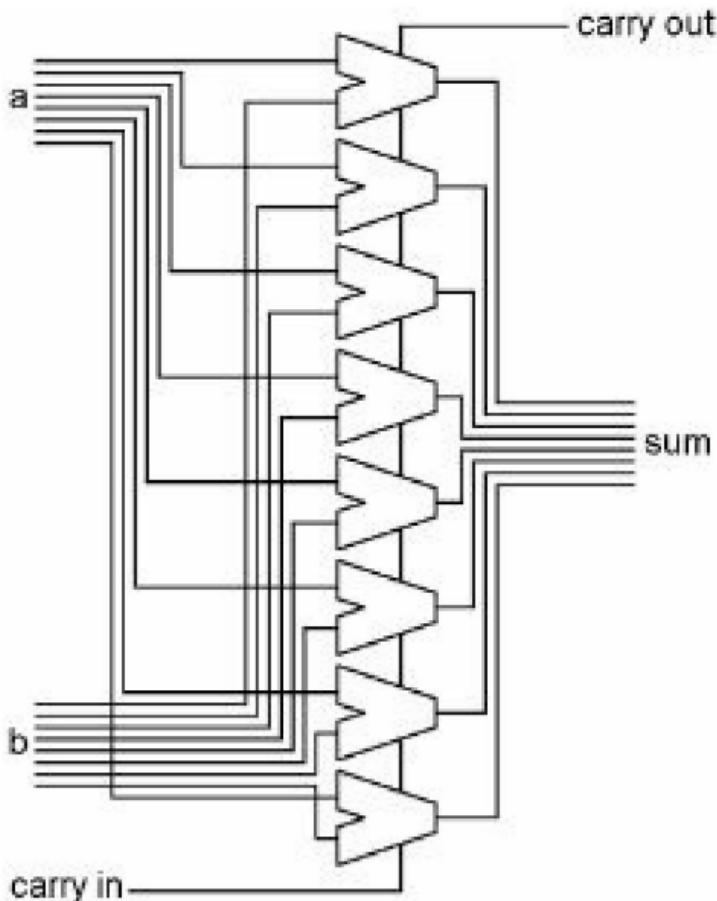


Le XOR gauche nous indique si 'a' et 'b' sont différents. S'ils le sont et que 'carry in' est désactivé, ou si 'a' et 'b' sont identiques et 'carry in' est activé, alors le XOR droit générera 1 comme somme pour la colonne actuelle. La porte ET inférieure s'activera "exécuter" si les deux entrées sont activées.

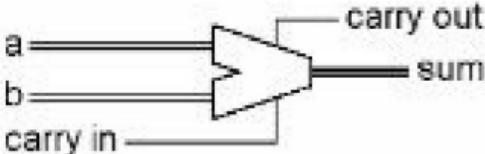
La porte ET supérieure s'activera 'effectuer' si 'faire entrer' et l'une des entrées sont activées. C'est la simplicité de la façon dont les ordinateurs font l'addition ! Maintenant que nous voyons que cela fonctionne, nous pouvons en faire une image plus simple :



Pour créer un additionneur qui additionne deux octets ensemble, nous avons besoin d'un additionneur d'un bit pour chaque bit des octets, avec la sortie de report de chaque bit connectée à l'entrée de report du suivant. Notez que chaque bit a un report, même le premier bit (la colonne de droite). Ceci est utilisé lorsque nous voulons ajouter des nombres qui peuvent être supérieurs à 255.



Et l'image simplifiée de celui-ci avec les entrées et sorties de bus :



Le bit de sortie de report de la colonne la plus à gauche (en haut) s'activera si la somme des deux nombres est supérieure à 255, et ce bit sera utilisé ailleurs dans l'ordinateur.

C'est ainsi que les ordinateurs font l'addition, juste cinq portes par bit, et l'ordinateur peut faire de l'arithmétique !

Le comparateur et zéro

Toutes les choses que nous avons décrites ci-dessus prennent un ou deux octets en entrée et génèrent un octet en sortie. Les décaleurs et l'additionneur génèrent également un bit de sortie supplémentaire lié à leur octet de sortie. Le comparateur ne génère que deux bits de sortie, pas un octet entier.

Le comparateur est en fait intégré directement dans le XORer car il peut utiliser les portes qui sont déjà là. Le XORer génère son octet de sortie, et le comparateur génère ses deux bits. Ces deux fonctions ne sont pas vraiment liées l'une à l'autre, il se trouve juste qu'il est facile de le construire de cette façon.

Ce que nous voulons que le comparateur fasse, c'est savoir si les deux octets sur le bus d'entrée sont exactement égaux, et sinon, si celui sur le bus 'a' est plus grand selon le système de nombre binaire.

La chose égale est assez simple. Les portes XOR s'éteignent lorsque les entrées sont identiques, donc si toutes les portes XOR sont éteintes, alors les entrées sont égales.

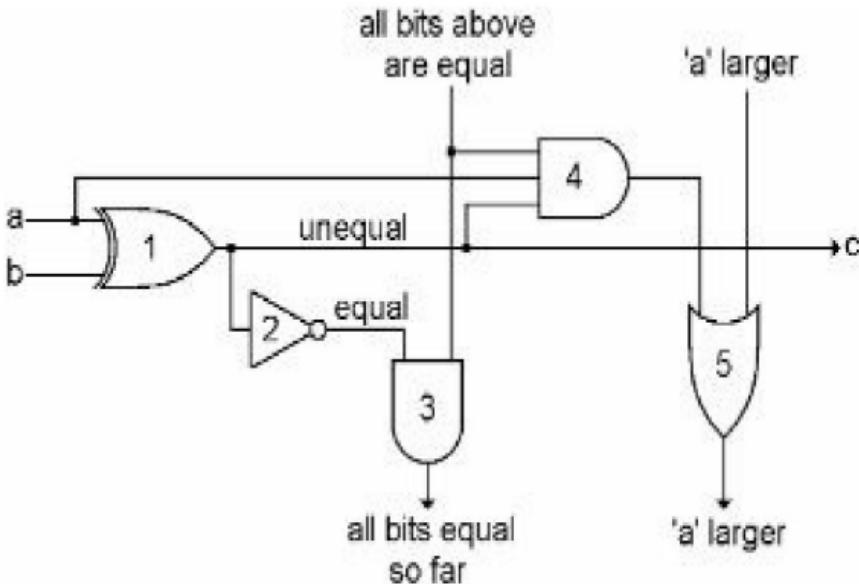
Déterminer le plus grand de deux nombres binaires est un peu plus délicat. Vous devez commencer par les deux bits supérieurs, et si l'un est allumé et l'autre éteint, alors celui qui est allumé est le plus grand nombre. S'ils sont identiques, vous devez vérifier la prochaine paire de bits inférieure, etc., jusqu'à ce que vous trouviez une paire où ils sont différents.

Mais une fois que vous avez trouvé une paire différente, vous ne voulez plus vérifier de bits. Par exemple, 0010 0000 (32) est supérieur à 0001 1111 (31). Les deux premiers bits sont identiques dans les deux octets. Le troisième bit est activé dans le premier octet et désactivé dans le second, et par conséquent le premier octet est plus grand. Bien que les autres bits soient activés dans le deuxième octet, leur total est inférieur au bit activé dans le premier octet.

C'est ce que nous voulons qu'il se produise, et il faut cinq portes multipliées par huit positions pour y parvenir. Puisque nous commençons avec le XORer, nous ajouterons quatre autres portes à chaque position, comme indiqué dans ce diagramme. Rappelez-vous que dans l'additionneur, nous avions un bit de report qui passait de la position de bit la plus basse jusqu'au bit le plus élevé. Dans

Le comparateur, nous avons deux bits qui passent de la position de bit la plus élevée à la plus basse.

Voici une partie du comparateur. Vous pouvez voir la porte XOR d'origine, étiquetée '1', connectée à un bit de chaque bus d'entrée à gauche et générant un bit pour le bus de sortie à droite.



Si la sortie de la porte 1 est allumée, cela signifie que 'a' et 'b' sont différents, ou inégaux. Nous ajoutons la porte 2, qui s'activera lorsque 'a' et 'b' sont égaux.

Si la porte 2 est activée à chaque position, la porte 3 sera également activée à chaque position, et le bit qui sort du bas nous indique que les deux octets d'entrée sont égaux.

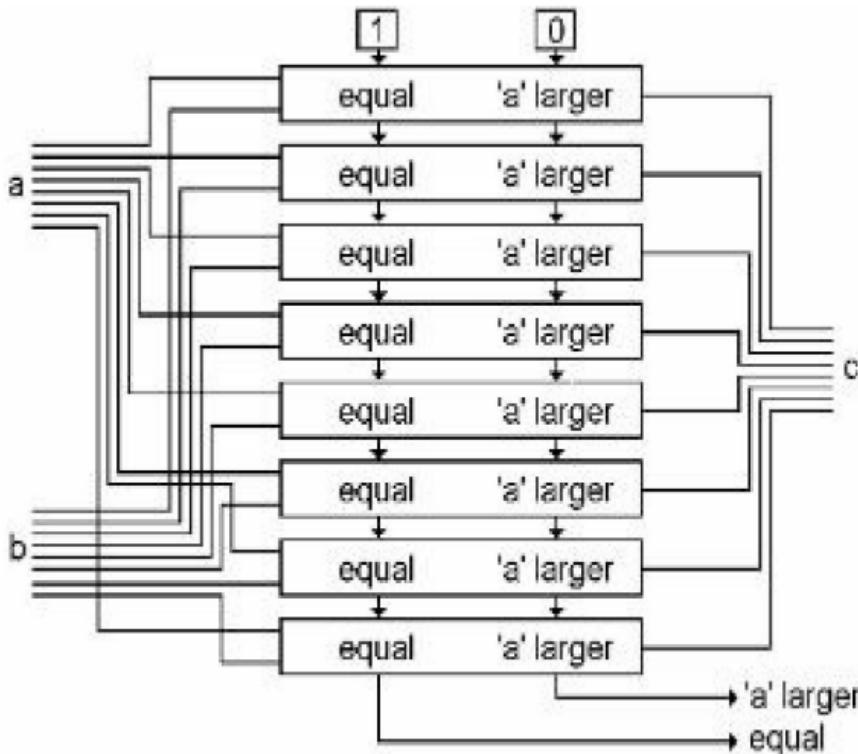
La porte 4 s'allume si trois choses sont vraies. 1) Les bits 'a' et 'b' sont différents. 2) Le bit 'a' est celui qui est activé.

3) Tous les bits au-dessus de ce point ont été égaux. Lorsque la porte 4 s'allume, elle allume la porte 5, et cela allume toutes les autres portes 5 en dessous de ce point, et donc le 'a'

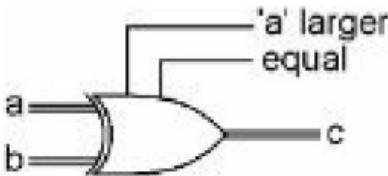
sortie plus grande du comparateur.

Lorsque l'octet 'b' est le plus grand, le bit 'égal' et le bit 'a plus grand' seront désactivés.

Vous empilez huit de ces comparateurs de bits comme le schéma suivant, avec un « 1 » et un « 0 » connectés à celui du haut pour le démarrer. Vous avez toujours la fonction XOR qui sort en 'c', et maintenant les deux bits du comparateur en bas.

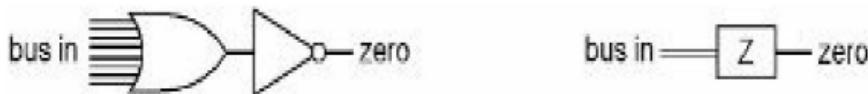


En simplifiant encore, nous allons reprendre le schéma XOR de type bus, et ajouter simplement les deux nouveaux bits de sortie du comparateur.



Il y a encore une chose dont nous aurons besoin dans notre ordinateur qui nous donne une autre information. Il s'agit d'une simple combinaison de portes qui prend un octet entier en entrée et ne génère qu'un seul bit en sortie. Le bit de sortie est activé lorsque tous les bits de l'octet sont désactivés.

En d'autres termes, le bit de sortie nous indique quand le contenu de l'octet est entièrement nul.



Il s'agit simplement d'une porte OU à huit entrées et d'une porte NON. Lorsque l'une des entrées de la porte OU est activée, sa sortie sera activée et la sortie de la porte NON sera désactivée. Ce n'est que lorsque les huit entrées du OU sont désactivées, et que sa sortie est donc désactivée, que la sortie de la porte NOT sera activée.

La représentation plus simple du bus est illustrée à droite.

Logique

Le sujet de la pensée a fait l'objet de nombreuses études et spéculations à travers les âges. Il y avait un homme dans la Grèce antique nommé Aristote qui a fait beaucoup de travail dans ce domaine. Il a dû rencontrer beaucoup de gens illogiques dans sa vie, car il a inventé un sujet censé aider les gens à penser plus合理ement.

L'une de ses idées était que si vous avez deux faits, vous pourriez être en mesure de dériver un troisième fait des deux premiers. À l'école, ils donnent parfois des tests qui présentent deux faits, puis ils vous donnent un troisième fait et demandent si le troisième fait est « logique » basé sur les deux premiers. Vous vous souvenez peut-être de questions telles que :

Si Joe est plus vieux que Bill, Et Fred est plus vieux que Joe, Alors Fred est plus vieux que Bill.
Vrai ou faux?

Ou

Les enfants aiment les bonbons.
Jeanne est une enfant.
Par conséquent, Jane aime les bonbons. Vrai ou faux?
Aristote a appelé son étude de ce genre de choses « Logique ».

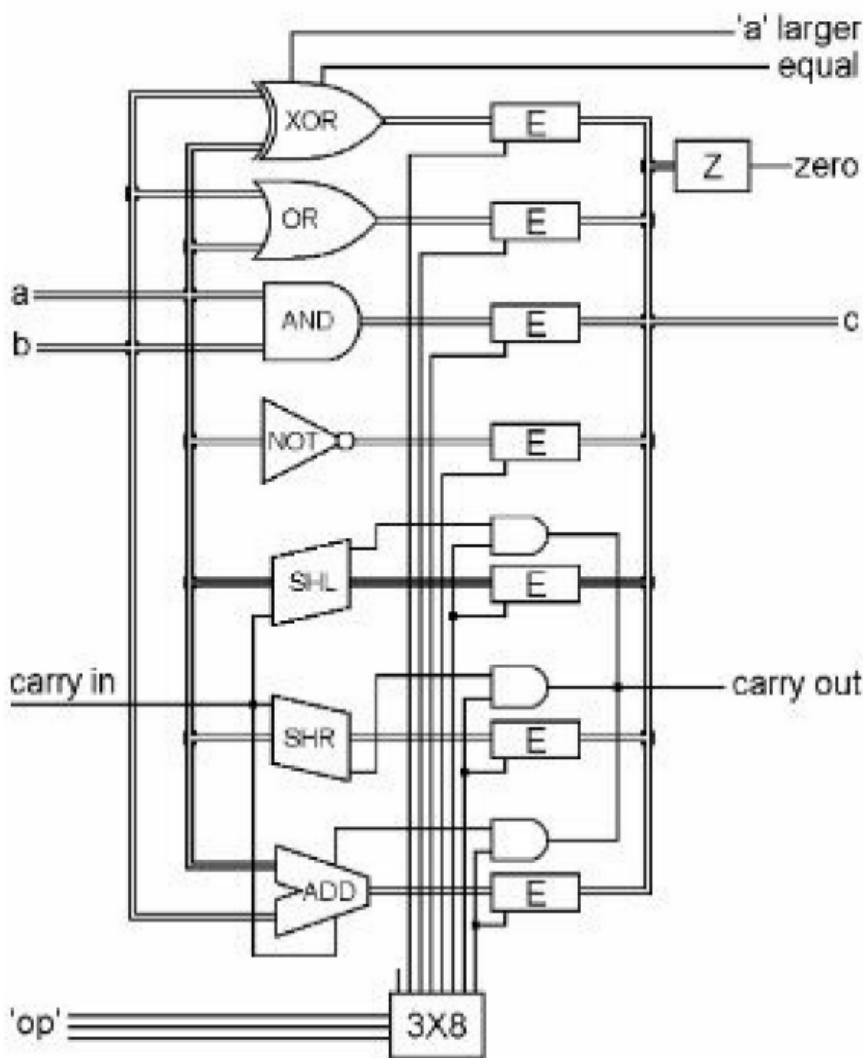
La seule pertinence que cela a pour notre discussion sur les ordinateurs est ce mot « logique ». La logique d'Aristote impliquait deux faits faisant un troisième fait. Beaucoup de nos composants informatiques, tels que les portes ET, prennent deux bits et font un troisième bit, ou huit portes ET prennent deux octets et font un troisième octet. Et ainsi, les choses que font ces portes sont connues sous le nom de logique. Il peut y avoir une logique ET et une logique OU et une logique XOR, mais le terme général pour tous est la logique.

ANDing, ORing et XORing prennent deux octets pour en faire un troisième, ils correspondent donc assez bien à cette définition de la logique.
Le décalage et le NOTing sont également connus sous le nom de logique même s'ils ne prennent qu'un octet d'entrée pour générer leur sortie. L'ADDer, bien qu'il ait deux entrées et qu'il soit également très logique, n'est pas connu pour être dans la catégorie de la logique, il est dans sa propre catégorie, l'arithmétique.

Ainsi, toutes les façons que nous avons décrites ci-dessus de faire des choses avec des octets relèvent de la rubrique "arithmétique et logique".

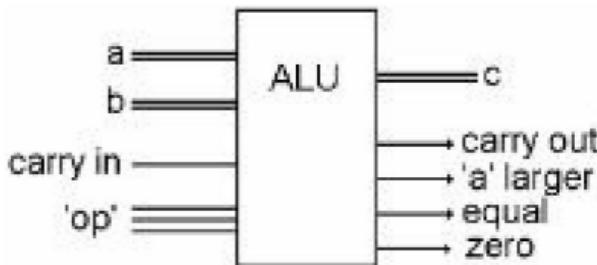
L'unité d'arithmétique et de logique

Maintenant, nous avons construit sept appareils différents qui peuvent faire de l'arithmétique ou de la logique sur des octets de données. Nous allons prendre ces sept appareils, les assembler en une seule unité et fournir une méthode pour sélectionner lequel de ces appareils nous voulons utiliser à un moment donné. C'est ce qu'on appelle «l'unité arithmétique et logique», ou «ALU» en abrégé.



Les sept appareils sont connectés à l'entrée 'a', les appareils qui ont deux entrées sont également connectés à l'entrée 'b'. Les sept appareils sont connectés aux entrées à tout moment, mais chaque sortie est reliée à l'un de ces activateurs. Les fils qui allument les activateurs sont connectés aux sorties d'un décodeur, ainsi un seul activateur peut être allumé à un moment donné. Sept des sorties du décodeur permettent à un seul appareil de continuer sur la sortie commune, « c ». La huitième sortie du décodeur est utilisée lorsque vous ne souhaitez sélectionner aucun appareil. Les trois fils d'entrée du décodeur sont étiquetés « op », car ils choisissent le « fonctionnement » souhaité. La seule petite complication ici est les bits de report de l'additionneur et les bits de « décalage entrant » et de « décalage sortant » des décaleurs. Ceux-ci sont utilisés de manière très similaire, et donc à partir de maintenant, nous les appellerons tous des bits de report. L'additionneur et les deux décaleurs prennent le report en entrée et génèrent le report en sortie. Ainsi, les trois entrées de report sont connectées à une seule entrée ALU, et l'une des trois sorties est sélectionnée avec la sortie de bus de son appareil. Regardez la sortie la plus à droite du décodeur 3X8 ci-dessus et voyez qu'elle active à la fois le bus d'additionneur et le bit de report d'additionneur.

Qu'avons-nous ici ? C'est une boîte qui a deux entrées de bus, une sortie de bus et quatre autres bits d'entrée et quatre autres bits de sortie. Trois des bits d'entrée sélectionnent quelle "opération" aura lieu entre les bus d'entrée et de sortie. Encore une fois, maintenant que nous savons ce qu'il contient et comment cela fonctionne, nous n'avons pas besoin de montrer toutes ses parties. Voici une manière simplifiée de le dessiner :



Notez que les trois entrées à bit unique étiquetées "op",

ci-dessus, peut avoir huit combinaisons différentes. Sept de ces combinaisons sélectionnent l'un des appareils décrits précédemment. La huitième combinaison ne sélectionne aucun octet de sortie, mais les bits "a plus grand" et "égal" sont toujours travailler, comme ils le font à tout moment, donc c'est le code à choisissez si vous voulez seulement faire une comparaison.

La combinaison de bits à 'op' signifie quelque chose. Ce ressemble à un autre code. Oui, voici un code à trois bits que nous utiliserons bientôt.

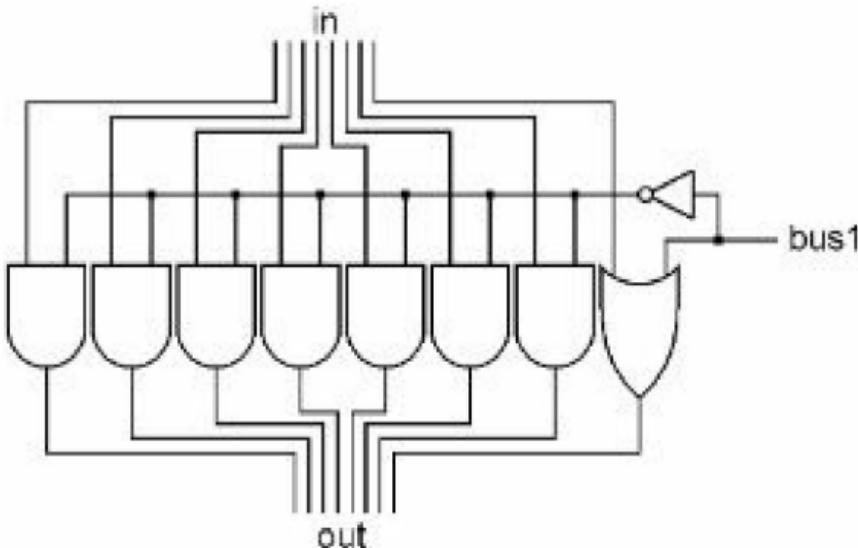
000	AJOUTER	Ajouter
001	SHR	Décalage à droite
010	SHL	Décalage à gauche
011	NE PAS	Pas
100	ET	Et
101	OU	Ou
110	LIBRE	OU exclusif
111	CMP	Comparer

L'unité d'arithmétique et de logique est le centre même, le cœur de l'ordinateur. C'est là que toute l'action arrive. Je parie que c'est beaucoup moins compliqué que tu pensais.

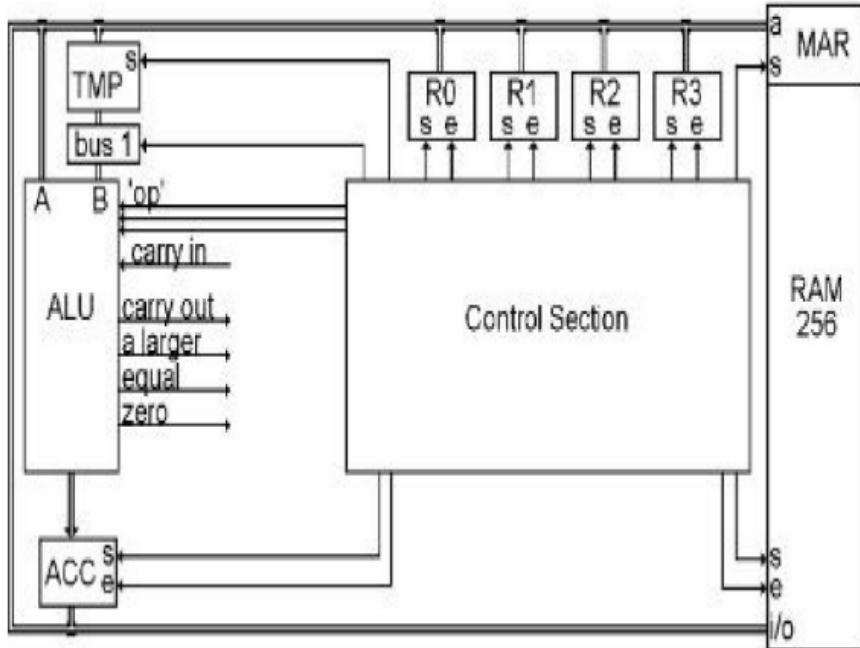
Plus du processeur

Il y a encore un petit appareil dont nous avons besoin. C'est une chose très simple, il a une entrée de bus, une sortie de bus et un autre bit d'entrée. C'est très similaire à un activateur. Sept des bits passent par des portes ET et l'un d'eux passe par une porte OU. L'entrée à bit unique détermine ce qui se passe lorsqu'un octet tente de traverser ce périphérique.

Lorsque le bit 'bus 1' est désactivé, tous les bits du bus d'entrée passent sans changement au bus de sortie. Lorsque le bit 'bus 1' est activé, l'octet d'entrée est ignoré et l'octet de sortie sera 0000 0001, qui est le nombre 1 en binaire. Nous appellerons cet appareil un "bus 1" car il placera le numéro 1 sur un bus lorsque nous en aurons besoin.



Nous pouvons maintenant mettre ce "bus 1" et l'ALU dans le CPU. Nous allons changer l'endroit où les fils entrent et sortent de l'ALU afin qu'il corresponde mieux à notre schéma. Les entrées du bus viennent en haut, la sortie du bus sort en bas et tous les bits d'entrée et de sortie sont à droite.



La sortie de l'ALU est connectée à ACC. L'ACC reçoit et stocke temporairement le résultat de l'opération ALU la plus récente. La sortie de l'ACC est ensuite connectée au bus, de sorte que son contenu peut être envoyé ailleurs si nécessaire.

Lorsque nous voulons effectuer une opération ALU à une entrée, nous devons définir les trois bits "op" de l'ALU sur l'opération souhaitée, activer le registre souhaité sur le bus et définir la réponse dans ACC.

Pour une opération ALU à deux entrées, il y a deux étapes.

Nous activons d'abord l'un des registres sur le bus et le mettons en TMP. Ensuite, nous activons le deuxième registre sur le bus, choisissons l'opération ALU et définissons la réponse dans ACC.

Comme vous pouvez le voir, nous pouvons maintenant déplacer des octets de données vers et

de la RAM, vers et depuis les registres, via l'ALU vers l'ACC, et de là, dans un registre ou une RAM si nous activons et désactivons les bits appropriés au bon moment. C'est ce qui se passe à l'intérieur des ordinateurs.

Ce n'est pas si compliqué, n'est-ce pas ?

Il ne manque qu'une seule chose ici, et cela a à voir avec tous ces bits de contrôle sur les registres, ALU et RAM. La RAM a trois bits de contrôle, un pour définir MAR, un pour définir l'octet actuellement sélectionné en entrée, un pour activer l'octet actuellement sélectionné en sortie. Chacun des registres, R0, R1, R2, R3 et ACC ont un bit défini et un bit d'activation, TMP n'a qu'un bit défini, le bus 1 a un bit de contrôle et l'ALU a ces trois bits "op" qui sélectionnent le bit souhaité. opération.

Nous avons besoin de quelque chose qui activera et désactivera tous ces bits de contrôle aux moments appropriés afin que nous puissions faire quelque chose d'utile. Il est maintenant temps de regarder dans cette boîte intitulée "Section de contrôle".

L'horloge

Nous devons activer et désactiver les bits de contrôle appropriés aux moments appropriés. Nous examinerons les éléments appropriés plus tard, d'abord nous examinerons les moments appropriés.

Voici un nouveau type de dessin, nous l'appellerons un graphe.

Il montre comment un bit change avec le temps. Le temps commence à gauche et avance vers la droite. La ligne sur le graphique a deux positions possibles, vers le haut signifie que le bit est activé et vers le bas signifie que le bit est désactivé.



Ce graphique montre que le bit 'X' s'allume et s'éteint, s'allume et s'éteint régulièrement. Il pourrait y avoir une échelle de temps en bas pour montrer à quelle vitesse cela se produit. Si toute la largeur de la page représentait une seconde, alors le bit 'X' s'allumerait et s'éteindrait environ huit fois par seconde. Mais nous n'aurons pas besoin d'une échelle de temps dans ces graphiques, car nous ne nous intéresserons qu'à la synchronisation relative entre deux bits ou plus. La vitesse d'un ordinateur réel sera très rapide, comme le bit qui s'allume et s'éteint un milliard de fois par seconde.

Quand quelque chose répète régulièrement une action, l'une de ces actions, individuellement, s'appelle un cycle. Le graphique ci-dessus montre environ huit cycles. Vous pouvez dire qu'à partir d'un moment où le bit s'allume jusqu'à la fois suivante, le bit s'allume est un cycle, ou vous pouvez dire que du milieu du temps d'arrêt au milieu du prochain temps d'arrêt est le cycle, tant que le cycle commence à un moment donné lorsque le bit est à un certain stade de son activité, et se poursuit jusqu'à ce que le bit revienne à nouveau au même stade de l'activité. Le mot "Cycle" vient du mot " cercle", donc quand le bit fait un cercle complet, c'est un cycle.

Il y avait un scientifique qui vivait en Allemagne dans les années 1800 qui a fait certaines des premières recherches qui ont conduit à la radio.

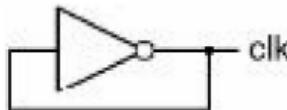
Il s'appelait Heinrich Hertz et, entre autres, il

étudié l'électricité qui s'allumait et s'éteignait très rapidement. Quelques décennies après sa mort, il a été décidé d'utiliser son nom pour décrire la vitesse à laquelle l'électricité s'allumait et s'éteignait, ou combien de cycles se produisaient par seconde.

Ainsi, un Hertz (ou Hz en abrégé) signifie que l'électricité s'allume et s'éteint une fois par seconde. 500 Hz signifie 500 fois par seconde. Pour des vitesses plus rapides, nous rencontrons à nouveau ces langues anciennes, et mille fois par seconde s'appelle un kilohertz ou kHz en abrégé.

Allumer et éteindre un million de fois par seconde s'appelle un mégahertz, ou mHz en abrégé, et un milliard de fois s'appelle un gigahertz, ou GHz en abrégé.

Chaque ordinateur a un bit spécial. Tous les autres bits d'un ordinateur viennent de quelque part, ils sont activés et désactivés par d'autres bits ou commutateurs. Ce bit spécial s'allume et s'éteint tout seul. Mais il n'y a rien de mystérieux là-dedans, il s'allume et s'éteint très régulièrement et très rapidement. Ce bit spécial est construit très simplement, comme ceci :

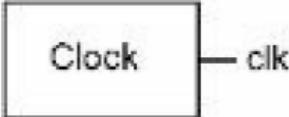


Cela semble une chose stupide à faire. Reconnectez simplement la sortie d'une porte NOT à son entrée ? Qu'est-ce que cela fera ?

Eh bien, si vous commencez avec la sortie activée, l'électricité retourne à l'entrée, où elle entre dans la porte qui désactive la sortie, qui retourne à l'entrée qui active la sortie. Oui, cette porte s'ouvrira et se fermera aussi rapidement que possible. Ce sera en fait trop rapide pour être utilisé pour quoi que ce soit, et il peut donc être ralenti simplement en allongeant le fil qui fait la boucle.



Le schéma simplifié montre qu'il s'agit du seul bit spécial de l'ordinateur qui a une sortie mais qui n'a aucune entrée.



Ce bit a un nom. Elle s'appelle l'horloge. Maintenant, nous considérons généralement une horloge comme une chose avec un cadran et des aiguilles, ou des chiffres sur un écran, et nous avons vu de telles horloges dans le coin d'un écran d'ordinateur.

Malheureusement, quelqu'un a nommé ce type de bit, une horloge, et le nom est resté avec les pionniers de l'informatique. Cela aurait pu s'appeler le battement de tambour ou le rythmeur ou le cœur ou la section rythmique, mais ils l'appelaient une horloge.

C'est ce que nous voulons dire quand nous disons horloge tout au long de ce livre. Je suppose que c'est une horloge qui fait tic-tac, mais qui n'a pas de cadran. Si nous voulons parler du type d'horloge qui vous indique l'heure qu'il est, nous l'appellerons une «horloge de l'heure du jour» ou «horloge TOD» en abrégé.

Mais le mot 'horloge' désignera ce type de bit.

À quelle vitesse cette horloge s'allume-t-elle et s'éteint-elle ? De nos jours, c'est bien plus d'un milliard de fois par seconde, soit plusieurs gigahertz. C'est l'une des principales caractéristiques dont les sociétés informatiques vous parlent pour vous montrer à quel point leurs ordinateurs sont performants. Lorsque vous voyez des ordinateurs à vendre, la vitesse qu'ils annoncent est la vitesse de son horloge.

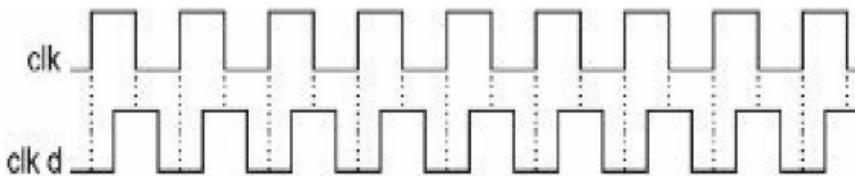
Plus un ordinateur est rapide, plus il est cher, car il peut faire plus de choses en une seconde. C'est la vitesse à laquelle ce bit unique s'allume et s'éteint qui définit le tempo de tout l'ordinateur.

Pour déplacer des données via le bus, nous devons d'abord activer la sortie d'un et un seul registre, afin que son électricité puisse voyager à travers le bus vers les entrées d'autres registres. Ensuite, pendant que les données sont sur le bus, nous voulons activer et désactiver le bit défini du registre de destination. Étant donné que le registre de destination capture l'état du bus à l'instant où le bit défini s'éteint, nous voulons nous assurer qu'il s'éteint avant de désactiver le bit d'activation du premier registre pour nous assurer qu'il n'y a pas de problèmes.

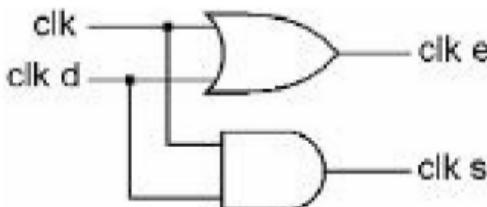
Attachons d'abord une longueur de fil à la sortie de l'horloge. Cela retardera légèrement l'électricité. Nous voulons qu'il soit retardé d'environ un quart de cycle.



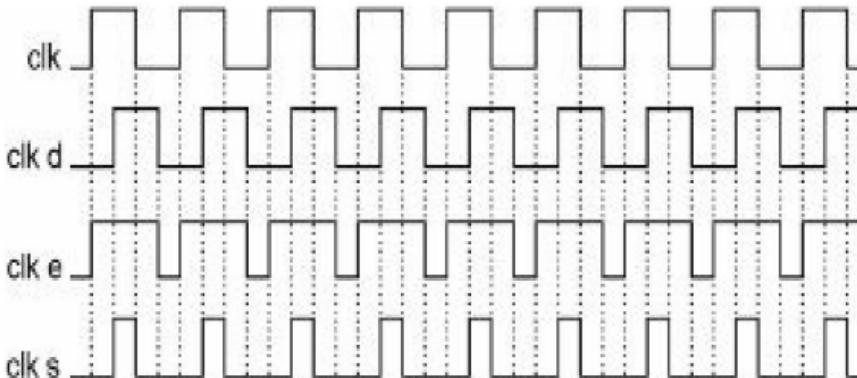
Si nous affichons la sortie d'horloge d'origine (clk) et la sortie d'horloge retardée (clk d) sur un graphique, elles ressembleront à ceci :



Maintenant, nous allons faire quelque chose d'assez simple. Nous prendrons l'horloge d'origine et l'horloge retardée, ainsi que les ET et les OU pour créer deux nouveaux bits, comme ceci :



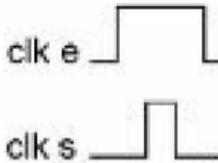
L'un des nouveaux bits est activé lorsque 'clk' ou 'clk d' sont activés, et l'autre nouveau bit n'est activé que lorsque 'clk' et 'clk d' sont activés. Le graphique des entrées et sorties des portes ET et OU est représenté ici. Ils s'allument et s'éteignent tous les deux régulièrement, mais l'un d'eux est allumé plus longtemps qu'il n'est éteint et l'autre est éteint plus longtemps qu'il ne l'est. L'heure de la seconde est en plein milieu de l'heure de la première.



Notez qu'ils ont des noms, "clk e", qui signifie activation de l'horloge, et "clk s", qui signifie réglage de l'horloge.

Et que savez-vous, ces deux bits ont le timing parfait pour déplacer un octet de données d'un registre, à travers le bus et dans un autre registre. Connectez simplement 'clk e' au bit d'activation du registre 'from' et connectez 'clk s' au bit set du registre 'to'.

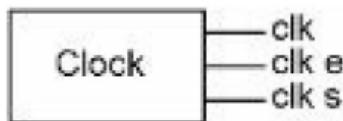
Voici un seul cycle marche/arrêt de ces deux bits.



Si vous regardez la synchronisation ici, cela répond à nos exigences de devoir d'abord activer la sortie d'un registre, puis, après que les données ont un peu de temps pour voyager dans le bus, activer le bit défini du registre de destination et désactivé avant de désactiver le bit d'activation au premier registre.

Bien sûr, ces bits d'horloge ne peuvent pas simplement être connectés directement à chaque registre. Il doit y avoir d'autres portes entre les deux, qui permettent à un seul registre d'obtenir une activation à la fois, et uniquement au(x) registre(s) souhaité(s) de recevoir un ensemble. Mais toutes les activations et tous les ensembles proviennent finalement de ces deux bits car ils ont le bon timing.

Puisque nous utiliserons clk, clk e et clk s dans tout l'ordinateur, voici le schéma que nous utiliserons pour afficher l'horloge :



Faire quelque chose d'utile

Disons que nous voulons faire quelque chose d'utile, comme ajouter un nombre à un autre nombre. Nous avons un nombre dans R0, et il y a un autre nombre dans R1 que nous voulons ajouter au nombre dans R0. Le processeur que nous avons construit jusqu'à présent possède toutes les connexions pour effectuer cet ajout, mais il faudra plus d'un cycle d'horloge pour le faire.

Dans le premier cycle d'horloge, nous pouvons activer R1 sur le bus et le définir sur TMP.

Dans le deuxième cycle, nous pouvons activer R0 sur le bus, régler l'ALU sur ADD et régler la réponse sur ACC.

Dans le troisième cycle, nous pouvons activer ACC sur le bus et le régler sur R0.

Nous avons maintenant l'ancienne valeur de R0, plus R1 dans R0. Cela ne semble peut-être pas très utile, mais c'est l'un des genres de petites étapes que font les ordinateurs. Beaucoup de ces petites étapes donnent l'impression que l'ordinateur est capable de faire des choses très complexes.

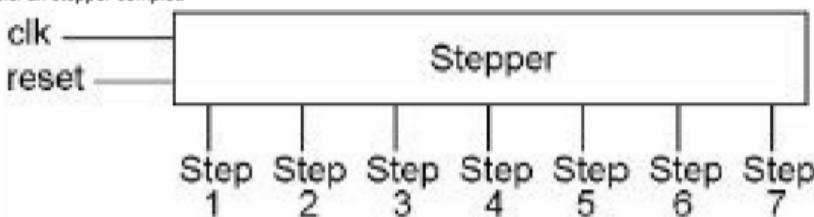
Ainsi on voit que pour que le processeur fasse quelque chose d'utile, il faut plusieurs étapes. Il doit être capable de faire des actions dans une séquence. Nous avons besoin d'une autre pièce à l'intérieur de cette "section de contrôle".

Pas à pas Ce

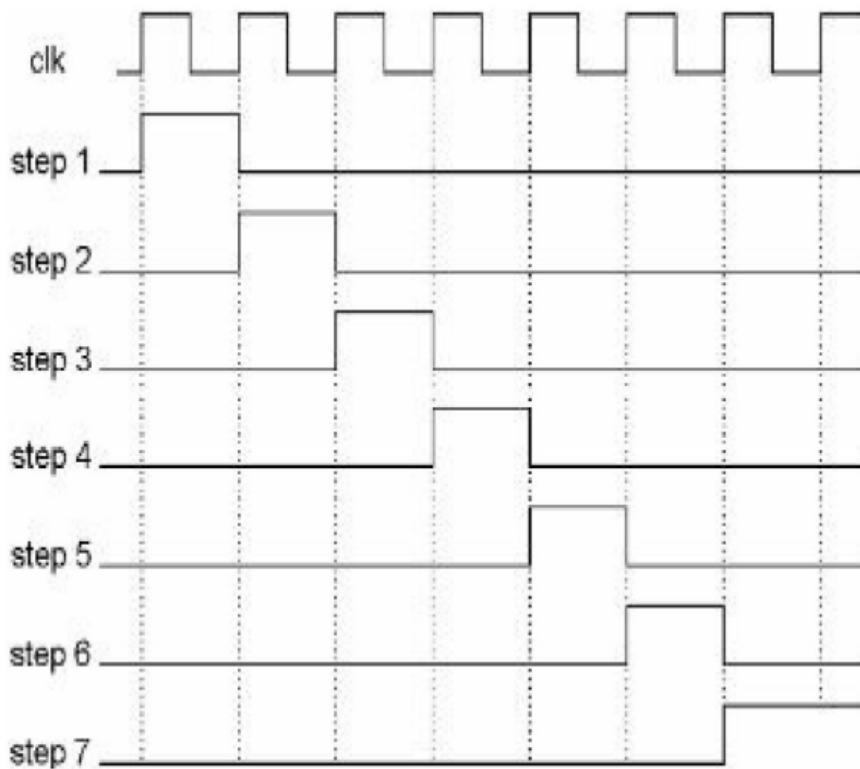
chapitre présente une nouvelle partie appelée « Stepper ».

Tout d'abord, nous décrirons le stepper terminé, en montrant exactement ce qu'il fait. Après cela, nous verrons exactement comment il est construit. S'il vous arrive de faire suffisamment confiance à votre auteur pour croire qu'un tel stepper peut être construit à partir de portes, et que vous êtes tellement pressé que vous voulez sauter la partie "comment il est construit" du chapitre, vous pourriez toujours comprendre l'ordinateur.

Voici un stepper complet.

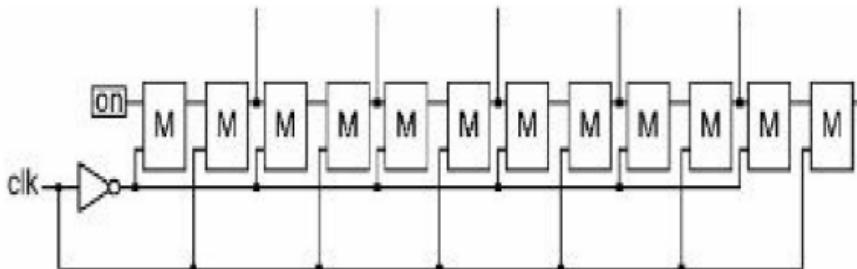


Il a deux entrées. L'un s'appelle "clk", car c'est là que nous connectons un bit qui s'allume et s'éteint, comme notre bit d'horloge d'origine. L'autre entrée est appelée 'reset', qui est utilisée pour ramener le stepper à la première étape. Pour les sorties, il a un certain nombre de bits, dont chacun s'allumera pendant un cycle d'horloge complet, puis s'éteindra, l'un après l'autre. La sortie étiquetée 'Step 1' s'allume pendant un cycle d'horloge, puis 'Step 2' pour le cycle d'horloge suivant, etc. Un stepper peut être construit pour avoir autant d'étapes que nécessaire pour toute tâche particulière que vous voulez faire. Dans le cas de cet ordinateur que nous construisons, sept étapes suffisent. Lorsque la dernière étape (7) s'allume, elle reste allumée et le moteur pas à pas ne fait rien d'autre jusqu'à ce que le bit de réinitialisation soit activé brièvement, moment auquel les étapes recommencent en commençant par «Étape 1». Voici un graphique du bit d'entrée 'clk' et des sorties d'un stepper à sept étapes.



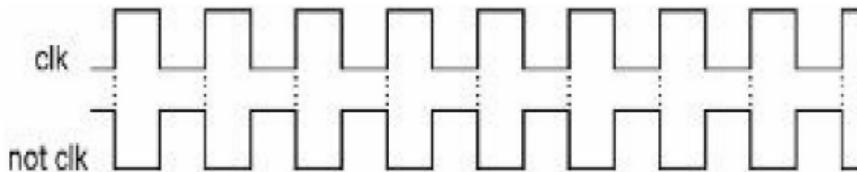
Voici comment le stepper est construit. Cela se fait en utilisant certains des mêmes bits de mémoire que nous avons utilisés pour créer des registres, mais ils sont arrangés très différemment. Nous n'allons rien stocker dans ces bits, nous allons les utiliser pour créer une série d'étapes.

Le stepper se compose de plusieurs bits de mémoire connectés en une chaîne, la sortie de l'un étant connectée à l'entrée du suivant. Voici un schéma qui montre la plupart du stepper :



Regardez d'abord la série de bits de mémoire 'M', tout comme ceux que nous avons utilisés plus tôt dans le livre. Dans cette image, il y en a douze connectés ensemble, avec la sortie de l'un connectée à l'entrée du suivant, tout le long de la ligne. L'entrée du premier bit à gauche est connectée à un endroit où l'électricité est toujours allumée, donc lorsque le bit défini de ce «M» s'allume, ce «M» recevra cet état et le transmettra à son production.

Si vous regardez les bits définis de ces 'M', vous verrez que les bits définis des 'M' pairs sont connectés à 'clk', et les bits définis des 'M' impairs sont connectés à la même horloge après son passage. une porte NON. Ce nouveau bit créé en faisant passer 'clk' par une porte NOT peut être appelé "not clk", et nous pouvons montrer les deux sur ce graphique :



Alors que va-t-il se passer avec ce tas de portails ? Si vous supposez que tous les 'M's démarrent à l'état off, puis que 'clk' commence à "tic-tac", voici ce qu'il va faire.

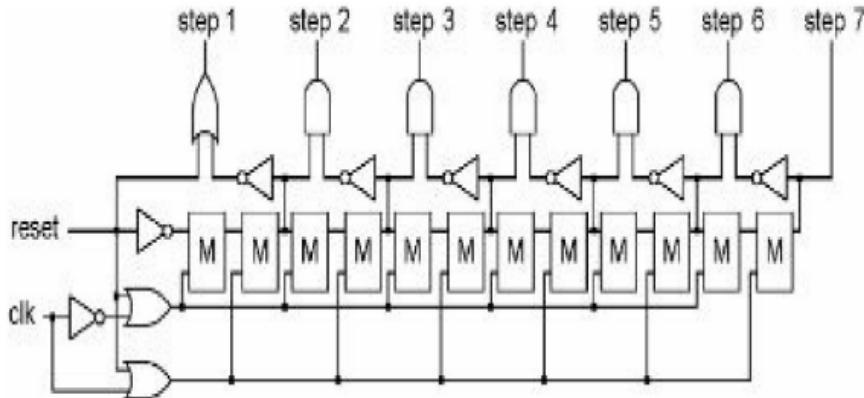
La première fois que 'clk' s'allume, rien ne se passe, car le bit défini du premier 'M' est connecté à 'not clk', qui est désactivé lorsque 'clk' est activé. Lorsque 'clk' s'éteint, 'not clk' s'allume et le premier 'M' s'allume, mais rien ne se passe au second 'M'

car son bit 'set' est connecté à 'clk', qui est maintenant désactivé. Lorsque 'clk' revient, le deuxième 'M' va maintenant s'allumer. Au fur et à mesure que l'horloge tourne, le "on" qui entre dans le premier bit de mémoire descendra la ligne, un bit à chaque fois que l'horloge s'allume et un bit à chaque fois que l'horloge s'éteint. Ainsi, deux bits s'allument pour chaque cycle d'horloge.

Maintenant, en passant au diagramme pas à pas complet ci-dessous, l'étape 1 provient d'une porte NOT connectée à la sortie du deuxième "M". Étant donné que tous les "M" démarrent, l'étape 1 sera activée jusqu'à ce que le second 'M' s'allume, moment auquel l'étape 1 sera terminée. Pour les étapes restantes, chacune durera du moment où son côté gauche 'M' s'allume jusqu'au moment où son côté droit 'M' s'allume. Les portes ET des étapes 2 à 6 ont les deux entrées activées lorsque le "M" gauche est activé et le "M" droit est désactivé. Si nous connectons la sortie d'un 'M' et le NON de la sortie d'un 'M' deux espaces plus loin à une porte ET, sa sortie sera allumée pendant un cycle d'horloge complet. Chacun s'allume lorsque son entrée gauche s'est allumée, mais que son entrée droite n'est pas encore allumée. Cela nous donne une série de bits qui s'allument chacun pendant un cycle d'horloge, puis s'éteignent.

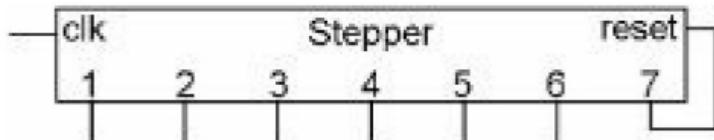
La seule chose qui manque ici est que les bits 'M' s'allument et restent allumés. Une fois qu'ils sont tous allumés, il n'y a plus d'action malgré le tic-tac continu de l'horloge. Nous avons donc besoin d'un moyen de les réinitialiser tous afin que nous puissions recommencer.

Nous devons avoir un moyen de désactiver l'entrée du premier "M", puis d'activer tous les bits définis en même temps. Lorsque cela se produit, le "off" à l'entrée du premier "M" parcourt tous les "M" aussi vite que possible. Nous allons ajouter une nouvelle entrée appelée "reset", qui accomplira ces choses.



Lorsque nous activons 'reset', cela fait de l'entrée du premier bit 'M' un zéro et active tous les 'sets' en même temps afin que le zéro puisse parcourir très rapidement la ligne des 'M'. La réinitialisation est également associée à l'étape 1 de sorte que l'étape 1 s'active immédiatement. Maintenant, tous les bits sont désactivés et nous avons commencé une autre séquence. La réinitialisation ne doit être activée que pendant une fraction de cycle d'horloge.

Pour récapituler, il s'agit d'un stepper. Il a deux entrées : une horloge et une réinitialisation. Pour les sorties, il a un certain nombre de bits, dont chacun s'allumera pendant un cycle d'horloge. Nous pouvons en fait faire cela aussi longtemps que nécessaire, mais pour les besoins de ce livre, un pas à pas en sept étapes sera suffisant. Il n'y aura qu'un seul stepper dans notre ordinateur, nous allons le représenter avec ce schéma simplifié.

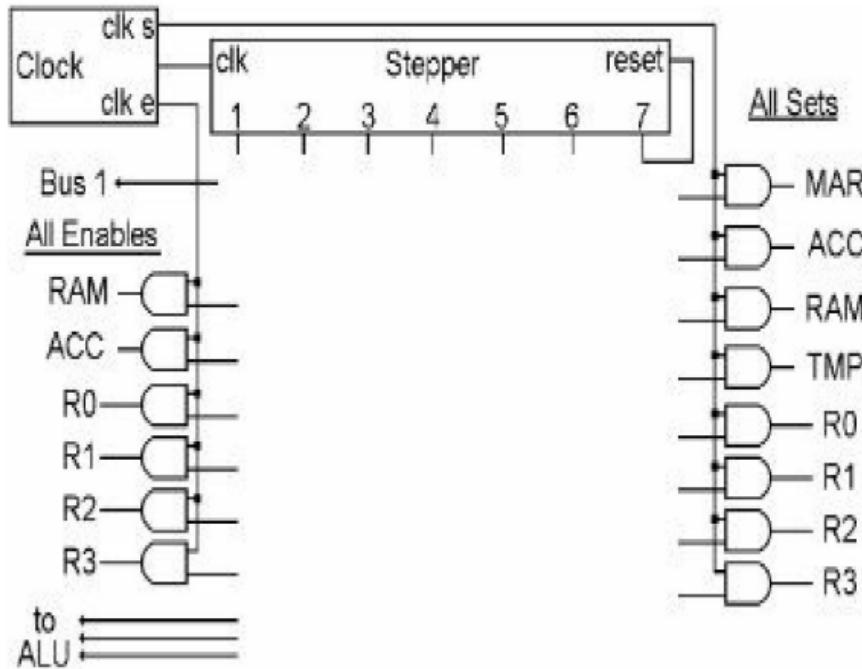


Nous avons déplacé le bit de réinitialisation sur le côté droit du diagramme et l'avons connecté à la dernière étape (7.) afin que le stepper se réinitialise automatiquement. L'étape 7 ne sera cependant pas activée très longtemps, car elle se ferme

s'éteint dès que le zéro peut passer à travers la chaîne de 'M'. Cela signifie que l'étape 7 ne durera pas assez longtemps pour être utilisée pour l'un de nos transferts de données sur le bus. Toutes les choses que nous voulons accomplir auront lieu dans les étapes 1 à 6.

Tout est sous contrôle Avec notre horloge, nous avons un battement de tambour pour faire avancer les choses. Il a une sortie de base et deux autres conçues pour faciliter le mouvement du contenu des registres de l'un à l'autre. Avec le stepper, nous avons une série de bits qui s'allument les uns après les autres, chacun pour un cycle d'horloge.

Vous souvenez-vous du schéma du CPU que nous avons vu il y a quelques chapitres ? Il a montré le bus, l'ALU, six registres et même l'autre moitié de l'ordinateur (la RAM) tous connectés assez proprement. Au moins toutes les connexions de bus étaient là. Mais tous les registres, la RAM, le Bus 1 et l'ALU sont contrôlés par des fils qui viennent de cette mystérieuse boîte étiquetée 'Control Section' dont nous ne savons rien encore. Il est maintenant temps de regarder à l'intérieur de cette boîte.



Ce dessin est le début de la section de contrôle de l'ordinateur. En haut se trouvent l'horloge et le stepper.

Ensuite, tous les bits de contrôle des registres et de la RAM ont été réunis ici en un seul endroit, avec tous les bits "activer" à gauche et tous les bits "définir" à droite. Ensuite, nous avons connecté la sortie d'une porte ET à chaque bit 'enable' et chaque bit 'set'.

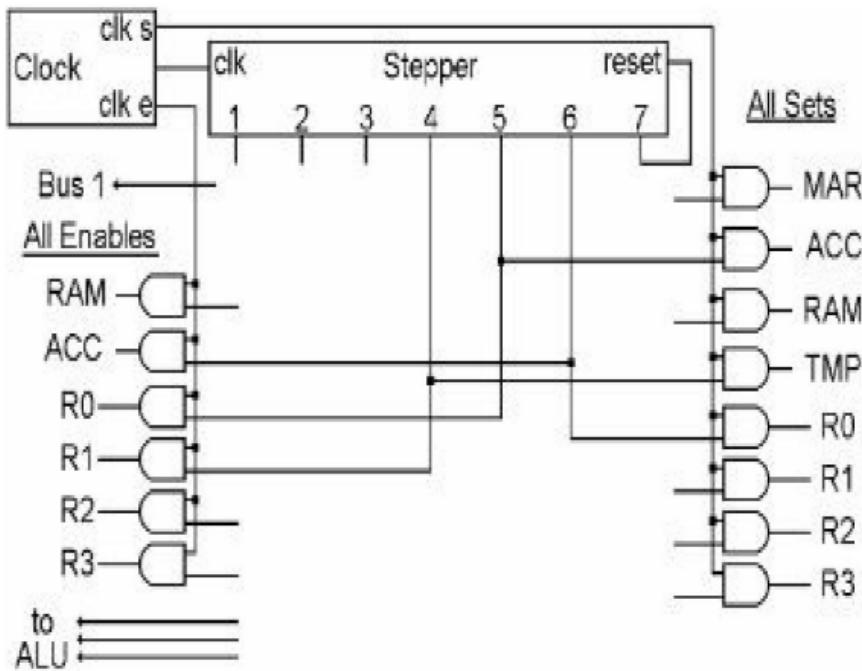
Une entrée de chaque porte ET est connectée soit à 'clk e' pour les 'enables' à gauche, soit à 'clk s' pour les 'sets' à droite. Ainsi, si nous utilisons l'autre entrée de ces portes ET pour sélectionner l'un de ces registres, le bit "activer" de tous les éléments de gauche ne sera jamais activé sauf pendant le temps 'clk e'. De même, à droite, le bit 'set' de l'un de ces registres ne sera activé que pendant le temps 'clk s'.

C'est une sorte de standard. Tout ce dont nous avons besoin pour

faire faire quelque chose à l'ordinateur est ici en un seul endroit. Tout ce que nous avons à faire est de connecter des bits de contrôle à certaines étapes de manière intelligente, et quelque chose d'utile se produira.

Faire quelque chose d'utile, revisité

Maintenant que nous avons le début de notre section de contrôle, nous pouvons simplement ajouter quelques fils, et nous pourrons faire l'addition simple que nous avons postulée plus tôt, celle d'ajouter R1 à R0.



Tout ce que nous avons à faire pour «faire quelque chose d'utile», comme ajouter R1 à R0, est de connecter quelques fils au milieu, comme indiqué dans ce schéma avec les étapes quatre, cinq et six.

Chaque étape provoque quelque chose qui se passe dans certaines des parties qui sont montrées dans le diagramme de la CPU. Chaque étape est connectée à un «enable» à gauche et à un «set» à droite, et oblige donc une partie à connecter sa sortie au bus, et une autre partie à enregistrer ce qui apparaît maintenant à son entrée. La quatrième étape est câblée à R1 "activer"

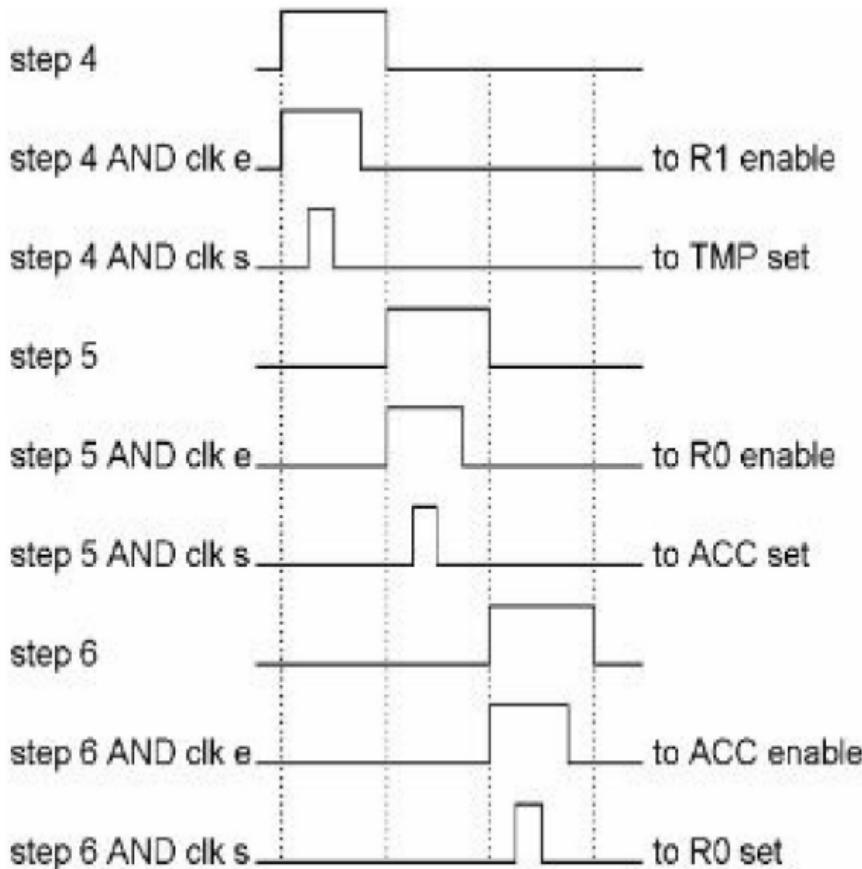
et TMP 'set.' L'étape cinq est câblée à R0 'enable' et ACC 'set'. Les bits 'op' ALU n'ont pas besoin de connexions puisque le code 'op' pour ADD est 000. La sixième étape est câblée à ACC 'enable' et R0 'set'.

Au cours de la quatrième étape, R1 est activé et TMP est défini. Le contenu de R1 voyage à travers le bus (dans le diagramme CPU) et est capturé par TMP.

Au cours de l'étape cinq, R0 est activé et ACC est défini. Si nous voulions faire autre chose que ADD, c'est l'étape où nous activerions les bits de code 'op' ALU appropriés.

Au cours de l'étape six, ACC est activé et R0 est défini.

Voici un graphique des étapes, indiquant quand chaque registre est activé et défini.



R0 contient maintenant la somme du contenu original de R0 plus R1.

C'est ainsi que l'ordinateur fait bouger les choses dans un ballet étroitement contrôlé de bits et d'octets se déplaçant à l'intérieur de la machine.

À l'étape sept, le stepper est réinitialisé à l'étape 1, où le processus se répète. Bien sûr, il n'est pas très utile de simplement

faites cet ajout encore et encore, même si vous commencez avec le numéro 1 dans R0 et R1, R0 atteindra 255 assez rapidement.

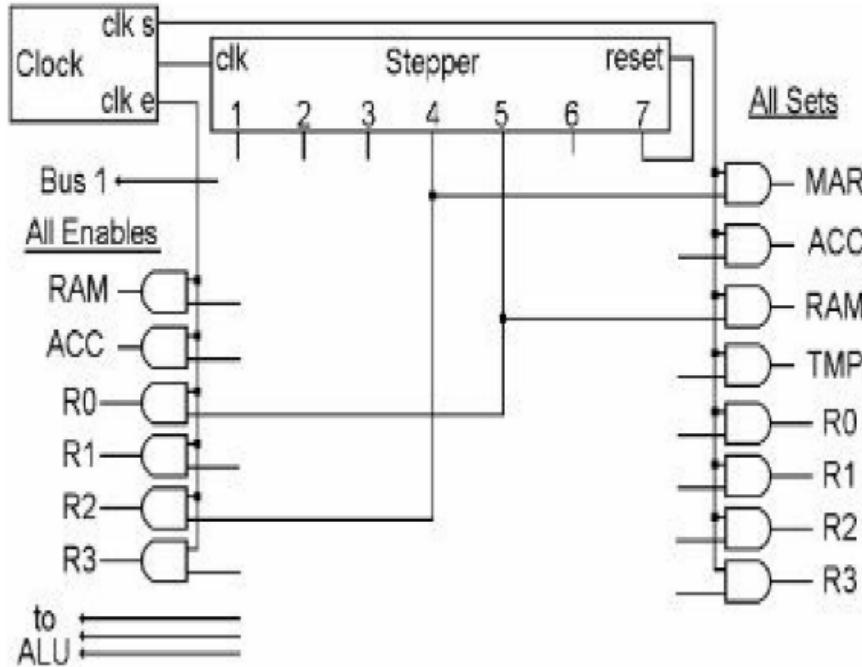
Si l'horloge de notre ordinateur tourne un milliard de fois par seconde, autrement appelé un gigahertz, et même si nous utilisons plusieurs cycles d'horloge pour "faire quelque chose d'utile" comme ça, cela signifie que l'ordinateur peut faire quelque chose comme ça des centaines de millions de fois en une seconde.

Mais nous ne voulons pas simplement ajouter R1 à R0 encore et encore.

Peut-être que maintenant que nous avons ajouté R1 à R0, nous voulons stocker ce nouveau numéro à une adresse particulière dans la RAM, et R2 contient cette adresse. Encore une fois, notre processeur dispose de toutes les connexions nécessaires pour le faire, et encore une fois, il faudra plus d'un cycle d'horloge pour le faire. À l'étape 4, nous pouvons déplacer R2 à travers le bus vers MAR. À l'étape 5, nous pouvons déplacer R0 à travers le bus vers la RAM.

C'est tout ce qui est nécessaire, juste deux cycles d'horloge et nous avons terminé.

Le câblage pour cette opération est plus simple que l'autre, seulement deux validations et deux ensembles.



Il existe de nombreuses combinaisons de choses que nous pouvons faire avec la RAM, les six registres et l'ALU. Nous pourrions obtenir un octet de la RAM et le déplacer vers l'un des quatre registres, nous pourrions déplacer un ou deux des registres à travers l'ALU et les AJOUTER, les ET, les OU, les XOR, etc.

Nous avons besoin d'un moyen pour notre CPU de faire une chose une fois, et une chose différente la fois suivante. La section de contrôle a besoin de quelque chose pour lui dire quoi faire dans chaque séquence.

Et après?

Maintenant, voici une idée effrayante. Imaginez que le travail d'un employé dans un restaurant de restauration rapide soit décomposé en ses éléments individuels. Marchez jusqu'au comptoir, dites « Puis-je prendre votre commande ? » écoutez la réponse, appuyez sur le bouton "cheeseburger" sur la caisse enregistreuse, etc. Disons maintenant qu'il y a 256 actions individuelles ou moins impliquées dans le travail de travail dans un tel établissement.

Vous pourriez alors inventer un code qui associerait l'un des états d'un octet à chacune des activités individuelles d'un employé. Ensuite, vous pouvez exprimer la séquence des actions d'un employé sous la forme d'une séquence d'octets.

Nous créons d'abord une table de codes. Nous écrivons quelques codes sur le côté gauche de la page. Ensuite, nous décidons de ce que nous voulons que ces codes signifient et écrivons ces significations à côté des codes. Nous avons maintenant une liste de toutes les actions possibles qu'un employé peut entreprendre, et un code qui représente chacune d'entre elles :

0000 0000 = Marchez jusqu'au comptoir 0000 0001 = Dites

"Puis-je prendre votre commande ?"

0000 0010 = Écoutez la réponse 0000 0011 = Appuyez sur

le bouton cheeseburger 0000 0100 = Appuyez sur le bouton frites.

0000 0101 = Appuyer sur le bouton lait 0000 0110 = Appuyer
sur le bouton total 0000 0111 = Récupérer l'argent 0000 1000 =

Rendre la monnaie au client 0000 1001 = Ouvrir un sac vide

0000 1010 = Placer un cheeseburger dans le sac 0000 1011 = Placer des
frites dans le sac 0000 1100 = Placer un récipient à lait dans le sac 0000

1101 = Remettre le sac au client 1000 0000 = Aller au numéro d'étape dans les 6
bits de droite.

0100 0000 = Si "oui", passez au numéro d'étape dans les 6 bits de droite.

0001 0000 = Rentrez chez vous.

Maintenant, si nous voulons décrire comment l'employé est censé agir, nous écrivons une séquence d'événements qu'il doit suivre:

1. 0000 0000 = Marchez jusqu'au comptoir. 2.
 0000 0001 = Dites « Puis-je prendre votre commande ? » 3.
 0100 0010 = Si le client ne répond pas, passez à l'étape 2. 4.

 0000 0010 = Écoutez la réponse. 5.
 0100 0111 = Si le client ne dit pas cheeseburger, passez à l'étape 7. 6.

 0000 0011 = Appuyez sur le bouton cheeseburger. sept.
 0100 1001 = Si le client ne dit pas frites, passez à l'étape 9. 8.

 0000 0100 = Appuyez sur le bouton frites. 9.
 0100 1011 = Si le client ne dit pas lait, passez à l'étape 11.

10. 0000 0101 = Appuyez sur le bouton lait.
11. 0100 1101 = Si le client dit que c'est tout, passez à l'étape 13.

12. 1000 0100 = Revenir à l'étape 4.
13. 0000 0110 = Appuyez sur le bouton total.
14. 0000 0111 = Récupérez l'argent.
15. 0000 1000 = Rendre la monnaie et la donner au client.
16. 0000 1001 = Ouvrir un sac vide.
17. 0101 0011 = Si la commande n'inclut pas le cheeseburger, passez à l'étape 19.

18. 0000 1010 = Placez un cheeseburger dans le sac.
19. 0101 0110 = Si la commande n'inclut pas les frites, passez à l'étape 22.

21. 0000 1011 = Placer les frites dans le sac.
22. 0101 1000 = Si la commande ne comprend pas de lait, passez à l'étape 24.

23. 0000 1100 = Placez un récipient à lait dans le sac.
24. 0000 1101 = Remettez le sac au client.
25. 0101 1011 = S'il est temps d'arrêter, passez à l'étape 27.
26. 1000 0001 = Revenir à l'étape 1.
27. 0001 0000 = Rentrez chez vous.

J'espère que personne n'essaiera jamais de faire apprendre aux employés d'un fast-food un code comme celui-ci. Les gens n'apprécient pas d'être aussi mécanisés. Mais peut-être que quelqu'un essaiera un jour de doter l'un de ces restaurants de robots. Dans ce cas, les robots fonctionneraient probablement mieux en utilisant ce type de code.

Et notre ordinateur pourrait être capable de "comprendre" un code comme celui-ci.

La première grande invention

Ce dont nous avons besoin, c'est d'un moyen d'effectuer différentes opérations d'une séquence pas à pas à la suivante. Comment pourrions-nous l'avoir câblé d'une manière pour une séquence, puis d'une manière différente pour la séquence suivante ? La réponse, bien sûr, est d'utiliser plus de portes. Le câblage d'une opération peut être connecté ou déconnecté avec des portes ET, et le câblage d'une opération différente peut être connecté ou déconnecté avec d'autres portes ET. Et il pourrait y avoir une troisième et une quatrième possibilité ou plus. Tant qu'une seule de ces opérations est connectée à la fois, cela fonctionnera correctement. Nous avons maintenant plusieurs opérations différentes qui peuvent être effectuées, mais comment sélectionner celle qui sera effectuée ?

Le titre de ce chapitre est « La première grande invention », alors quelle est l'invention ? L'invention est que nous aurons une série d'instructions dans la RAM qui indiqueront au CPU quoi faire. Nous avons besoin de trois choses pour que cela fonctionne.

La première partie de l'invention est que nous allons ajouter un autre registre au CPU. Ce registre sera appelé le « registre d'instructions » ou « IR » en abrégé.

Les bits de ce registre "diront" au CPU ce qu'il doit faire. L'IR reçoit son entrée du bus et sa sortie va dans la section de contrôle du CPU où les bits sélectionnent l'une des nombreuses opérations possibles.

La deuxième partie de l'invention est un autre registre dans le CPU appelé "Instruction Address Register", ou "IAR" en abrégé. Ce registre a son entrée et sa sortie connectées au bus tout comme les registres à usage général, mais celui-ci n'a qu'un seul but, et c'est de stocker l'adresse RAM de la prochaine instruction que nous voulons déplacer dans l'IR. Si l'IAR contient 0000 1010 (10 décimal), la prochaine instruction qui sera déplacée vers l'IR est l'octet résidant à l'adresse RAM dix.

La troisième partie de l'invention est un câblage dans la section de contrôle qui utilise le stepper pour déplacer "l'instruction" souhaitée de la RAM vers l'IR, ajouter 1 à l'adresse dans l'IAR et faire l'action demandée par le

instruction qui a été mise dans l'IR. Lorsque cette instruction est terminée, le stepper recommence, mais maintenant l'IAR a été ajouté à 1, donc quand il obtient cette instruction de la RAM, ce sera une instruction différente qui était située à l'adresse RAM suivante.

Le résultat de ces trois parties est une grande invention.

C'est ce qui nous permet de faire faire à l'ordinateur beaucoup de choses différentes. Nos bus, ALU, RAM et registres permettent de nombreuses combinaisons. Le contenu de l'IR déterminera quels registres sont envoyés où, et quel type d'arithmétique ou de logique sera fait sur eux. Tout ce que nous avons à faire est de placer une série d'octets dans la RAM qui représentent une série de choses que nous voulons faire, l'une après l'autre.

Cette série d'octets résidant dans la RAM que le CPU va utiliser s'appelle un "programme".

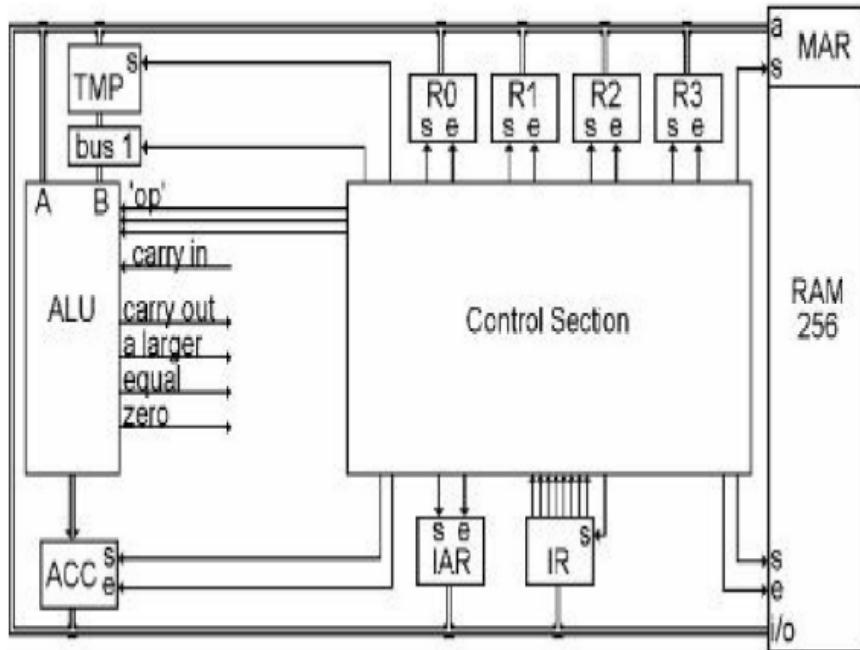
La chose fondamentale qui se passe ici est que le CPU "récupère" une instruction de la RAM, puis "exécute" l'instruction. Ensuite, il récupère le suivant et

l'exécute. Cela se produit encore et encore et encore, des millions ou des milliards de fois chaque seconde. C'est la simplicité de ce que fait un ordinateur. Quelqu'un met un programme dans la RAM, et ce programme, s'il est conçu intelligemment, oblige l'ordinateur à faire quelque chose que les gens trouvent utile.

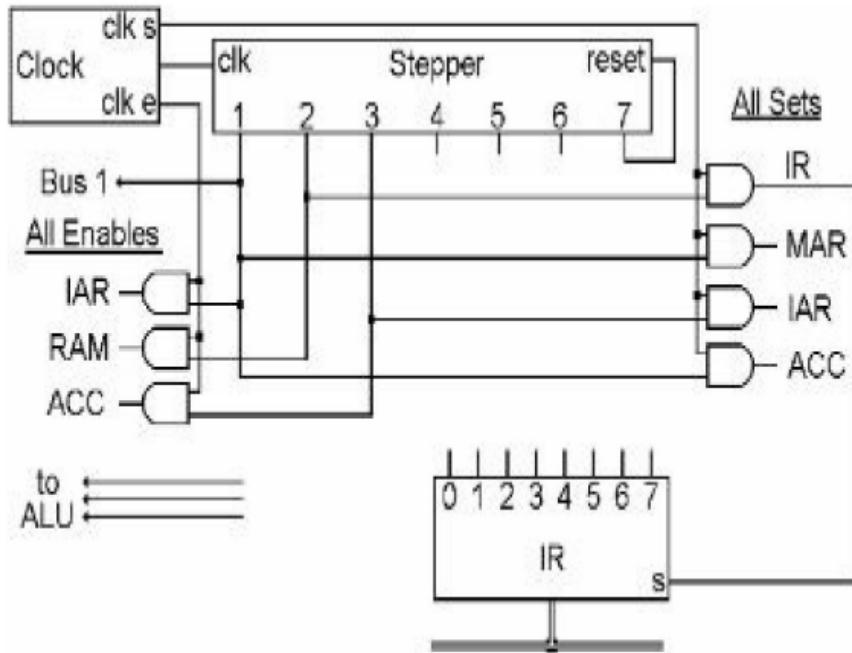
Le stepper de cet ordinateur comporte sept étapes. Le but de l'étape 7 est uniquement de réinitialiser le stepper à l'étape 1. Il y a donc six étapes au cours desquelles le CPU fait de petites choses. Chaque étape dure un cycle d'horloge. Les six étapes prises dans leur ensemble s'appellent un "cycle d'instruction". Il faut six étapes au processeur pour effectuer toutes les actions nécessaires pour récupérer et exécuter une instruction.

Si nous supposons que notre horloge tourne à un gigahertz, alors notre ordinateur pourra exécuter 166 666 666 instructions par seconde.

Voici l'image du CPU avec les deux nouveaux registres ajoutés. Là, ils sont sous la section de contrôle, connectés au bus. L'IAR a un "set" et un "enable", l'IR n'a qu'un "set", tout comme TMP et MAR parce que leurs sorties ne sont pas connectées au bus, nous n'avons donc jamais besoin de les éteindre.



Vous trouverez ci-dessous le câblage de la section de contrôle qui effectue la partie "extraction" du cycle d'instructions. Il utilise les trois premières étapes du stepper et est le même pour tous les types d'instructions.



Les trois premières étapes du stepper sont illustrées ici et aboutissent à "récupérer" la prochaine "instruction" de la RAM.

Ensuite, le reste des étapes « exécute » l'« instruction ». Ce qui sera fait exactement aux étapes 4, 5 et 6 est déterminé par le contenu de l'instruction qui a été extraite. Ensuite, le stepper recommence, récupère l'instruction suivante et l'exécute.

Le bas de ce diagramme comprend le registre d'instructions. Notez que nous avons donné des numéros aux bits individuels de l'IR, de 0 à gauche à 7 à droite. Nous ferons référence aux bits individuels

bientôt.

Voici les détails de la manière exacte dont les étapes 1, 2 et 3 aboutissent à la récupération d'une instruction dans notre petit ordinateur :

L'étape 1 est la plus compliquée car nous accomplissons en fait deux choses en même temps. La principale chose que nous voulons faire est de faire passer l'adresse dans IAR à MAR.

C'est l'adresse de la prochaine instruction que nous voulons récupérer de la RAM. Si vous regardez le fil sortant de l'étape 1 du stepper, vous pouvez voir que deux des endroits auxquels il est connecté sont le « enable » d'IAR et le « set » de MAR. Ainsi, le contenu d'IAR sera placé sur le bus pendant 'clk e' et placé dans MAR pendant 'clk s.' À un moment donné pendant le cycle d'instructions, nous devons ajouter 1 à la valeur dans IAR, et puisque IAR est déjà sur le bus, autant le faire maintenant. Si nous n'envoyons rien aux bits 'op' de l'ALU, ils seront tous nuls, et puisque 000 est le code pour ADD, l'ALU effectuera une opération ADD sur tout ce qui se trouve sur ses deux entrées, et présentera la réponse à l'ACC. Une entrée provient du bus, qui a IAR dessus pendant ce temps. Si nous activons également le bit 'bus 1' pendant l'étape 1, l'autre entrée de l'ALU sera un octet avec la valeur binaire de 1. Si nous activons le 'set' de ACC pendant 'clk s,' nous allons capturer la somme de IAR plus 1 dans ACC. Il s'agit simplement de l'adresse de l'instruction que nous voudrons récupérer après avoir terminé avec l'instruction actuelle !

L'étape 2 active l'octet actuellement sélectionné dans la RAM sur le bus et le définit dans IR. C'est l'instruction que nous allons "exécuter" dans les étapes 4, 5 et 6 de ce cycle d'instructions. Dans le diagramme, vous pouvez voir que le fil venant de l'étape 2 est connecté à la "activation" de la RAM et au "set" de l'IR.

À l'étape 3, nous devons terminer la mise à jour de l'IAR. Nous y avons ajouté 1 à l'étape 1, mais la réponse est toujours dans ACC. Il doit être déplacé vers l'IAR avant le début du cycle d'instruction suivant. Ainsi, vous pouvez voir que le fil sortant de l'étape 3 est connecté à « activer » de l'ACC et « régler » de l'IAR.

Au moment où nous arrivons à l'étape 4, l'instruction a déjà été déplacée de la RAM vers l'IR, et maintenant les étapes 4, 5 et 6 peuvent alors faire tout ce qui est demandé par le contenu de l'IR. Lorsque cette opération est terminée et que le moteur pas à pas est réinitialisé, la séquence recommencera, mais maintenant IAR s'est vu ajouter 1, de sorte que l'instruction à l'adresse RAM suivante sera récupérée et exécutée.

Cette idée de mettre une série d'instructions en RAM et

faire en sorte que le CPU les exécute est une excellente invention.

Des instructions

Nous avons maintenant ce nouveau registre, appelé le registre d'instructions, qui contient un octet qui va dire à la section de contrôle quoi faire. Les motifs qui sont mis dans ce registre ont un sens. Cela ressemble à un autre code, et en effet, il l'est. Ce code sera appelé le « code d'instruction ».

Puisque nous construisons cet ordinateur à partir de zéro, nous devons inventer notre propre code d'instruction. Nous prendrons les 256 codes différents qui peuvent être mis dans le registre d'instructions et déciderons de ce qu'ils signifieront. Ensuite, nous devons concevoir le câblage à l'intérieur de l'unité de contrôle qui fera en sorte que ces instructions fassent ce que nous avons dit qu'elles feraien

Vous souvenez-vous du code binaire ? Nous avons dit que c'était la chose la plus proche d'un code informatique "naturel" parce qu'il était basé sur la même méthode que nous utilisons pour notre système numérique normal. Ensuite, il y avait le code ASCII, qui vient d'être inventé par un groupe de personnes lors d'une réunion. Il n'y a rien de naturel dans l'ASCII, c'est juste ce que ces gens ont décidé que ce serait.

Nous avons maintenant le code d'instruction, qui sera également un code totalement inventé - rien de naturel à ce sujet. De nombreux codes d'instructions différents ont été inventés pour de nombreux types d'ordinateurs différents. Nous n'en étudierons aucune ici, et vous n'aurez pas non plus besoin d'en étudier aucune plus tard, à moins que vous n'entriez dans une carrière hautement technique où cela est nécessaire. Mais tous les codes d'instruction sont similaires, en ce sens qu'ils font fonctionner l'ordinateur. Le seul code d'instruction dans ce livre sera celui que nous inventons pour notre simple ordinateur. La chose la plus importante dans l'invention de notre code d'instructions sera la simplicité du câblage qui fera fonctionner le code.

Combien d'instructions différentes pourrait-il y avoir ? Comme le registre d'instructions est un octet, il peut y avoir jusqu'à 256 instructions différentes. Heureusement, nous n'aurons que neuf types d'instructions, et les 256 combinaisons tomberont dans l'une de ces catégories.

Ils sont assez faciles à décrire.

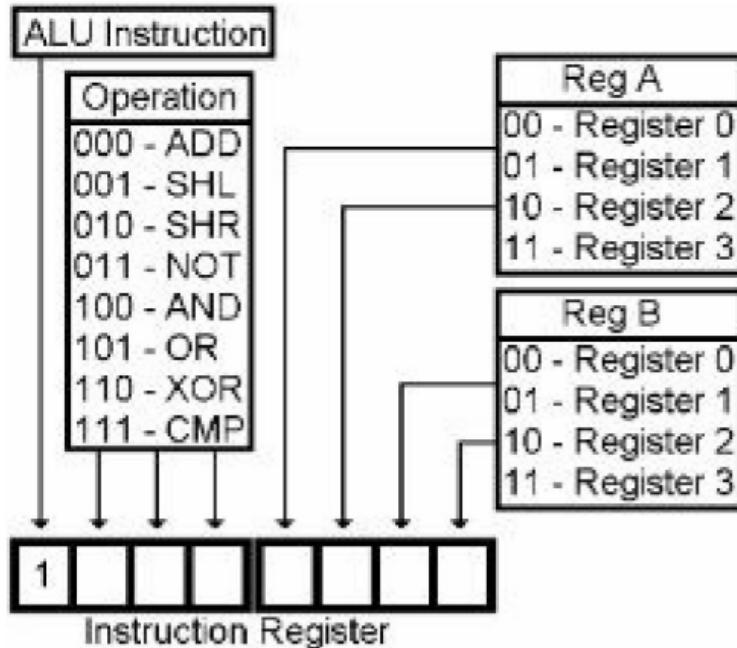
Toutes les instructions impliquent le déplacement d'octets sur le bus.

Les instructions entraîneront le transfert d'octets vers ou depuis la RAM, vers ou depuis des registres, et parfois via l'ALU. Dans les chapitres suivants, pour chaque type d'instruction, nous examinerons les bits de cette instruction, les portes et le câblage nécessaires pour la faire fonctionner, et un autre code pratique que nous pouvons utiliser pour faciliter l'écriture de programmes.

L'instruction arithmétique ou logique Ce premier type d'instruction est le type qui utilise l'ALU comme notre opération ADD plus tôt. Comme vous vous en souvenez, l'ALU a huit choses qu'elle peut faire, et pour certaines de ces choses, elle utilise deux octets d'entrée, pour d'autres choses, elle n'utilise qu'un octet d'entrée. Et dans sept de ces cas, il a un octet de sortie.

Ce type d'instruction choisira une des opérations ALU, et deux registres. C'est l'instruction la plus polyvalente que l'ordinateur peut faire. Il a en fait 128 variations, car il y a huit opérations et quatre registres, et vous pouvez choisir deux fois parmi les quatre registres. Soit huit fois quatre fois quatre, soit 128 façons possibles d'utiliser cette instruction. Ainsi, il ne s'agit pas d'une seule instruction, mais plutôt de toute une classe d'instructions qui utilisent toutes le même câblage pour faire le travail.

Voici le code d'instruction pour l'instruction ALU. Si le premier bit du registre d'instructions est un 1, il s'agit alors d'une instruction ALU. C'est la simplicité de celui-ci.
Si le premier bit est activé, les trois bits suivants de l'instruction sont envoyés à l'ALU pour lui dire quoi faire, les deux bits suivants choisissent l'un des registres qui seront utilisés et les deux derniers bits choisissent l'autre registre. qui sera utilisé.



Par conséquent, l'Instruction ALU (1), ajouter (000)

Registre 2 (10) et Registre 3 (11), et placer la réponse dans le Registre 3, serait :

1000 1011. Si vous avez placé ce code (1000 1011) dans la RAM à l'adresse 10, et que vous réglez l'IAR sur 10, et que vous démarrez l'ordinateur, il récupèrera le 1000 1011 de l'adresse 10, le placerait en IR, puis le câblage dans la section de contrôle ferait l'addition de R2 et R3.

Si vous choisissez une opération à une entrée, telle que SHL, SHR ou NOT, l'octet viendra du Reg A, passera par l'ALU et la réponse sera placée dans le Reg B. Vous pouvez choisir d'aller d'un registre à un autre tel que R1 à R3, ou choisir de revenir d'un registre au même, tel que R2 à R2. Lorsque vous faites cela, le contenu original du registre sera remplacé.

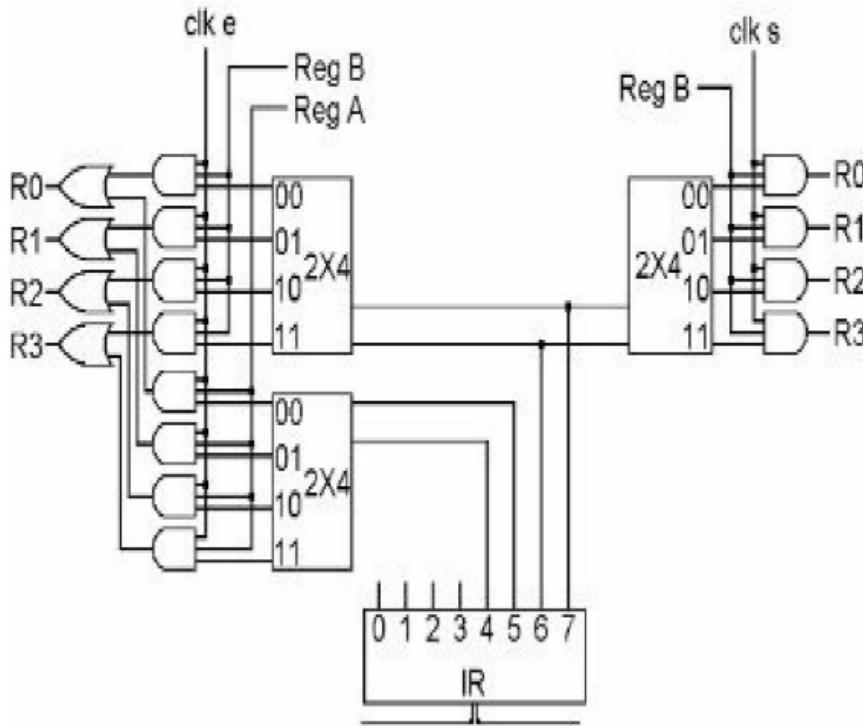
Pour deux opérations d'entrée, Reg A et Reg B seront envoyés

à l'ALU, et la réponse sera envoyée à Reg B. Ainsi, tout ce qui se trouvait dans Reg B, qui était l'une des entrées de l'opération, sera remplacé par la réponse. Vous pouvez également spécifier le même registre pour les deux entrées. Cela peut être utile, par exemple, si vous voulez mettre tous les zéros dans le registre 1, juste XOR R1 avec R1. Peu importe ce qui se trouve dans R1 pour commencer, toutes les comparaisons de bits seront les mêmes, ce qui rend la sortie de tous les bits nulle, qui est remplacée dans R1.

L'opération CMP prend deux entrées et les compare pour voir si elles sont égales, et sinon, si la première est plus grande. Mais l'opération CMP ne stocke pas son octet de sortie. Il ne remplace pas le contenu de l'un ou l'autre des octets d'entrée.

Le câblage dans l'unité de contrôle pour l'instruction ALU est assez simple, mais il y a une chose supplémentaire qui sera utilisée par de nombreux types d'instructions que nous devons d'abord examiner. Cela a à voir avec les registres. Dans « Doing Something Useful Revisited », nous avons utilisé deux registres. Pour les utiliser, nous avons juste connecté la porte ET pour chaque registre au pas désiré du stepper.

C'était bien, mais dans l'instruction ALU, et bien d'autres, il y a des bits dans le registre d'instruction qui spécifient quel registre utiliser. Par conséquent, nous ne voulons pas nous connecter directement à un registre, nous devons pouvoir nous connecter à n'importe lequel des registres, mais laissez les bits de l'instruction choisir exactement lequel. Voici le câblage de la section de contrôle qui le fait :



Regardez d'abord le côté droit. Lorsque nous voulons définir un registre à usage général, nous connectons l'étape appropriée à ce fil que nous appellerons 'Reg B.' Comme vous pouvez le voir, 'clk s' est connecté aux quatre portes ET. 'Reg B' est également connecté aux quatre portes ET. Mais ces quatre portes ET ont chacune trois entrées. La troisième entrée de chaque porte ET provient d'un décodeur 2x4. Vous vous souvenez qu'une et une seule sortie d'un décodeur est activée à un moment donné, donc un seul registre sera réellement sélectionné pour avoir son bit "set" activé. L'entrée du décodeur provient des deux derniers bits de l'IR, ils déterminent donc quel registre sera défini par ce fil étiqueté "Reg B". Si vous revoyez le tableau des

bits du code d'instruction ALU, cela montre que les deux derniers bits de l'instruction déterminent le registre que vous souhaitez utiliser pour Reg B.

Le côté gauche de l'image ressemble beaucoup au côté droit, sauf qu'il y en a deux de tout. Rappelez-vous que dans une instruction ALU telle que ADD, nous devons activer deux registres, un à la fois, pour les entrées de l'ALU. Les deux derniers bits de l'instruction sont également utilisés pour 'Reg B' sur la gauche, et vous pouvez voir que 'clk e', 'Reg B' et un décodeur sont utilisés pour activer un registre au cours de son étape appropriée. Les bits 4 et 5 de l'IR sont utilisés pour activer 'Reg A' pendant son étape appropriée, en utilisant un décodeur séparé et un fil appelé 'Reg A.' Les sorties de ces deux structures sont reliées par un OU avant d'aller aux bits d'activation du registre réel. Nous ne sélectionnerons jamais 'Reg A' et 'Reg B' en même temps.

Que se passe-t-il lorsque l'instruction qui a été récupérée commence par un 1 ? Cela signifie qu'il s'agit d'une instruction ALU et que nous devons faire trois choses. Nous voulons d'abord déplacer "Reg B" vers TMP. Ensuite, nous voulons dire à l'ALU quelle opération effectuer, mettre "Reg A" sur le bus et régler la sortie de l'ALU sur ACC. Ensuite, nous voulons déplacer l'ACC vers "Reg B".

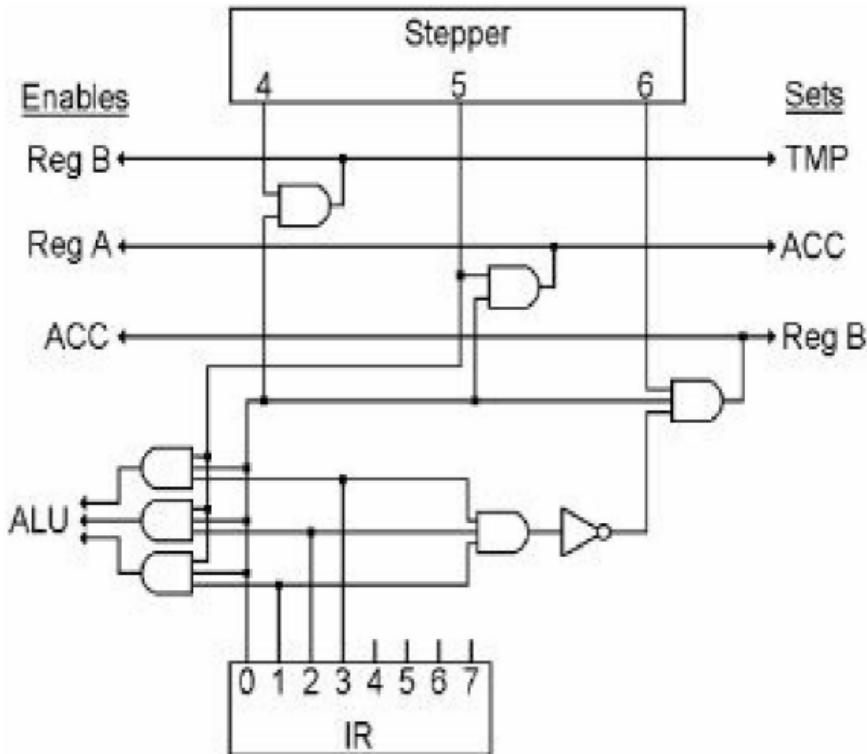
Le bit 0 de l'IR est celui qui détermine s'il s'agit d'un Instruction ALU. Lorsque le bit 0 est activé, les choses que le bit 0 est câblé pour que toutes les étapes d'une instruction ALU se produisent.

Le diagramme suivant montre les huit portes et les fils qui sont ajoutés à la section de contrôle qui font que les étapes 4, 5 et 6 d'une instruction ALU font ce que nous avons besoin qu'elles fassent.

Dans le schéma ci-dessous, juste au-dessus et à gauche de l'IR, il y a trois portes ET. Les sorties de ces portes vont aux trois fils "op" de l'ALU qui lui indiquent quelle opération effectuer. Chacune de ces trois portes ET a trois entrées. Une entrée de chaque porte est câblée au bit 0 de l'IR. Une deuxième entrée de chaque porte est câblée à l'étape 5 du moteur pas à pas. L'entrée restante de chaque porte est câblée aux bits 1, 2 et 3 de l'IR.

Par conséquent, les trois fils qui vont à l'ALU seront 000 à tout moment sauf pendant l'étape 5 où le bit IR 0 se trouve être un 1. À un tel moment, les fils allant à

l'ALU sera la même que les bits 1, 2 et 3 de l'IR.



Le bit IR 0 continue dans le diagramme, tourne à droite et est connecté à un côté de trois autres portes ET. Les autres côtés de ces portes sont connectés aux étapes 4, 5 et 6.

La sortie de la première porte s'allume pendant l'étape 4, et vous pouvez la voir aller à deux endroits. Sur la gauche, il active 'Reg B' sur le bus, et sur la droite, il met le bus en TMP. Cette étape n'est en fait pas nécessaire pour les opérations SHL, SHR et NOT, mais cela ne nuit en rien, et il serait assez compliqué de s'en débarrasser, donc par souci de simplicité, nous allons la laisser ainsi.

La deuxième porte s'allume pendant l'étape 5 (la même étape que l'ALU reçoit ses commandes), et vers la gauche se trouve un fil qui active «Reg A» sur le bus. L'ALU a maintenant une entrée dans TMP, l'autre entrée sur le bus et son fonctionnement spécifié par ces trois fils "op", donc à droite se trouve un fil qui définit la réponse dans ACC.

La troisième porte s'allume pendant l'étape 6. Le fil allant vers la gauche active l'ACC sur le bus, et le fil allant vers la droite règle le bus sur "Reg B". Il n'y a qu'une seule situation spéciale dans une instruction ALU, et c'est lorsque l'opération est CMP, code 111. Pour une opération de comparaison, nous ne voulons pas stocker les résultats dans "Reg B". Par conséquent, il y a une porte ET à trois entrées connectée aux bits IR 1, 2 et 3, qui est ensuite connectée à une porte NON, puis à une troisième entrée sur la porte ET qui effectue l'étape 6 de l'instruction ALU. Ainsi, lorsque l'opération est 111, le premier ET s'allume, le NON s'éteint et la sortie de la porte ET de l'étape 6 ne s'allume pas.

Cette instruction ALU est maintenant terminée. L'étape 7 réinitialise le moteur pas à pas, qui recommence ensuite ses étapes, récupère l'instruction suivante, etc., etc.

Nous allons inventer une chose de plus ici, et c'est une manière abrégée d'écrire des instructions CPU sur une feuille de papier. Dans le code d'instruction, 1000 1011 signifie "Ajouter R2 à R3", mais il faut beaucoup de pratique pour qu'une personne regarde 1000 1011 et pense immédiatement à l'addition et aux registres. Il faudrait également beaucoup de mémorisation pour y penser dans l'autre sens, c'est-à-dire que si vous vouliez XOR deux registres, quel est le code d'instruction pour XOR ? Il serait plus facile d'écrire quelque chose comme ADD R2,R3 ou XOR R1,R1.

Cette idée d'utiliser une sténographie a un nom et s'appelle un langage informatique. Ainsi, en plus d'inventer un code d'instruction, nous allons également inventer un langage informatique qui représente le code d'instruction. L'instruction ALU donne les huit premiers mots de notre nouvelle langue.

Langue		Sens
AJOUTER	RA, RB	Ajoutez RA et RB et mettez la réponse dans

SHR	RA, RB	Décaler RA vers la droite et mettre la réponse i
SHL	RA, RB	Décalez RA à gauche et mettez la réponse
NE PAS	RA, RB	Pas RA et mettre la réponse dans RB
ET	RA, RB	Et RA et RB et mettre la réponse dans
OU	RA, RB	Ou RA et RB et mettre la réponse dans
LIBRE	RA, RB	OU exclusif RA et RB dans RB
CMP	RA, RB	Comparer RA et RB

Lorsqu'une personne veut écrire un programme informatique, elle peut écrirez-le directement dans le code d'instruction, ou utilisez un Langage informatique. Bien sûr, si vous écrivez un programme en un langage informatique, il faudra le traduire en le code d'instruction réel avant qu'il puisse être placé dans RAM et exécuté.

Les instructions de chargement et de stockage

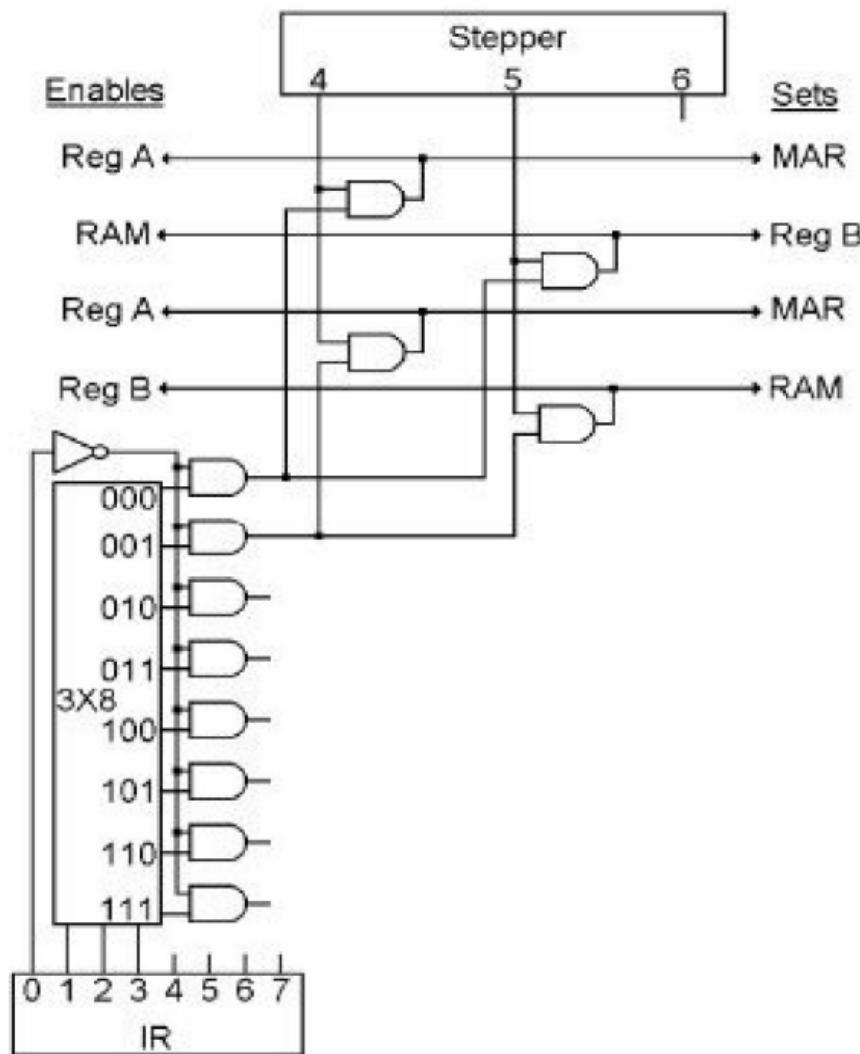
Les instructions de chargement et de stockage sont assez simples. Ils déplacent un octet entre la RAM et un registre. Ils sont très similaires les uns aux autres, nous allons donc les couvrir tous les deux dans un chapitre.

Nous verrons les détails de ces instructions dans un instant, mais nous devons d'abord avoir quelque chose qui nous indique quand nous avons une instruction de chargement ou de stockage dans le registre d'instructions. Avec l'instruction ALU, tout ce que nous avions besoin de savoir, c'était que le bit 0 était activé. Le code pour tous les autres types d'instruction commence avec le bit 0 désactivé, donc si nous connectons une porte NOT au bit 0, lorsque cette porte NOT s'allume, cela nous indique que nous avons un autre type d'instruction. Dans cet ordinateur, il existe huit types d'instructions qui ne sont pas des instructions ALU, donc lorsque le bit 0 est désactivé, nous utiliserons les trois bits suivants de l'IR pour nous dire exactement quel type d'instruction nous avons.

Les trois bits qui sont allés à l'ALU dans un ALU

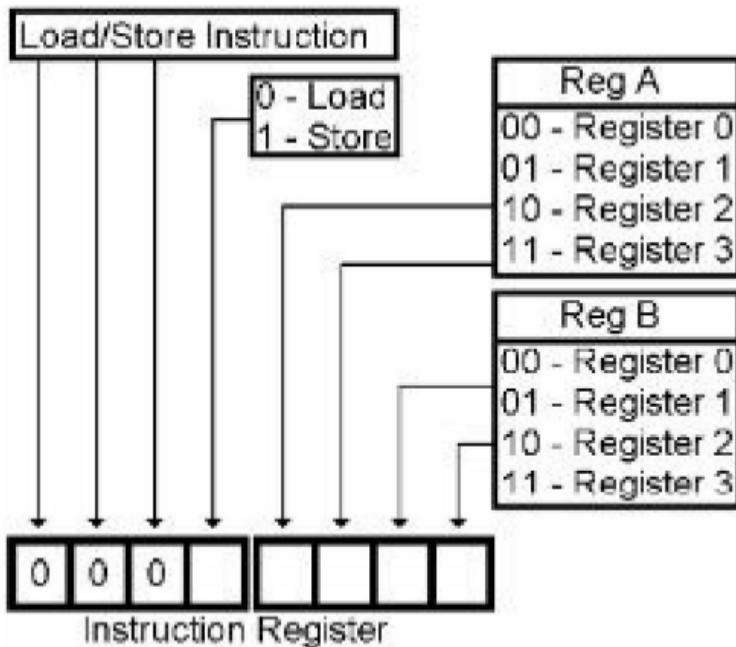
l'instruction va également à un décodeur 3x8 ici dans la section de contrôle. Comme vous vous en souvenez, une et une seule des sorties d'un décodeur est allumée à tout moment, nous aurons donc des portes ET sur les sorties pour empêcher toute sortie d'aller n'importe où pendant une instruction ALU. Mais lorsqu'il ne s'agit pas d'une instruction ALU, la seule sortie du décodeur qui est activée passera par sa porte ET et sera à son tour connectée à d'autres portes qui feront fonctionner l'instruction appropriée.

Dans le diagramme ci-dessous, vous pouvez voir les bits IR 1, 2 et 3 entrer dans un décodeur qui a huit portes ET sur ses sorties. Le bit IR 0 a une porte NON qui va de l'autre côté de ces huit portes ET. Ce décodeur est utilisé pour le reste des instructions que notre ordinateur aura.



Ce chapitre traite des instructions qui utilisent les deux premières sorties du décodeur, celles qui s'allument lorsque l'IR commence par 0000 ou 0001.

La première instruction déplace un octet de la RAM vers un registre, c'est ce qu'on appelle l'instruction "Load". L'autre fait la même chose en sens inverse, il déplace un octet d'un registre vers la RAM, et s'appelle l'instruction "Store".



Le code d'instruction pour l'instruction Load est 0000 et pour l'instruction Store est 0001. Les quatre bits restants dans les deux cas spécifient deux registres, tout comme l'instruction ALU, mais dans ce cas, un registre sera utilisé pour sélectionner l'un des emplacements dans la RAM, et l'autre registre sera soit chargé à partir de cet emplacement RAM, soit stocké dans celui-ci.

L'étape 4 est la même pour les deux instructions. Un de

registres est sélectionné par les bits IR 4 et 5 et est activé sur le bus. Le bus est alors mis en MAR, sélectionnant ainsi une adresse en RAM.

À l'étape cinq, les bits IR 6 et 7 sélectionnent un autre des registres CPU. Pour l'instruction de chargement, la RAM est activée sur le bus et le bus est défini dans le registre sélectionné. Pour l'instruction Store, le registre sélectionné est activé sur le bus et le bus est placé dans la RAM.

Chacune de ces instructions ne nécessite que deux étapes, l'étape 6 ne fera rien.

Voici deux nouveaux mots pour notre langage informatique : Langage

		Sens
LD	RA, RB	Charger RB à partir de l'adresse RAM dans RA
ST	RA,RB	Stocke RB à l'adresse RAM dans RA

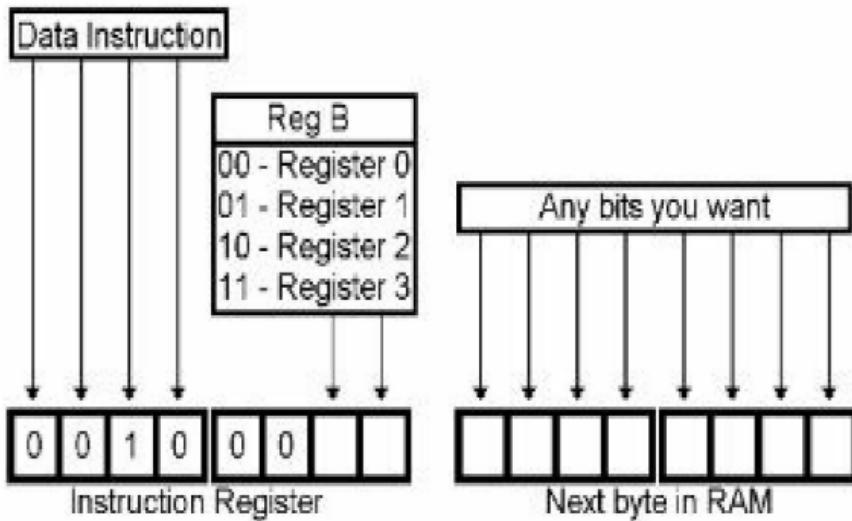
L'instruction de données Voici

maintenant une instruction intéressante. Tout ce qu'il fait est de charger un octet de la RAM dans un registre comme l'instruction Load ci-dessus. La chose qui est différente à ce sujet, c'est l'endroit où, dans la RAM, il obtiendra cet octet.

Dans l'instruction Data, les données proviennent de l'endroit où la prochaine instruction devrait être. Vous pourriez donc considérer que cette instruction fait en réalité deux octets ! Le premier octet est l'instruction, et l'octet suivant est des données qui seront placées dans un registre. Ces données sont faciles à trouver, car au moment où nous avons l'instruction dans l'IR, l'IAR a déjà été mis à jour et pointe donc directement sur cet octet.

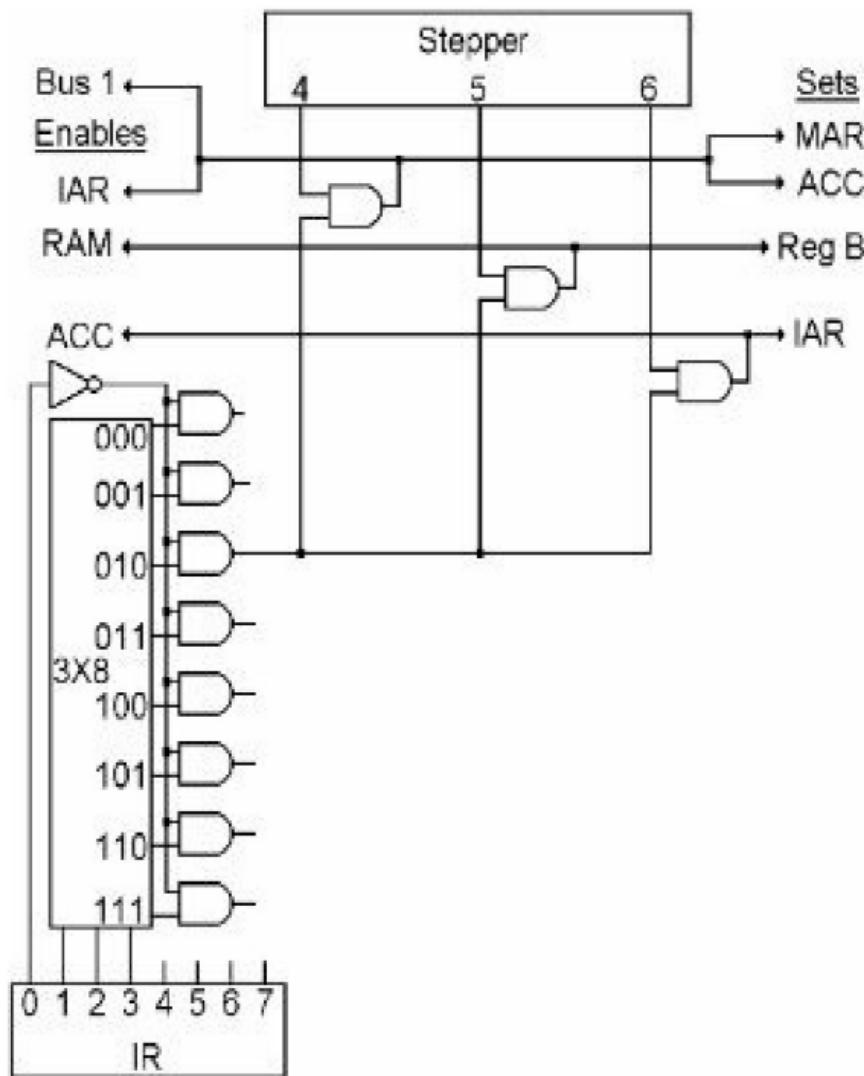
Voici le code d'instruction pour l'instruction Data.

Les bits 0 à 3 sont 0010. Les bits 4 et 5 ne sont pas utilisés. Bits 6 et 7 sélectionner le registre qui sera chargé avec les données qui sont dans le deuxième octet.



Tout ce que cette instruction doit faire est, à l'étape 4, d'envoyer IAR à MAR, et à l'étape 5, d'envoyer de la RAM au CPU souhaité

S'inscrire. Cependant, il y a encore une chose qui doit arriver. Étant donné que le deuxième octet de l'instruction n'est que des données qui pourraient être n'importe quoi, nous ne voulons pas exécuter ce deuxième octet en tant qu'instruction. Nous devons ajouter 1 à l'IAR une seconde fois pour qu'il saute cet octet et pointe vers l'instruction suivante. Nous le ferons de la même manière qu'aux étapes 1 et 3. À l'étape 4, lorsque nous enverrons IAR à MAR, nous profiterons du fait que l'ALU calcule IAR plus quelque chose en même temps, nous allons allumer le "Bus 1" et réglez la réponse sur ACC. L'étape 5 déplace toujours les données vers un registre, et à l'étape 6, nous pouvons déplacer ACC vers IAR.



Voici un autre nouveau mot pour notre langage informatique :

Langue		Sens
LES DONNÉES	RB, xxxx xxxx	Charger ces 8 bits dans RB

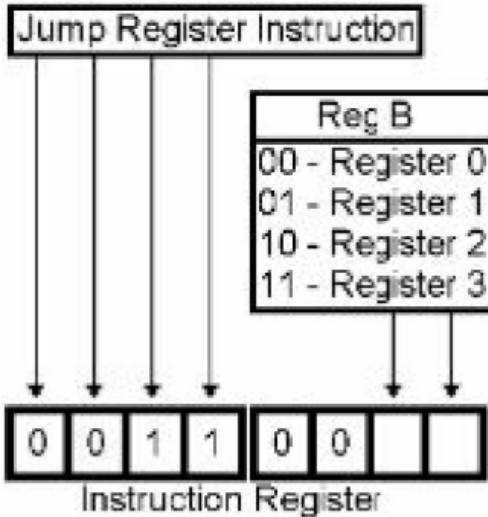
La deuxième grande invention

La première grande invention est cette idée d'avoir une chaîne d'instructions dans la RAM qui sont exécutées une par une par le CPU. Mais notre horloge est très rapide et la quantité de RAM dont nous disposons est limitée. Que se passera-t-il, en bien moins d'une seconde, lorsque nous aurons exécuté toutes les instructions en RAM ?

Heureusement, nous n'aurons pas à répondre à cette question, car quelqu'un a proposé un autre type d'instruction qui est si important qu'il est considéré comme la deuxième grande invention nécessaire pour permettre à l'ordinateur de faire ce qu'il fait. En raison de la disposition polyvalente de notre CPU et de sa section de contrôle, c'est une chose extrêmement simple à faire fonctionner, mais son importance ne doit pas être perdue à cause de cette simplicité.

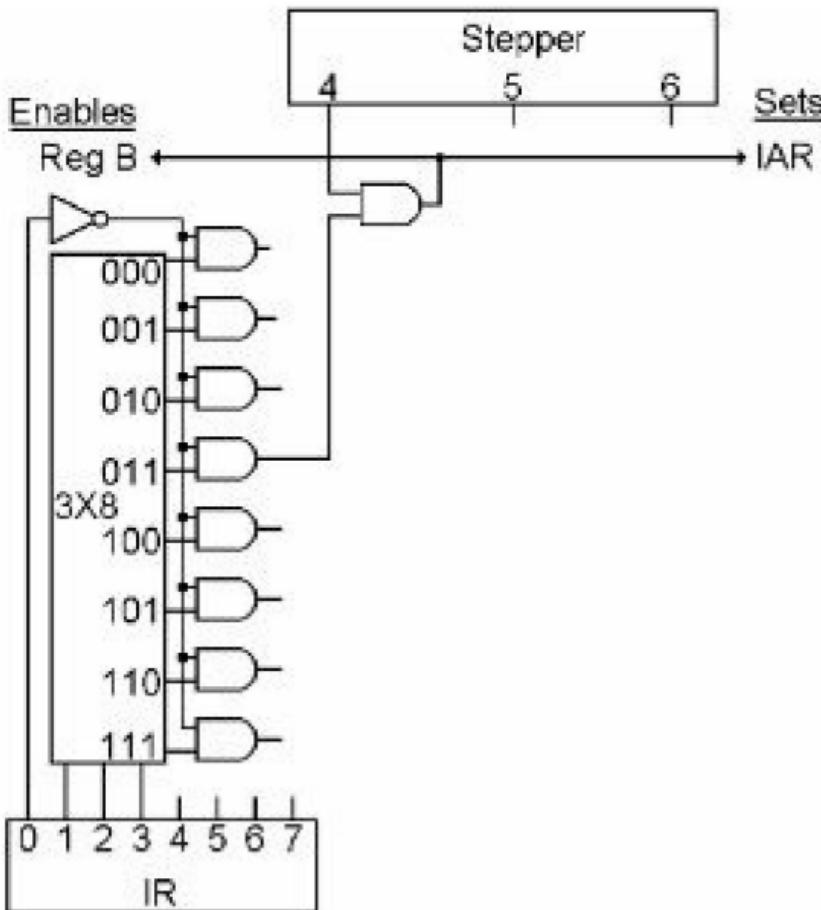
Ce nouveau type d'instruction s'appelle une instruction de saut, et tout ce qu'il fait est de changer le contenu de l'IAR, changeant ainsi d'où dans la RAM proviendront les instructions suivantes et suivantes.

Le type exact d'instruction de saut décrit dans ce chapitre est appelé une instruction « Jump Register ». Il déplace simplement le contenu de Reg B dans l'IAR. Voici le code d'instruction pour cela:



L'ordinateur exécute une série d'instructions dans la RAM, l'une après l'autre, et soudain l'une de ces instructions modifie le contenu de l'IAR. Que se passera-t-il alors ? La prochaine instruction qui sera récupérée ne sera pas celle qui suit la dernière. Ce sera celui qui se trouve à l'adresse RAM chargée dans l'IAR. Et il continuera à partir de ce point avec le suivant, etc. jusqu'à ce qu'il exécute une autre instruction de saut.

Le câblage de l'instruction Jump Register ne nécessite qu'une seule étape. À l'étape 4, le registre sélectionné est activé sur le bus et défini dans l'IAR, et c'est tout. Si nous voulions accélérer notre CPU, nous pourrions utiliser l'étape 5 pour réinitialiser le stepper. Mais pour garder notre schéma simple, nous ne nous en soucierons pas. Les étapes 5 et 6 ne feront rien.

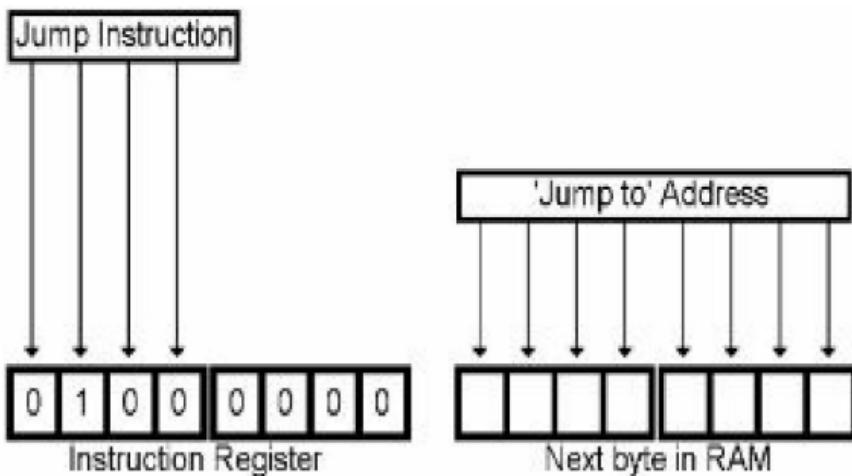


Voici un autre nouveau mot pour notre langage informatique :

Langue		Sens
JMPR	RB	Aller à l'adresse dans RB

Une autre façon de sauter

C'est un autre type d'instruction de saut. Elle est similaire à l'instruction Data en ce sens qu'elle utilise deux octets. Il remplace l'AR par l'octet qui se trouve dans la RAM immédiatement après l'octet d'instruction, modifiant ainsi d'où proviendront les instructions suivantes et suivantes dans la RAM. Voici le code d'instruction pour cela. Les bits 4, 5, 6 et 7 ne sont pas utilisés dans cette instruction :

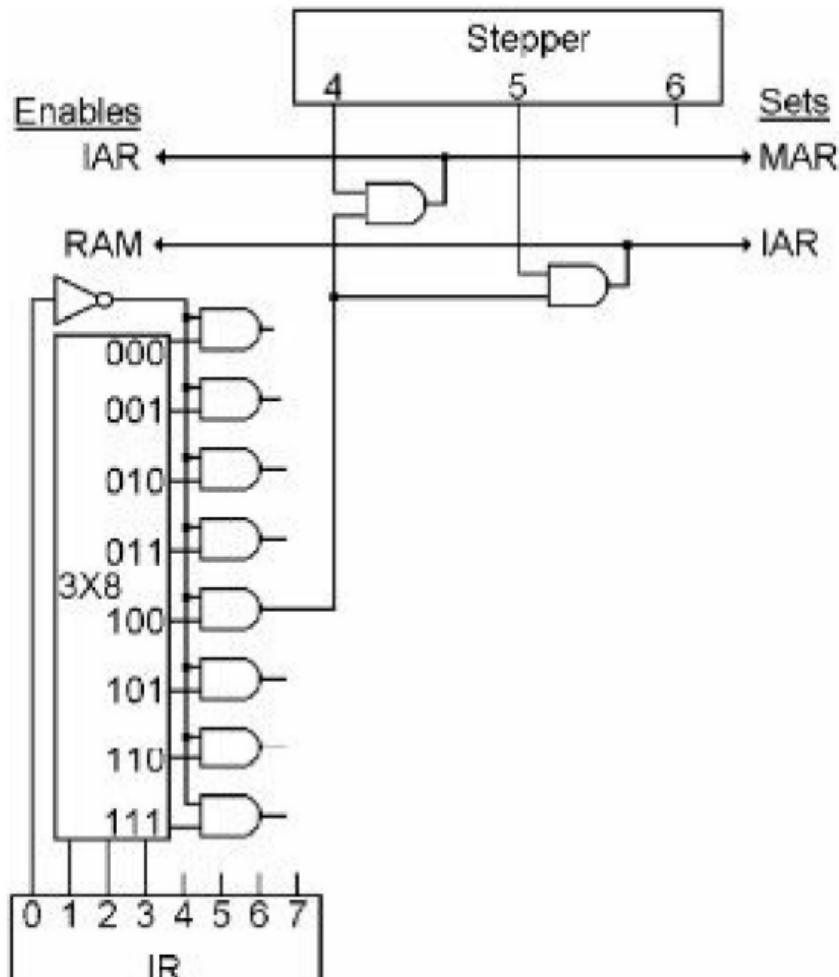


Ce type exact d'instruction de saut s'appelle simplement un "saut". C'est utile quand vous connaissez l'adresse à laquelle vous voulez sauter, quand vous écrivez le programme. L'instruction de saut de registre est plus utile lorsque l'adresse à laquelle vous allez vouloir sauter est calculée comme le programme en cours d'exécution et peut ne pas toujours être la même.

L'une des choses que vous pouvez faire avec une instruction Jump est de créer une boucle d'instructions qui s'exécutent encore et encore. Vous pouvez avoir une série de cinquante instructions en RAM, et la dernière instruction "Revient" à la première.

Comme l'instruction Data, l'IAR pointe déjà sur l'octet dont nous avons besoin. Contrairement à l'instruction de données, nous n'avons pas besoin d'ajouter 1 à l'IAR une deuxième fois car nous allons le remplacer de toute façon. Nous n'avons donc besoin que de deux étapes. À l'étape 4, nous envoyons IAR à MAR. À l'étape 5, nous déplaçons l'octet de RAM sélectionné vers l'IAR. L'étape 6 ne fera rien.

Voici le câblage qui le fait fonctionner :



Voici un autre nouveau mot pour notre langage informatique :

Langue		Sens
--------	--	------

JMP

Addr Aller à l'adresse dans l'octet suivant

La troisième grande invention

Voici la troisième et dernière invention qui fait d'un ordinateur un ordinateur.

C'est comme l'instruction de saut, mais parfois ça saute, et parfois non. Bien sûr, sauter ou ne pas sauter n'est que deux possibilités, il suffit donc d'un bit pour déterminer ce qui se passera. La plupart du temps, ce que nous allons introduire dans ce chapitre, c'est d'où vient ce bit.

Vous souvenez-vous du bit "Carry" qui sort et retourne dans l'ALU ? Ce bit provient soit de l'additionneur, soit d'un des shifters. Si vous additionnez deux nombres qui donnent un montant supérieur à 255, le bit de retenue s'activera. Si vous décalez à gauche un octet dont le bit gauche est activé ou que vous décalez à droite un octet dont le bit droit est activé, ces situations activeront également le bit d'exécution de l'ALU.

Il y a aussi un bit qui nous dit si les deux entrées de l'ALU sont égales, un autre qui nous dit si l'entrée A est plus grande, et un autre bit qui nous dit si la sortie de l'ALU est entièrement à zéro.

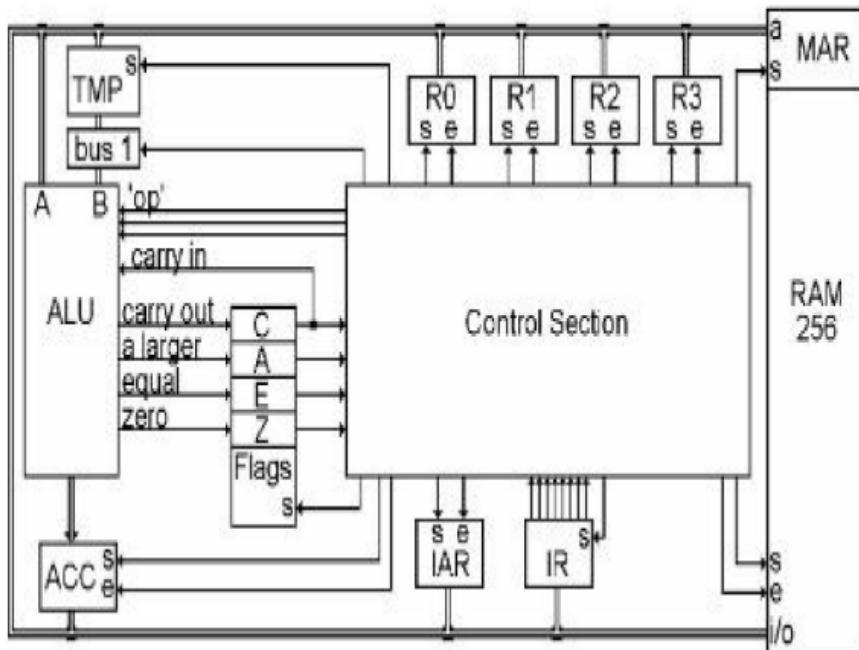
Ces bits sont les seules choses pour lesquelles nous n'avons pas encore trouvé de place dans le CPU. Ces quatre bits seront appelés les bits "Flag", et ils seront utilisés pour prendre la décision d'une instruction "Jump If" quant à savoir si elle exécutera l'instruction suivante dans la RAM ou sautera à une autre adresse.

Ce que nous essayons de faire en sorte que l'ordinateur puisse accomplir, c'est qu'il exécute d'abord une instruction ALU, puis qu'une ou plusieurs instructions "Sauter si" la suivent. Le "Jump If" sautera ou non en fonction de quelque chose qui s'est passé pendant l'instruction ALU.

Bien sûr, au moment où le "Jump If" est exécuté, les résultats de l'instruction ALU ont disparu depuis longtemps. Si vous revenez en arrière et regardez les détails de l'instruction ALU, ce n'est qu'à l'étape 5 que toutes les entrées appropriées entrent dans l'ALU et que la réponse souhaitée sort.

C'est à ce moment que la réponse est mise dans ACC. La synchronisation est la même pour les quatre bits de drapeau, ils ne sont que

valide pendant l'étape 5 de l'instruction ALU. Par conséquent, nous avons besoin d'un moyen de sauvegarder l'état des bits Flag tels qu'ils étaient lors de l'étape 5 de l'instruction ALU.



Voici le dernier registre que nous allons ajouter au CPU. Cela s'appellera le registre FLAG, et nous n'en utiliserons que quatre bits, un pour chacun des drapeaux.

Les bits de drapeau de l'ALU sont connectés à l'entrée de ce registre, et il sera défini lors de l'étape 5 de l'instruction ALU tout comme ACC et il restera défini de cette façon jusqu'à la prochaine fois qu'une instruction ALU sera exécutée.

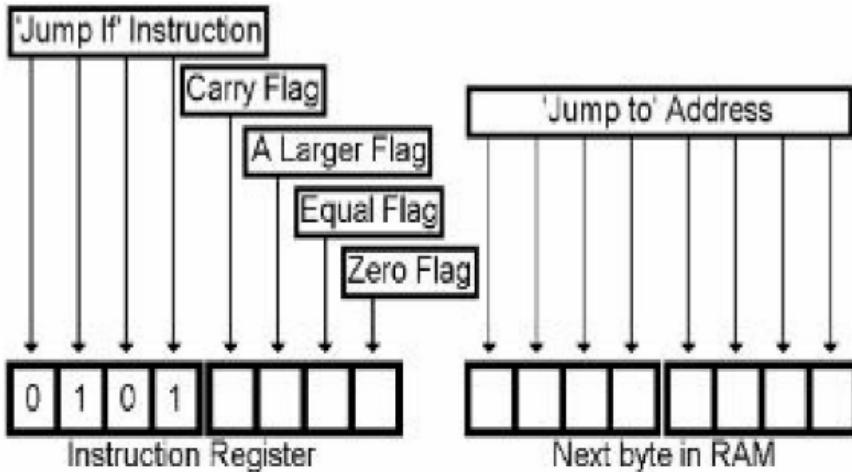
Ainsi, si vous avez une instruction ALU suivie d'une instruction "Jump If", les bits "Flag" peuvent être utilisés pour "décider" de sauter ou non.

Chaque cycle d'instruction utilise l'ALU à l'étape 1 pour ajouter 1

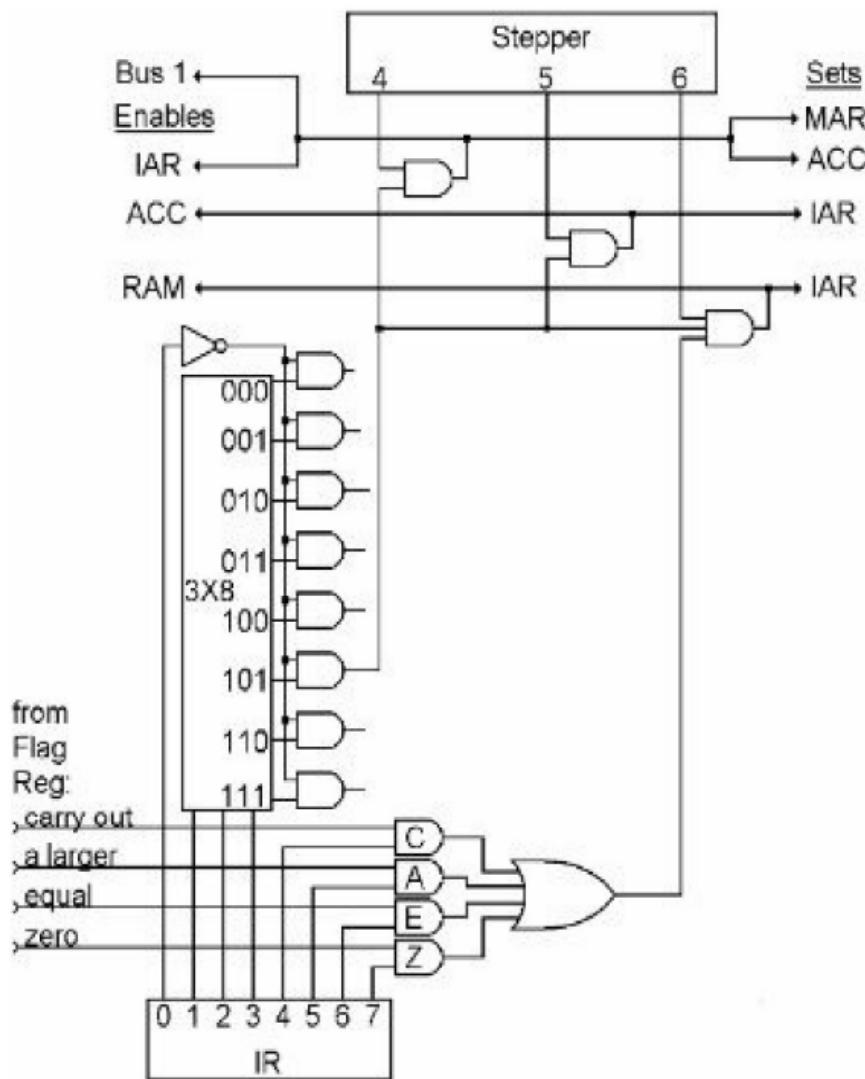
à l'adresse de l'instruction suivante, mais seule l'étape 5 de l'instruction ALU a une connexion qui définit les drapeaux. (Nous n'avons pas montré cette connexion dans le câblage de l'instruction ALU car nous n'avions pas encore introduit le Flag Reg, mais il apparaîtra dans le schéma de la section de contrôle complété.)

Cette combinaison de bits Flag et de l'instruction Jump IF est la troisième et dernière grande invention qui fait fonctionner les ordinateurs tels que nous les connaissons aujourd'hui.

Voici le code d'instruction pour une instruction 'Sauter si'. Les quatre seconds bits de l'instruction indiquent au CPU quel indicateur ou quels indicateurs doivent être vérifiés. Vous mettez un '1' dans le(s) bit(s) d'instruction correspondant au(x) drapeau(s) que vous voulez tester. Si l'un des drapeaux que vous testez est activé, le saut aura lieu. Cet arrangement nous donne un certain nombre de façons de décider de sauter ou non. Il y a un deuxième octet qui contient l'adresse à laquelle sauter, si le saut est effectué.



Voici le câblage dans la section de contrôle qui fait fonctionner l'instruction Jump If.



L'étape 4 déplace IAR vers MAR afin que nous soyons prêts à obtenir le 'Aller à l'adresse' que nous utiliserons SI nous sautons. Mais parce que nous pourrions ne pas sauter, nous devons également calculer le adresse de la prochaine instruction en RAM. Et donc étape 4 active également le bus 1 et définit la réponse dans ACC.

À l'étape 5, nous déplaçons ACC vers IAR afin d'être prêts à récupérer la prochaine instruction SI nous ne sautons pas.

L'étape 6 est celle où la « décision » est prise. Nous déplacerons le deuxième octet de l'instruction de RAM à IAR SI le troisième entrée de cette porte ET est activée. Cette troisième entrée provient d'une porte OU à quatre entrées. Ces quatre les entrées proviennent des quatre bits de drapeau après avoir été combinées par AND avec les quatre derniers bits de l'instruction Jump If dans IR. Si, par exemple, il y a un '1' dans le bit 'Equal' de l'instruction, et le bit de drapeau 'Equal' est activé, alors le saut aura lieu.

Voici plus de mots pour notre langage informatique. 'J' signifie Sauter, 'C' signifie Carry, 'A' signifie A est plus grand, 'E' signifie A est égal à B et 'Z' signifie que la réponse est entièrement composée de zéros.

Voici les mots de la langue qui testent un seul

Drapeau:

Langue		Sens
JC	Adr Jump	si Carry est activé
ET	Addr Jump	si A est plus grand que B
EST	Addr Jump	si A est égal à B
JZ	Addr Jump	si la réponse est zéro

Vous pouvez également tester plusieurs bits d'indicateur en même temps temps en mettant un 1 dans plus d'un des quatre bits.

En fait, puisqu'il y a quatre bits, il y a 16 combinaisons possibles, mais celle avec les quatre bits off n'est pas utile car il ne sautera jamais. Pour le par souci d'exhaustivité, voici le reste de la possibilités :

Langue		Sens
JCA	Adresse	Sauter si Carry ou A plus grand
ECJ	Adresse	Sauter si porter ou A égal B
JCZ	Adresse	Sauter si Carry ou la réponse est zéro

JAE	Adresse	Sauter si A est supérieur ou égal à B
MOI	Adresse	Sauter si A est plus grand ou si la réponse est zéro
JEZ	Adresse	Sauter si A est égal à B ou si la réponse est zéro
JCAE	Adresse	Sauter si Carry ou A plus grand ou égal à
JCAZ	Adresse	Sauter si Porter ou A plus grand ou Zéro
JCEZ	Adresse	Sauter si Carry ou A est égal à B ou zéro
JAEZ	Adresse	Sauter si A supérieur ou égal à B ou zéro
JCAEZ	Adresse	Sauter si Porter, A plus grand, Égal ou Zéro

L'instruction Clear Flags

Il y a un détail ennuyeux que nous devons avoir ici.

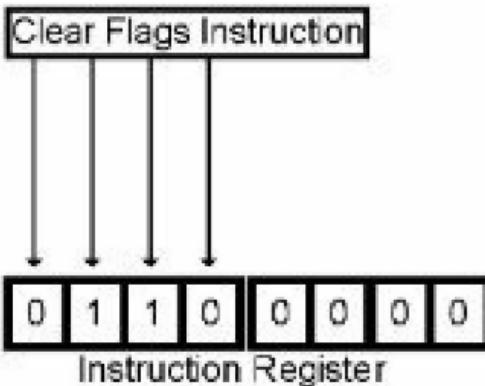
Lorsque vous effectuez une addition ou un décalage, vous avez la possibilité d'activer le drapeau de retenue par l'opération. Ceci est nécessaire, nous l'utilisons pour l'instruction Jump If comme dans le chapitre précédent.

Le Carry Flag est également utilisé comme entrée pour les opérations d'addition et de décalage. Le but est que vous puissiez additionner des nombres supérieurs à 255 et décaler des bits d'un registre à un autre.

Le problème qui se pose est que si vous ajoutez simplement deux nombres à un octet, vous ne vous souciez pas des Carry précédents, mais le Carry Flag peut toujours être défini à partir d'une opération précédente. Dans ce cas, vous pourriez ajouter $2+2$ et obtenir 5 !

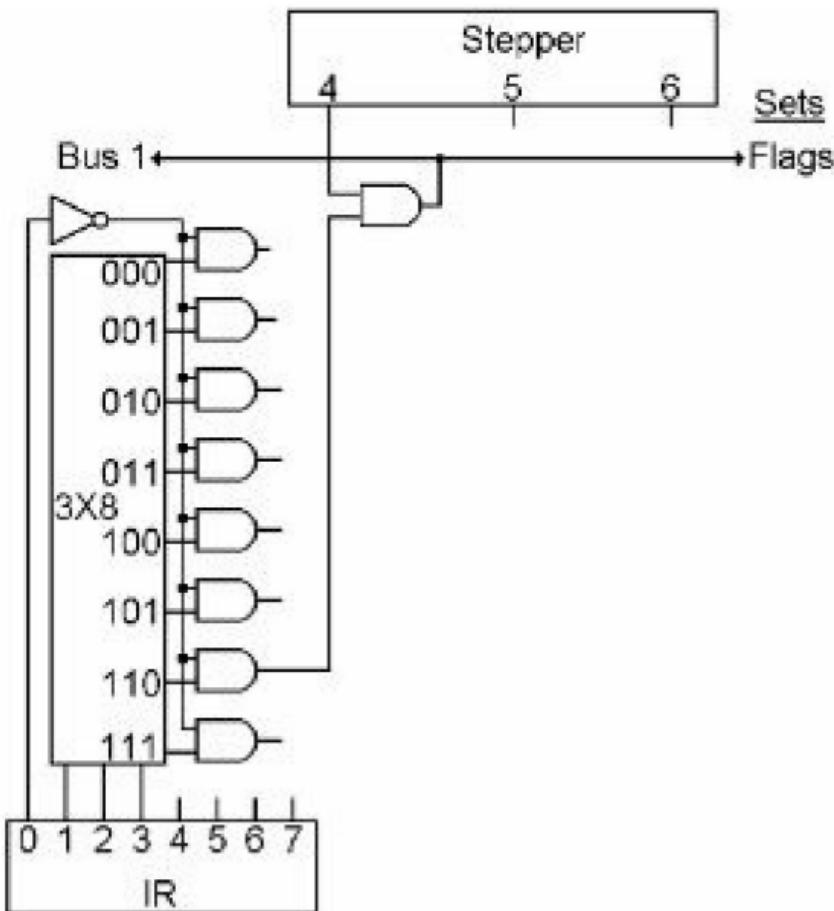
Les ordinateurs plus gros ont plusieurs façons de le faire, mais pour nous, nous aurons juste une instruction Clear Flags que vous devez utiliser avant tout ajout ou changement où un bit de report inattendu serait un problème.

Voici le code d'instruction pour cette instruction. Les bits 4, 5, 6 et 7 ne sont pas utilisés.



Le câblage pour cela est très simple et un peu délicat. Nous n'activerons rien sur le bus, donc lui et l'entrée ALU 'A' seront tous des zéros. Nous allons activer 'Bus 1'

donc l'entrée 'B' est 0000 0001. Nous n'enverrons pas d'opération à l'ALU, elle sera donc en mode ADD. L'ALU ajoutera donc 0 et 1, et il peut y avoir une entrée de report. La réponse sera alors soit 0000 0001 soit 0000 0010. Mais il n'y aura pas de report de sortie, la réponse n'est pas zéro et B est plus grand que A donc 'égal' et 'A plus grand' seront tous les deux désactivés. Nous "définissons" le Flag Reg à ce moment alors que les quatre bits de Flag sont désactivés.

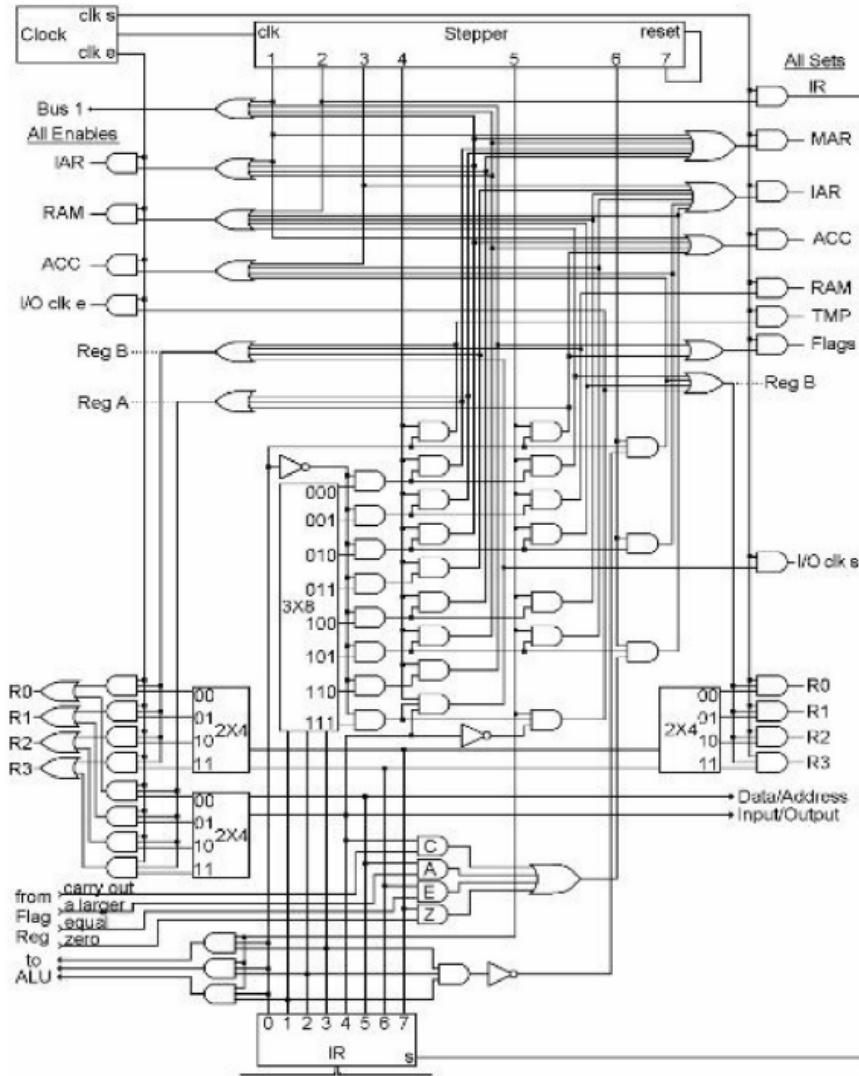


Voici un autre mot pour notre langue.

Langue	Sens
FLC	Effacer tous les drapeaux

Ta Daa !

Nous avons maintenant câblé la section de contrôle de notre CPU. En conséquence, nous pouvons placer une série d'instructions dans la RAM, et l'horloge, le stepper, le registre d'instructions et le câblage récupéreront et exécuteront ces instructions. Voici toute la section contrôle :



Oui, cela semble assez compliqué, mais nous avons regardé à chaque partie de celui-ci déjà. La seule chose que nous devions ajouter étaient des portes OU parce que la plupart des "permet" et 'sets' ont besoin de plusieurs connexions. Cela a en fait un beaucoup moins de pièces que la RAM, mais c'était beaucoup plus répétitif. La plupart des dégâts ici ne font que recevoir le fils d'un endroit à un autre.

L'octet placé dans le registre d'instructions provoque une certaine activité. Chaque possible modèle provoque une activité différente. Par conséquent, nous avons un code où chacun des 256 codes possibles représente un activité spécifique différente.

Comme mentionné, cela s'appelle le code d'instruction. Un autre nom pour cela est « langage machine », car ce est le seul langage (code) que la machine (ordinateur) "comprend". Vous "dites" à la machine ce qu'elle doit faire en en lui donnant une liste des commandes que vous voulez qu'il exécute. Mais vous devez parler la seule langue qu'il "comprend". Si vous l'alimentez avec la bonne taille d'octet modèles de marches et arrêts, vous pouvez lui faire faire quelque chose ce sera utile.

Voici tous les codes d'instruction et notre raccourci langue réunie en un seul lieu.

Code d'instructions		Langue		Sens
1000	rare	AJOUTER	RA, RB	Ajouter
1001	rare	SHR	RA, RB	Décalage à droite
1010	rare	SHL	RA, RB	Décalage à gauche
1011	rare	NE PAS	RA, RB	Pas
1100	rare	ET	RA, RB	Et
1101	rare	OU	RA, RB	Ou
1110	rare	LIBRE	RA, RB	O exclusif
1111	rare	CMP	RA, RB	Comparer
0000	rare	LD	RA, RB	Charger RB depuis
0001	rare	ST	RA, RB	Stocker RB à
0010	00rb xxxxxxxx	LES DONNÉES	RB, Données	Chargez ces 8
0011	00rb	JMPR	RB	Sauter à la
0100	0000 xxxxxxxx	JMP	Adresse	Sauter à la

0101	Caez xxxxxxx	JCAEZ	Adresse	Sauter le cas échéant
0110	0000	FLC		Tout supprimer

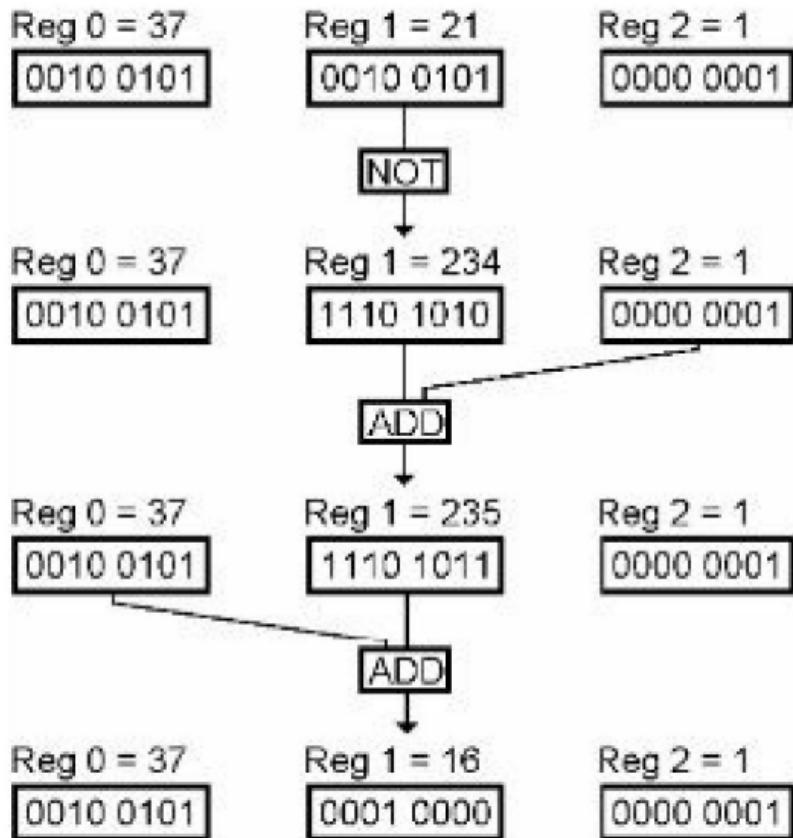
Croyez-le ou non, tout ce que vous avez déjà vu
ordinateur, est simplement le résultat d'un CPU exécutant un
longue série d'instructions telles que celles ci-dessus.

Quelques mots de plus sur l'arithmétique

Nous ne voulons pas passer beaucoup de temps sur ce sujet, mais la seule chose que nous ayons vue jusqu'à présent qui ressemble à de l'arithmétique est l'additionneur, nous allons donc examiner des exemples simples d'arithmétique légèrement plus complexe. Pas pour vous apprendre à agir comme un ordinateur, mais juste pour vous prouver que ça marche.

Voici comment faire une soustraction. Cela se fait avec l'additionneur et les portes NOT. Si vous voulez soustraire R1 de R0, vous devez d'abord PAS R1 revenir en lui-même. Ensuite, vous ajoutez 1 à R1, puis vous ajoutez R0 à R1.

Ceci montre un exemple de soustraction de 21 à 37 :



La dernière étape consiste à ajouter 37 + 235, dont la réponse devrait être 272. Mais un seul registre ne peut pas contenir un nombre supérieur à 255. Par conséquent, l'additionneur active son bit de retenue et les huit bits restants de la réponse sont 0001 0000, qui est 16, la bonne réponse pour 37 moins 21.

Pourquoi NOTting et ADDing entraînent-ils une soustraction ? Pourquoi devez-vous ajouter 1 après NOTting ? Pourquoi ignores-tu

le bit de transport? Nous n'allons pas tenter de répondre à ces questions dans ce livre. Ce sont ces détails qui empêchent très peu d'ingénieurs de passer une bonne nuit de sommeil. Ces personnes courageuses étudient ces problèmes et conçoivent des moyens pour que les gens ordinaires n'aient pas à les comprendre.

Voici comment faire la multiplication. Quand on fait des multiplications avec un crayon et du papier dans le système décimal, il faut se souvenir de ses tables de multiplication, tu sais, 3 fois 8 égale 24, 6 fois 9 égale 54, etc.

En binaire, la multiplication est en fait beaucoup plus facile qu'en décimal. 1 fois 1 égale 1, et pour toute autre combinaison, la réponse est 0 ! Cela ne pourrait tout simplement pas être beaucoup plus simple que cela ! Voici un exemple de multiplication de 5 fois 5 avec un crayon et du papier en binaire.

$$\begin{array}{r}
 00000101 \\
 \times 00000101 \\
 \hline
 00000101 \\
 00000000 \\
 00000101 \\
 00000000 \\
 00000000 \\
 00000000 \\
 00000000 \\
 \hline
 00000000011001
 \end{array}$$

Si vous regardez ce qui se passe ici, si le chiffre de droite du chiffre du bas est un 1, vous mettez le chiffre du haut dans la réponse. Ensuite, pour chaque chiffre à gauche de celui-ci, décalez le chiffre du haut vers la gauche, et si le chiffre du bas est un 1, ajoutez le chiffre du haut décalé à la réponse. Lorsque vous parcourez les huit bits du nombre inférieur, vous êtes

Fini.

Donc la multiplication est accomplie avec l'additionneur et le manettes. C'est aussi simple que ça. Vous pouvez écrire un simple programme comme ceci :

R0 contient le numéro du bas, R1 contient le numéro du haut
nombre et R2 contiendra la réponse. R3 est utilisé pour
sortir de la boucle après l'avoir parcourue huit fois.

Adresse RAM	Instruction		commentaires
50	LES DONNÉES	R3,0000 0001	* Mettez '1' dans R3
52	LIBRE	R2,R2 * Mettez '0' dans R2	
53	FLC		* Effacer les drapeaux
54	SHR	R0	* Un peu pour transporter Fla
55	JC	59	* Faites l'AJOUT
57	JMP	61	* Sauter l'ajout
59	FLC		* Effacer les drapeaux
60	AJOUTER	R1,R2 * AJOUTER cette ligne	
61	FLC		* Effacer les drapeaux
62	SHL	R1	* Haut multiple par 2
63	SHL	R3	* Compteur de décalage
64	JC	68	* Out si fait
66	JMP	53	* Faites l'étape suivante
68	(Prochaine instruction dans le programme)		

Voyez ce qui se passe avec les registres au fur et à mesure de ce programme
dans sa boucle les trois premières fois.

R0

R2

R1

R3

Au début (après 52):

0000 0101	0000 0101	0000 0000	0000 0001
-----------	-----------	-----------	-----------

Première fois (après 63):

0000 0010	0000 1010	0000 0101	0000 0010
-----------	-----------	-----------	-----------

Deuxième fois (après 63):

0000 0001	0001 0100	0000 0101	0000 0100
-----------	-----------	-----------	-----------

Troisième fois (après 63):

0000 0000	0010 1000	0001 1001	0000 1000
-----------	-----------	-----------	-----------

La chose importante qui s'est produite ici est que R1 a été ajouté deux fois à R2. Cela s'est produit la première fois, lorsque R1 contenait 0000 0101, et la troisième fois, après que R1 ait été décalé deux fois vers la gauche et contenait donc 0001 0100. R2 contient maintenant 0001 1001 binaire, soit $16 + 8 + 1$, ou 25 décimal, qui est la bonne réponse pour 5 fois 5. La boucle se répétera 5 fois de plus jusqu'à ce que le bit de R3 soit décalé vers le Carry Flag, mais le total n'augmentera pas car il n'y a plus de 1 dans R0.

Ce programme passera huit fois. On commence par 0000 0001 en R3. Vers la fin du programme, R3 est décalé vers la gauche. Les sept premières fois, il n'y aura pas de report, donc le programme arrivera au 'JMP 53' et remontera à la troisième instruction du programme. La huitième fois que R3 est décalé vers la gauche, le bit qui est activé est décalé de R3 vers le drapeau Carry.

Par conséquent, le « JC 68 » sautera par-dessus le « JMP 53 » et continuera avec les instructions qui suivront.

L'octet dans R0 est décalé vers la droite pour tester quels bits sont activés. L'octet dans R1 est décalé vers la gauche pour le multiplier par deux. Quand il y avait un bit dans R0, vous ajoutez R1 à R2. Et c'est tout ce qu'il y a à faire.

Une chose que nous n'abordons pas dans cet exemple est ce qui se passe si la réponse de la multiplication est supérieure à 255. Si un programme de multiplication multiplie deux nombres à un octet, il devrait être capable de gérer une réponse à deux octets. Cela prendrait soin de deux nombres avec lesquels vous pourriez commencer. Cela serait accompli avec le drapeau de portage et quelques instructions supplémentaires de saut si. Nous ne torturerons pas le lecteur avec les détails.

La lecture d'un programme comme celui ci-dessus est une compétence entièrement différente de la lecture des diagrammes et des graphiques que nous avons vus jusqu'à présent dans le livre. J'espère que vous avez pu le suivre, mais personne ne devrait devenir un expert en lecture de programmes à cause de ce livre.

La division peut également être effectuée par notre ordinateur. Il y a plusieurs façons de le faire, et nous n'allons pas

examiner l'un d'eux en détail. Imaginez la méthode simple suivante. Disons que vous voulez diviser quinze par trois. Si vous soustrayez à plusieurs reprises trois de quinze et que vous comptez le nombre de soustractions que vous pouvez accomplir avant que les quinze ne soient épuisés, ce décompte sera la réponse. Comme ces cinq étapes : (1) $15-3=12$, (2) $12-3=9$, (3) $9-3=6$, (4) $6-3=3$, (5) $3-3=0$. Cela se transforme facilement en programme.

Les ordinateurs ont également des moyens de gérer les nombres négatifs et les nombres avec des points décimaux. Les détails sont très fastidieux, et les étudier n'améliorerait pas notre compréhension du fonctionnement des ordinateurs. Cela se résume toujours à rien de plus que des portes NAND. Notre simple ordinateur pourrait faire toutes ces choses avec des programmes.

Le monde extérieur

Ce que nous avons décrit jusqu'à présent est l'ensemble de l'ordinateur. Il comporte deux parties, la RAM et le CPU. C'est tout ce qu'il y a.

Ces opérations simples sont les choses les plus compliquées qu'un ordinateur puisse faire. La possibilité d'exécuter des instructions, de modifier des octets avec l'ALU, la possibilité de passer d'une partie du programme à une autre et, surtout, la possibilité de sauter ou non en fonction du résultat d'un calcul. C'est ce qu'un ordinateur est capable de faire. Ce sont des choses simples, mais comme il fonctionne si rapidement, il peut effectuer un grand nombre de ces opérations qui peuvent aboutir à quelque chose d'impressionnant.

Ces deux parties en font un ordinateur, mais si tout ce que l'ordinateur pouvait faire était d'exécuter un programme et de réorganiser les octets dans la RAM, personne ne saurait jamais ce qu'il fait. Il y a donc encore une chose dont l'ordinateur a besoin pour être utile, et c'est un moyen de communiquer avec le monde extérieur.

Traiter avec quoi que ce soit en dehors de l'ordinateur s'appelle 'Entrée/Sortie' ou 'E/S' en abrégé. La sortie signifie que les données sortent de l'ordinateur ; L'entrée signifie que les données arrivent dans l'ordinateur. Certaines choses ne sont que des entrées, comme un clavier, certaines choses ne sont que des sorties, comme un écran d'affichage, certaines choses font à la fois des entrées et des sorties, comme un disque.

Tout ce dont nous avons besoin pour les E/S est de quelques fils et d'une nouvelle instruction.

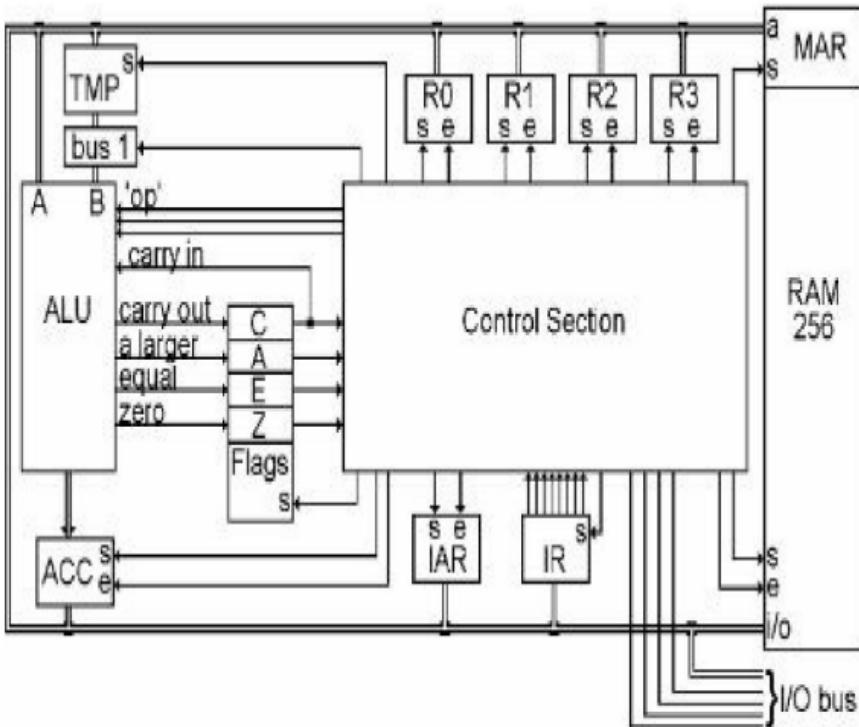
Pour les fils, tout ce que nous allons faire est d'étendre le bus CPU à l'extérieur de l'ordinateur et d'ajouter quatre fils supplémentaires pour l'accompagner. Cette combinaison de 12 fils sera appelée le bus d'E/S. Tout ce qui est connecté à l'ordinateur est attaché à ce bus d'E/S.

Les périphériques connectés au bus d'E/S sont appelés « périphériques », car ils ne sont pas à l'intérieur de l'ordinateur, ils sont à l'extérieur de l'ordinateur, à sa périphérie (la zone qui l'entoure).

Plusieurs éléments peuvent être connectés au bus d'E/S, mais l'ordinateur contrôle le processus et un seul de ces éléments est actif à la fois.

Chaque chose attachée au bus d'E/S doit avoir sa propre adresse d'E/S unique. Ce n'est pas la même chose que les adresses des octets dans la RAM, c'est juste un "nombre" que le périphérique reconnaîtra lorsqu'il sera placé sur le bus.

Voici à quoi ressemble le bus d'E/S dans le CPU, en bas à droite du dessin.



Dans le schéma ci-dessous se trouvent les fils du bus d'E/S. Le bus CPU est le même faisceau de huit fils qui va partout ailleurs. Le fil 'Entrée/Sortie' détermine la direction dans laquelle les données se déplaceront sur le bus CPU, soit vers l'intérieur, soit vers la sortie. Le fil 'Données/Adresse' nous indique si nous allons transférer un octet de données, ou un

Adresse d'E/S qui sélectionne l'un des nombreux périphériques pouvant être connectés au bus d'E/S. 'I/O Clk e' et 'I/O Clk s' sont utilisés pour activer et définir les registres afin que les octets puissent être déplacés d'avant en arrière.

CPU Bus _____

Input/Output _____

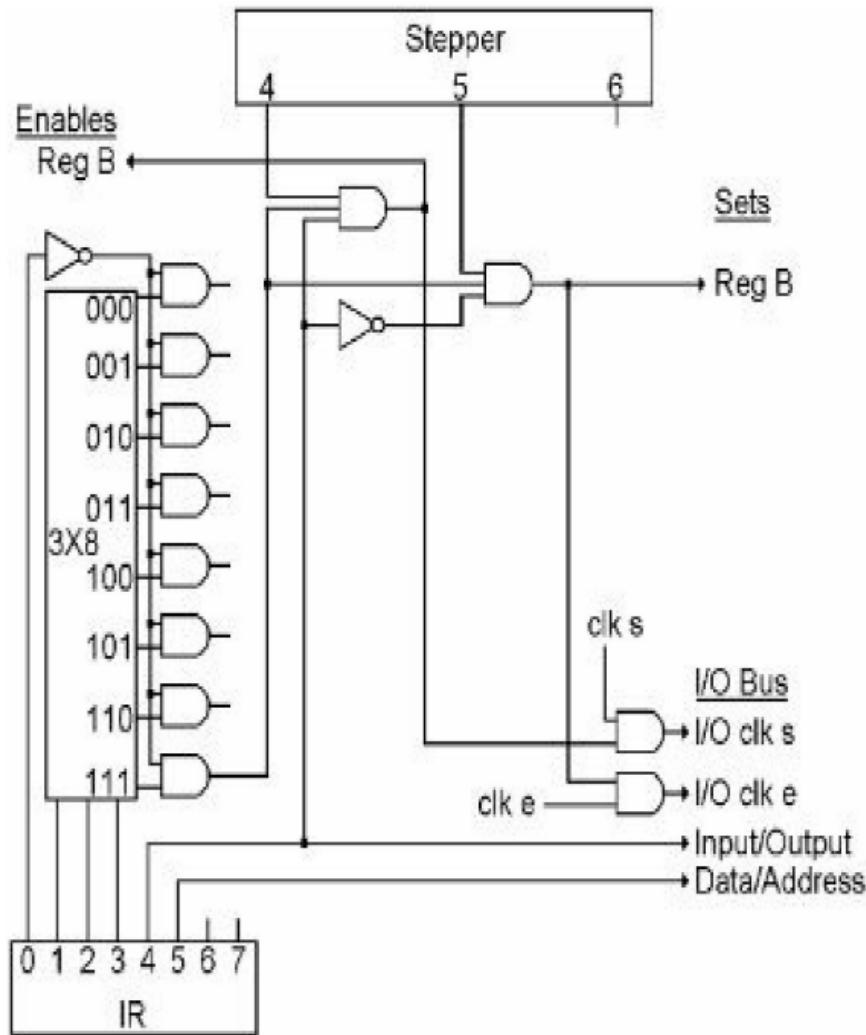
Data/Address _____

I/O clk e _____

I/O clk s _____

Voici le câblage de la section de contrôle pour la nouvelle instruction qui contrôle le bus d'E/S. Cela montre d'où viennent les quatre nouveaux fils pour le bus d'E/S. Ils sont en bas à droite du dessin. Ils figuraient également sur le diagramme de la section de contrôle complet quelques chapitres plus tôt.

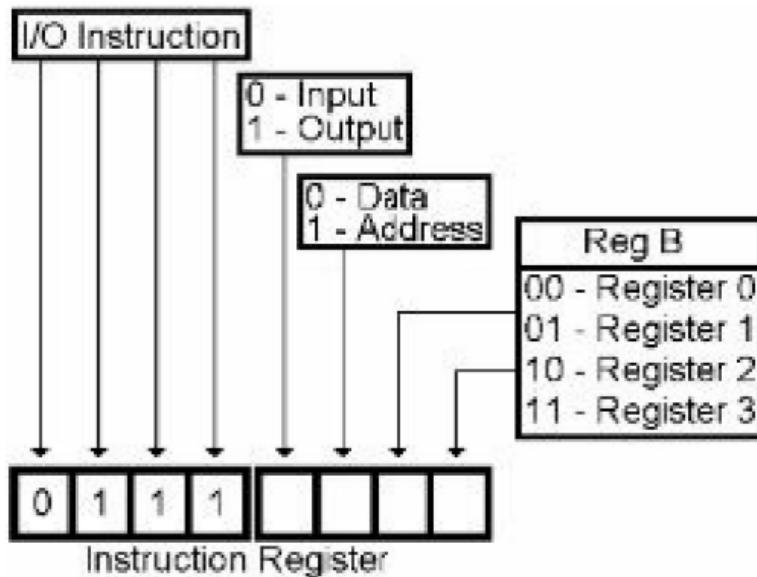
Désolé si c'était déroutant, mais avoir ce diagramme dans le livre une fois suffisait.



Les bits IR 4 et 5 sont placés sur le bus d'E/S à tout moment.

Pour que l'opération d'E/S se produise, une seule étape est nécessaire. Pour la sortie, Reg B est activé et I/O Clk s'est activé et désactivé à l'étape 4. Les étapes 5 et 6 rien. Pour Input, I/O Clk e est activé et Reg B est défini à l'étape 5. Les étapes 4 et 6 ne font rien.

Voici le code d'instruction pour l'instruction I/O :



Cette seule instruction peut être utilisée de quatre manières différentes en fonction des bits IR 4 et 5, et donc il y a quatre nouveaux mots pour notre langue.

Langue	Sens
DANS	Données, RB Données d'E/S d'entrée vers RB
DANS	Adr,RB Adresse d'E/S d'entrée vers RB
DEHORS	Données, RB Sortie RB vers E/S en tant que données
DEHORS	Adr,RB Sortie RB vers E/S comme adresse

Chaque périphérique d'E/S a ses propres caractéristiques uniques, et

nécessite donc des pièces et un câblage uniques pour le connecter au bus d'E/S. L'ensemble des pièces qui connectent l'appareil au bus est appelé « adaptateur d'appareil ». Chaque type d'adaptateur a un nom spécifique tel que « l'adaptateur de clavier » ou « l'adaptateur de disque ». L'adaptateur ne fait rien tant que son adresse n'apparaît pas sur le bus. Lorsque c'est le cas, l'adaptateur répond aux commandes que l'ordinateur lui envoie.

Avec une instruction 'OUT Addr', l'ordinateur active le fil d'adresse et met l'adresse de l'appareil avec lequel il veut parler sur le bus CPU. Le périphérique reconnaît son adresse et prend vie. Tous les autres périphériques ont une autre adresse, ils ne répondront donc pas.

Nous n'allons pas décrire chaque porte du système d'E/S. À ce stade, vous devriez croire que des octets de

les informations peuvent être transférées sur un bus avec quelques

fils de commande. Le message de ce chapitre est uniquement la simplicité du système d'E/S. Le CPU et la RAM sont l'ordinateur. Tout le reste, les disques, les imprimantes, les claviers, la souris, l'écran d'affichage, les choses qui émettent du son, les choses qui se connectent à Internet, toutes ces choses sont des périphériques, et tout ce qu'ils sont capables de faire est d'accepter des octets de données du ordinateur ou envoyer des octets de données à l'ordinateur. Les adaptateurs pour différents périphériques ont des capacités différentes, des nombres de registres différents et des exigences différentes en ce qui concerne ce que le programme exécuté dans le processeur doit faire pour faire fonctionner correctement le périphérique.

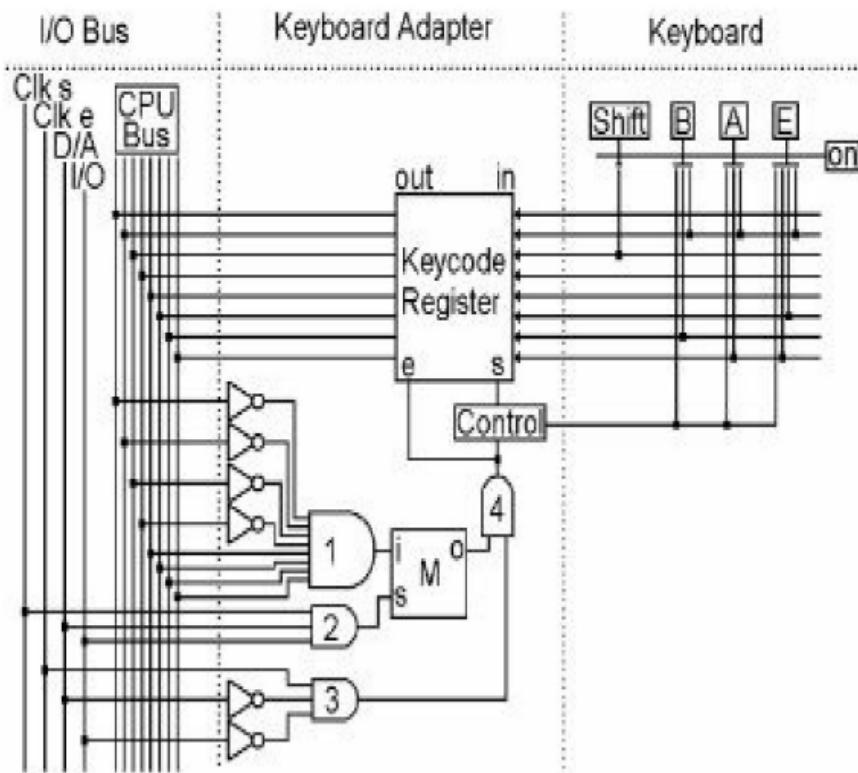
Mais ils ne font rien de plus fantaisiste que ça.

L'ordinateur contrôle le processus avec quelques commandes d'E/S simples qui sont exécutées par le CPU.

Le clavier

Un clavier est l'un des périphériques les plus simples connectés au bus d'E/S. Il s'agit d'un périphérique d'entrée uniquement et ne présente qu'un octet à la fois au processeur.

Le clavier a huit fils à l'intérieur, son propre petit bus comme illustré à droite. Lorsque vous appuyez sur une touche, il connecte simplement l'électricité aux fils nécessaires pour créer le code ASCII correspondant à la touche qui a été enfoncée. Cette petite boîte qui dit "Contrôle" est également notifiée lorsqu'une touche est enfoncée et définit le code ASCII dans le registre des codes clés.



Après avoir appuyé sur une touche, il y aura un code ASCII en attente dans le Keycode Register. Voici comment le CPU obtient ce code dans l'un de ses registres.

La porte ET #1 a huit entrées. Ils sont connectés au bus CPU, quatre d'entre eux via des portes NOT. Ainsi, cette porte ET s'activera à chaque fois que le bus contient 0000 1111.

Il s'agit de l'adresse d'E/S de cet adaptateur de clavier.

La porte ET #2 s'active uniquement pendant le temps 'clk s' d'une instruction OUT Addr. Il actionne l'entrée 'set' d'un bit mémoire. Si le bus contient 0000 1111 à ce moment, l'entrée 'i' sera activée et le bit de mémoire deviendra

sur. Lorsque ce bit de mémoire est activé, cela signifie que le adaptateur de clavier est actif.

La porte ET #3 s'allume pendant le temps 'clk e' d'un IN Data instruction. Si le bit de mémoire est activé, la porte ET #4 s'allume et le Keycode Register sera activé sur le bus, qui sera défini dans Reg B dans le CPU.

Chaque adaptateur connecté au bus d'E/S doit avoir le type de circuit que nous voyons dans les portes #1 et #2 et le bit de mémoire ci-dessus. Chaque adaptateur aura un combinaiion différente qui active la porte #1 ; c'est ce qui permet au CPU de sélectionner chaque adaptateur individuellement.

Voici un petit programme qui déplace la touche actuelle dans Reg 3 dans le CPU.

Instruction		commentaires
Données	R200,000 1111	* Mettez l'adresse du clavier dans Reg
DEHORS	Adresse, R2	* Sélectionnez le clavier
DANS	Données, R3	* Obtenir l'ASCII de la touche enfoncée
LIBRE	R2,R2	* Effacer l'adresse dans Reg 2
DEHORS	Adresse, R2	* Désélectionner le clavier

Cette petite case "Contrôle" efface le registre des codes clés après avoir été envoyé à la CPU.

Le programme en cours d'exécution dans le CPU vérifiera le clavier adaptateur sur une base régulière, et si l'octet qu'il reçoit tous les zéros, aucune touche n'a été enfoncée. Si l'octet a un ou plusieurs bits activés, alors le programme faire tout ce que le programme a été conçu pour faire avec un frappe à ce moment-là.

Encore une fois, nous n'allons pas franchir toutes les portes du Adaptateur clavier. Tous les adaptateurs d'appareils ont le même sortes de circuits afin d'être en mesure de répondre lorsque ils sont adressés, et envoient ou reçoivent des octets de informations au besoin. Mais ce n'est pas plus compliqué que ça. C'est tout ce que font les périphériques d'E/S et les adaptateurs.

Les écrans d'affichage de

télévision et d'ordinateur fonctionnent de la même manière, la principale différence entre eux est uniquement ce qu'ils affichent. Ce n'est pas réellement de la technologie informatique, car vous n'avez pas besoin d'un écran d'affichage pour avoir un ordinateur, mais la plupart des ordinateurs ont un écran, et l'ordinateur passe beaucoup de temps à faire ressembler l'écran à quelque chose, nous devons donc savoir un peu comment ça marche.

La télévision semble vous donner des images animées avec du son. Les images et le son sont faits séparément, et dans ce chapitre, nous ne nous intéressons qu'au fonctionnement de l'image.

La première chose à savoir est que bien que l'image semble bouger, il s'agit en fait d'une série d'images fixes présentées si rapidement que l'œil ne la remarque pas. Vous le saviez probablement déjà, mais voici la prochaine chose. Vous avez vu un film cinématographique. C'est une série de photos. Pour regarder un film, vous placez le film dans un projecteur, qui éclaire une image, puis déplace le film vers l'image suivante, éclaire, etc. Il tourne généralement à 24 images par seconde, ce qui est assez rapide pour donner l'illusion d'une image en mouvement constant.

La télévision va un peu plus vite, environ 30 images par seconde, mais il y a une autre différence, beaucoup plus grande, entre le cinéma et la télévision. Avec le film cinématographique, chaque image fixe est affichée en une seule fois. Chaque image est complète, lorsque vous faites passer la lumière à travers elle, chaque partie de l'image apparaît simultanément sur l'écran. La télévision n'est pas capable de faire cela.

Il n'a pas une image entière à mettre sur l'écran en une seule fois.

Tout ce qu'un téléviseur peut faire à un instant donné, c'est éclairer un seul point sur l'écran. Il allume un point, puis un autre point, puis un autre, très rapidement jusqu'à ce que la valeur d'une image entière de points ait été éclairée.

L'ensemble des points de cet écran constitue une image fixe, il doit donc allumer tous les points en un trentième de seconde, puis tout recommencer

avec l'image suivante, etc. jusqu'à ce qu'il ait placé 30 points sur l'écran en une seconde. Ainsi, le téléviseur est très occupé à éclairer des points individuels, 30 fois le nombre de points sur l'écran, chaque seconde.

Habituellement, le point supérieur gauche est allumé en premier, puis celui à sa droite, et ainsi de suite en haut de l'écran jusqu'au coin supérieur droit. Ensuite, il commence par la deuxième ligne de points, traversant à nouveau l'écran, la troisième ligne, etc. jusqu'à ce qu'il ait balayé tout l'écran. La luminosité de chaque point est élevée ou faible afin que chaque partie de l'écran soit éclairée à la luminosité appropriée pour que l'écran ressemble à l'image souhaitée.

À tout instant, le téléviseur n'a affaire qu'à un seul point solitaire sur l'écran. Donc, avec la télévision, il y a deux illusions - l'illusion de mouvement provenant d'une série d'images fixes, ainsi que l'illusion d'images fixes complètes qui sont en fait dessinées un point à la fois. Cette deuxième illusion est aidée par la composition de l'écran, chaque point ne s'allume que pendant une infime fraction de seconde, et il commence à s'estomper immédiatement. Heureusement, quelle que soit la composition de l'écran qui brille, continue de briller dans une certaine mesure entre le moment où le point est allumé et le 1/30 à nouveau.

d'une seconde plus tard lorsque ce même point s'allume

Pour les yeux, vous ne voyez qu'une image animée, mais il se passe beaucoup de choses pour la faire apparaître de cette façon.

Dans un ordinateur, un seul point sur l'écran est appelé un "élément d'image" ou "pixel" en abrégé.

Les écrans d'ordinateur fonctionnent exactement comme les téléviseurs. Ils doivent également balayer l'écran entier 30 fois par seconde pour éclairer chaque pixel individuel et ainsi faire apparaître une image. Même si le contenu de l'écran ne change pas, quelque chose dans l'ordinateur doit scanner cette image immuable sur l'écran 30 fois par seconde.

Pas de numérisation, pas d'image - c'est comme ça que ça marche.

Nous n'allons pas entrer dans la même quantité de détails ici que nous avons fait avec le CPU et la RAM, ces deux sont ce qui en fait un ordinateur, mais si nous voulons savoir comment notre ordinateur est capable de mettre quelque chose sur l'écran que nous pouvons lire, nous devons avoir l'idée de base de la façon dont il

œuvres.

Dans ce chapitre, nous examinerons le type d'écran le plus simple, celui qui est en noir et blanc, et dont les pixels ne peuvent être que complètement allumés ou complètement éteints. Ce type d'écran peut afficher des caractères et le type d'images qui sont faites de dessins au trait. Plus loin dans le livre, nous verrons les quelques changements simples qui permettent à un écran d'afficher des choses comme des photographies en couleur.

Les parties principales sont au nombre de trois. Il y a d'abord l'ordinateur, on a vu comment ça marche. Il dispose d'un bus d'E / S qui peut déplacer des octets vers et depuis des éléments extérieurs à l'ordinateur.

Deuxièmement, l'écran. L'écran est juste une grande grille de pixels, dont chacun peut être sélectionné, un à la fois, et lorsqu'il est sélectionné, peut être activé ou non. Le troisième élément est "l'adaptateur d'affichage". L'adaptateur d'affichage est connecté au bus d'E/S d'un côté et à l'écran de l'autre côté.

Le cœur d'un adaptateur d'affichage est de la RAM. L'adaptateur d'affichage a besoin de sa propre RAM pour pouvoir "se souvenir" des pixels qui doivent être allumés et des pixels qui doivent être éteints. Dans le type d'écran que nous allons décrire ici, il doit y avoir un bit en RAM pour chaque pixel à l'écran.

Afin de faire en sorte que l'écran analyse chaque pixel 30 fois par seconde, l'adaptateur d'affichage a besoin de sa propre horloge qui tourne à une vitesse 30 fois supérieure au nombre de pixels à l'écran. A chaque tick de l'horloge, un pixel est sélectionné et il est activé ou non par le bit correspondant de la RAM.

Par exemple, utilisons un ancien type d'écran. Il s'agit d'un écran noir et blanc qui affiche 320 pixels en travers de l'écran et 200 pixels en bas. Cela revient à 64 000 pixels individuels sur l'écran. Chaque pixel sur l'écran a une adresse unique composée de deux nombres, le premier étant la position gauche-droite ou horizontale, et l'autre étant la position haut-bas ou verticale.

L'adresse du pixel supérieur gauche est 0,0 et celle du pixel inférieur droit est 319 199. 64 000 pixels fois 30 images par seconde signifie que l'horloge de cet adaptateur d'affichage doit cocher 1 920 000 fois par seconde. Et comme il y a huit bits dans un octet, nous aurons besoin de 8 000 octets de RAM d'affichage pour indiquer à chacun des 64 000 pixels de l'écran s'il doit être allumé ou

à l'arrêt.

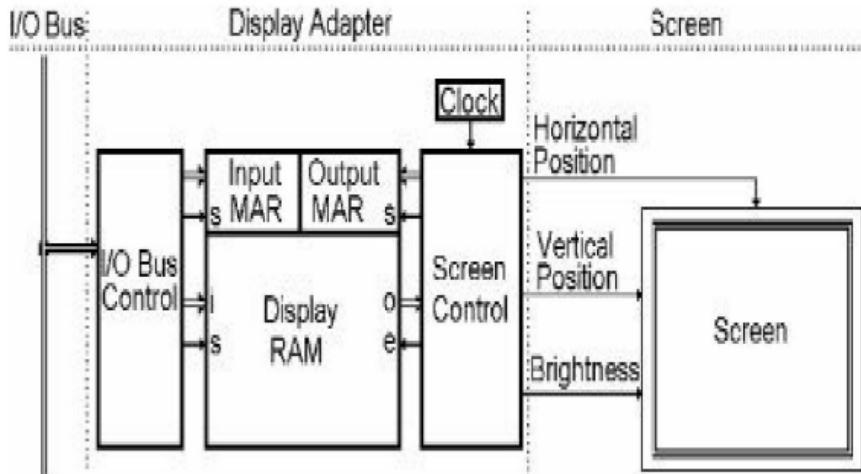
L'adaptateur d'affichage a un registre qui définit la position horizontale du pixel actuel. L'adaptateur d'affichage ajoute 1 à ce registre à chaque tick de l'horloge. Il commence à zéro et lorsque le nombre qu'il contient atteint 319, l'étape suivante le remet à zéro. Il passe donc de zéro à 319 encore et encore. Il existe également un registre qui définit la position verticale du pixel actuel. Chaque fois que le registre horizontal est remis à zéro, l'adaptateur d'affichage ajoute 1 au registre vertical. Lorsque le registre vertical atteint 199, l'étape suivante le remettra à zéro. Ainsi, comme le registre horizontal passe de zéro à 319 200 fois, le registre vertical passe de zéro à 199 une fois.

Le pixel d'écran actuellement sélectionné est contrôlé par ces registres, de sorte que lorsque le registre horizontal passe de 0 à 319, le pixel actuel traverse l'écran une fois.

Ensuite, le registre vertical en a un ajouté et le pixel actuel descend au premier pixel de la ligne suivante.

Ainsi, l'horloge et les registres horizontaux et verticaux sélectionnent chaque pixel sur l'écran, un à la fois, en allant de gauche à droite dans une rangée, puis en sélectionnant chaque pixel dans la rangée suivante vers le bas, puis le suivant, etc. jusqu'à ce que chaque pixel sur l'écran a été sélectionné une fois. Puis ça recommence encore.

En même temps, il y a un autre registre qui contient une adresse RAM d'affichage. Ce registre est également parcouru, bien que nous n'ayons besoin que d'un nouvel octet tous les huit pixels. Les bits de chaque octet, un à la fois, sont envoyés à l'écran à huit pixels consécutifs pour les activer ou les désactiver. Tous les huit pixels, le registre d'adresse de la RAM a 1 ajouté. Au moment où tous les pixels ont été parcourus, toute la RAM a également été parcourue et une image entière a été dessinée. Lorsque les registres horizontal et vertical ont tous deux atteint leur maximum et sont remis à zéro, l'adresse RAM est également remise à zéro.



L'adaptateur d'affichage passe la plupart de son temps à peindre l'écran. La seule autre chose qu'il doit faire est d'accepter les commandes du bus d'E/S qui modifieront le contenu de la RAM de l'adaptateur d'affichage. Lorsque le programme en cours d'exécution dans le processeur doit modifier ce qui est à l'écran, il utilise la commande I/O OUT pour sélectionner la carte graphique, puis envoie une adresse RAM de carte graphique, puis un octet de données à stocker à cette adresse. Alors que l'adaptateur continue à repeindre l'écran, les nouvelles données apparaîtront sur l'écran à l'endroit approprié.

La RAM de l'adaptateur d'affichage est construite différemment de la RAM de notre ordinateur. Il maintient les fonctions d'entrée et de sortie séparées. Les entrées de tous les emplacements de stockage sont connectées au bus d'entrée, et les sorties de tous les emplacements de stockage sont connectées au bus de sortie, mais le bus d'entrée et le bus de sortie sont maintenus séparés. Ensuite, il y a deux registres d'adresses mémoire séparés, un pour l'entrée et un pour la sortie. L'entrée MAR a une grille qui sélectionne uniquement quel octet sera "défini", et la sortie MAR a une grille séparée qui sélectionne uniquement quel octet sera "activé".

Avec cette configuration, l'écran et la RAM d'affichage peuvent tous deux être scannés en continu en utilisant uniquement la sortie MAR et

le bit d'activation. Lorsque le bus d'E/S est utilisé pour écrire dans la RAM d'affichage, il utilise uniquement l'entrée MAR et le bit défini.

C'est ainsi que l'adaptateur d'affichage crée une image sur l'écran. En raison de la façon dont cela fonctionne, il existe une relation intéressante entre les bits de la RAM d'affichage qui correspondent aux pixels de l'écran. Lorsqu'il scanne les huit premiers pixels de la ligne du haut, il utilise les bits individuels de l'octet 0 de sa RAM pour activer ou désactiver les pixels. Lorsqu'il scanne les huit pixels suivants, il utilise les bits individuels de l'octet 1 de sa RAM, etc.

Il faut 40 octets de RAM pour dessiner la première ligne, et donc les huit derniers pixels, qui sont numérotés de 312 à 319, proviennent de l'octet 39 de la RAM. La deuxième ligne utilise l'octet 40 pour dessiner ses 8 premiers pixels, etc.

Si vous voulez écrire des lettres et des chiffres sur l'écran, comment faites-vous ? Si vous mettez le code ASCII pour 'A' dans un octet de la RAM d'affichage, vous n'obtiendrez que huit pixels d'affilée où un est éteint, puis un est allumé, puis cinq sont éteints et le dernier est allumé. Ce n'est pas ce à quoi un 'A' devrait ressembler.

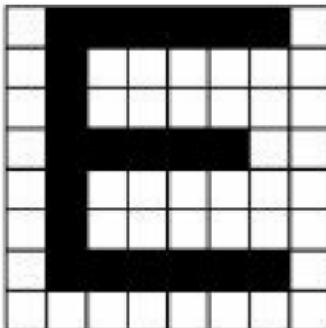
Il existe une solution pour cela, et cela implique...

Un autre code

Lorsque vous souhaitez imprimer ou afficher un langage écrit, vous devez traduire le code ASCII en quelque chose qui soit lisible par une personne vivante. Nous avons un code, 0100 0101, qui apparaît sur la table des codes ASCII à côté de la lettre « E ». Mais comment l'ordinateur transforme-t-il 0100 0101 en un « E » lisible ?

Nous avons un écran d'affichage, mais l'écran n'est qu'une grille de pixels, il n'y a pas de 'E' lisibles par l'homme dans tout ce que nous avons décrit jusqu'à présent. Pour obtenir un 'E' sur l'écran, il doit y avoir quelque chose qui crée cette forme que nous reconnaissons comme une lettre de l'alphabet.

Par conséquent, nous avons besoin d'un autre code. Ce code concerne vraiment de petites images faites de points. Pour chaque personnage que nous voulons pouvoir dessiner à l'écran, nous avons besoin d'une petite image de ce personnage. Si vous prenez une grille de 8 pixels de large et 8 pixels de haut, vous pouvez décider quels pixels doivent être activés pour créer une petite image qui ressemble au personnage que vous souhaitez dessiner à l'écran, comme ceci :



Si vous transformez cette image en ons et offs, vous pouvez la stocker sur huit octets. S'il y a 100 caractères différents que vous voulez pouvoir afficher à l'écran, alors vous auriez besoin de 100 petites images différentes comme celle-ci, et il faudrait 800 octets de RAM pour les stocker. Notre petit ordinateur n'a qu'une RAM de 256 octets,

ce serait donc le bon moment pour imaginer cette version plus grande que nous avons décrite précédemment.

Ces 800 octets sont un type de code connu sous le nom de "police".

Si vous voulez faire apparaître un caractère à un certain endroit sur l'écran, vous devez choisir la bonne petite image dans la police, puis utiliser les instructions d'E/S pour copier les huit octets de l'image dans les octets appropriés de l'affichage. RAM de l'adaptateur.

Si les images de notre police sont disposées dans le même ordre que la table de codes ASCII, nous pouvons utiliser la valeur numérique d'un code ASCII pour trouver l'image correspondante dans la police. Le code ASCII pour 'E' est 0100 0101. Si vous appliquez le code binaire au même modèle de uns et de zéros, vous obtenez le nombre décimal 69. 'E' est alors le code 69 en ASCII, et l'image e d'un 'E' sera l'image 69 dans la police.

e

Puisqu'il y a huit octets dans chaque image, vous multipliez le 69 par 8, et cela vous indique que l'image pour 'E' sera les huit octets commençant à l'adresse 552.

Nous devons maintenant savoir où copier ces octets dans la RAM d'affichage. Disons que nous voulons afficher un 'E' tout en haut à gauche de l'écran. Où sont les bits qui allument les pixels qui nous intéressent ? Eh bien, la première ligne est simple, ce sont les huit premiers bits de la RAM d'affichage, l'adresse 0. Nous utilisons donc une série d'instructions OUT pour copier l'adresse RAM 552 pour afficher l'adresse RAM 0. Maintenant, où est la deuxième ligne dans le afficher la RAM ? L'affichage peint les 320 bits de la rangée supérieure avant de descendre à la deuxième rangée. Cela signifie qu'il utilise 40 octets sur chaque ligne, donc la ligne du haut utilise les octets 0-39. Cela signifie que le deuxième octet de l'image de 'E' à l'adresse RAM 553 doit être écrit à l'adresse 40 dans la RAM d'affichage. De même, les troisième à huitième octets sont écrits aux octets 80, 120, 160, 200, 240 et 280. Lorsque vous avez fait tout cela, vous verrez alors un « E » complet à l'écran. Si vous vouliez écrire un 'X' sur l'écran juste à côté du 'E', vous localiserez les huit octets dans la police pour 'X' et les copieriez dans les octets 1, 41, 81, 121, 161 de la RAM d'affichage, 201, 241 et 281. Si vous avez besoin de 27 'E's sur votre écran, il vous suffit de copier le 'E' dans votre police à 27 endroits différents dans le

afficher la RAM.

RAM Address	Byte Contents	Display RAM Address	Screen Pixels	Screen
552	01111110	000		
553	01000000	040		E
554	01000000	080		
555	01111100	120		
556	01000000	160		
557	01000000	200		
558	01111110	240		
559	00000000	280		

Bien sûr, cela semble être beaucoup de travail juste pour faire apparaître une seule lettre à l'écran. Le programme qui fait cela aurait besoin d'une boucle d'instructions qui calcule les premières adresses «de» et «à», puis émet les instructions OUT appropriées pour copier le premier octet dans la RAM d'affichage. Ensuite, la boucle se répétait, mettant à jour les deux adresses à chaque fois, jusqu'à ce que les huit octets aient été copiés aux endroits appropriés.

Nous n'allons pas écrire ce programme, mais il pourrait facilement s'agir d'un programme de 50 instructions qui doit boucler huit fois avant d'être terminé. Cela signifie que cela pourrait prendre 400 cycles d'instruction juste pour mettre un caractère à l'écran ! Si vous dessinez 1 000 caractères à l'écran, cela peut prendre 400 000 cycles d'instructions.

D'un autre côté, cela ne représente encore qu'environ un quart de un pour cent de ce que cet ordinateur peut faire en une seconde.

Cela vous montre simplement pourquoi les ordinateurs doivent être si rapides. Les choses individuelles qu'ils font sont si petites qu'il faut un grand nombre d'étapes pour obtenir quoi que ce soit fait du tout.

Le dernier mot sur les codes

Nous avons vu plusieurs codes utilisés dans notre ordinateur. Chacun a été conçu dans un but précis. Les messages codés individuels sont mis en octets, déplacés et utilisés pour faire avancer les choses.

Les octets ne 'savent' pas quel code a été utilisé pour choisir le motif qu'ils contiennent. Il n'y a rien dans l'octet lui-même qui vous indique quel code il est censé être.

Certaines parties de l'ordinateur sont construites avec différents codes à l'esprit. Dans l'ALU, l'additionneur et le comparateur sont conçus pour traiter les octets comme s'ils contenaient des valeurs codées avec le code numérique binaire. Il en va de même pour le registre d'adresses mémoire et le registre d'adresses d'instructions.

Le registre d'instructions est construit pour traiter son contenu comme s'il contenait des valeurs codées avec le code d'instruction.

Les bits de RAM de l'adaptateur d'affichage ne sont que des activations ou des désactivations pour des pixels individuels. Les images et les polices sont des chaînes d'octets qui se traduiront par quelque chose qui peut être reconnu par une personne lorsqu'il est organisé, et les luminosités sont définies, par le câblage d'un adaptateur d'affichage et d'un écran.

La table de codes ASCII n'apparaît nulle part dans l'ordinateur car il n'y a aucun moyen de représenter une lettre de l'alphabet sauf en utilisant un code.

Les seuls endroits où l'ASCII est converti entre les caractères et le code du caractère sont dans les périphériques. Lorsque vous appuyez sur 'E' sur le clavier, vous obtenez le code ASCII pour un 'E.' Lorsque vous envoyez le code ASCII d'un « E » à une imprimante, celle-ci imprime la lettre « E ». Les personnes qui construisent ces périphériques ont une table de codes ASCII devant eux, et lorsqu'ils construisent un clavier, le commutateur sous le quatrième bouton de la deuxième rangée, sur lequel est imprimée la lettre « E », est câblé au fils de bus appropriés pour produire le code qui apparaît

à côté de la lettre 'E' sur la table des codes ASCII.

Un 'E' est la cinquième lettre d'un alphabet utilisé par les gens pour représenter les sons et les mots en cours d'écriture

vers le bas de leur langue parlée. Les seuls 'E' dans l'ordinateur sont celui du clavier et ceux qui apparaissent à l'écran. Tous les « E » qui sont en octets ne sont que le code qui apparaît à côté du « E » sur une table de codes ASCII. Ce ne sont pas des 'E', il n'y a aucun moyen de mettre un 'E' dans un ordinateur. Même si vous mettez une image d'un 'E' dans un ordinateur, ce n'est pas réellement un 'E' tant qu'il n'est pas affiché à l'écran. C'est à ce moment-là qu'une personne peut le regarder et dire "C'est un E".

Les octets sont muets. Ils contiennent juste des modèles de ons et offs. Si un octet contient 0100 0101 et que vous l'envoyez à l'imprimante, elle imprimerá la lettre « E ». Si vous l'envoyez au registre d'instructions, l'ordinateur exécutera une instruction de saut. Si vous l'envoyez au registre d'adresses mémoire, il sélectionnera l'octet numéro 69 de la RAM. Si vous l'envoyez d'un côté de l'additionneur, il ajoutera 69 à tout ce qui se trouve de l'autre côté de l'additionneur. Si vous l'envoyez à l'écran d'affichage, il activera trois pixels et désactivera cinq pixels.

Chacune de ces pièces de l'ordinateur est conçue avec un code à l'esprit, mais une fois qu'il est construit, l'esprit est parti et même le code est parti. Il fait juste ce pour quoi il a été conçu.

Il n'y a pas de limite aux codes qui peuvent être inventés et utilisé dans un ordinateur. Les programmeurs inventent constamment de nouveaux codes. Comme la caisse enregistreuse du fast-food mentionné plus tôt, quelque part dans cette machine se trouve un mot qui signifie "inclure des frites".

Le disque

La plupart des ordinateurs ont un disque. Il s'agit simplement d'un autre périphérique connecté au bus d'E/S. La mission du disque est très simple ; il peut faire deux choses. Vous pouvez lui envoyer des octets, qu'il stockera, ou vous pouvez lui dire de renvoyer certains octets, qui ont été stockés précédemment.

Il y a deux raisons pour lesquelles la plupart des ordinateurs ont un disque. Premièrement, ils ont la capacité de stocker un nombre énorme d'octets, plusieurs fois supérieur à la RAM de l'ordinateur. Le CPU ne peut exécuter que des programmes qui sont en RAM, il ne peut manipuler que des octets qui sont en RAM. Mais il n'y a jamais assez de RAM pour stocker toutes les choses que vous pourriez vouloir faire avec votre ordinateur. Et donc un disque contiendra tout, et quand vous voulez faire une chose, les octets sur le disque pour cette chose seront copiés dans la RAM et utilisés. Ensuite, lorsque vous voulez faire quelque chose de différent, les octets de la nouvelle activité seront copiés du disque dans la même zone de RAM qui avait été utilisée pour la première activité.

La deuxième raison pour laquelle les ordinateurs ont des disques est que les octets stockés sur le disque ne disparaissent pas lorsque vous éteignez l'alimentation. La RAM perd ses paramètres lorsque vous éteignez l'ordinateur, lorsque vous le rallumez, tous les octets sont 0000 0000, mais le disque conserve tout ce qui a été écrit dessus.

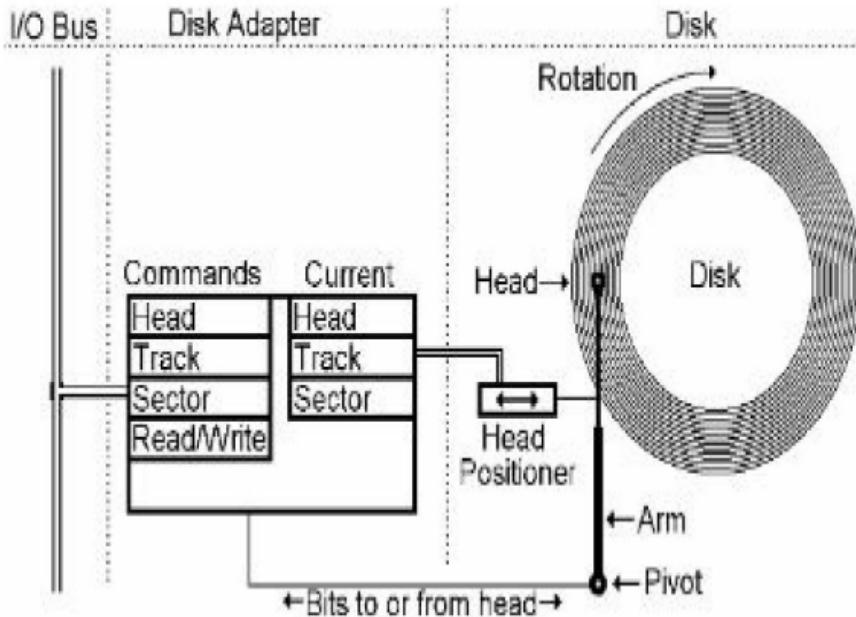
Un bit d'ordinateur a été défini jusqu'à présent comme un endroit où il y a ou n'y a pas d'électricité. Mais avant cela, nous l'avons défini comme un endroit qui peut être dans l'un des deux états différents. Sur un disque, les bits électriques sont transformés en des endroits sur la surface du disque qui ont été magnétisés d'une manière ou d'une autre. Puisque les aimants ont des pôles nord et sud, la tache sur le disque peut être magnétisée nord-sud ou sud-nord. Une direction représenterait un zéro, et l'autre direction, un un. Une fois qu'un spot est magnétisé, il le reste à moins que le même spot ne soit magnétisé dans l'autre sens. La mise hors tension n'a aucun effet sur les points magnétisés.

Un disque, comme son nom l'indique, est une chose ronde, qui tourne rapidement. Il est recouvert d'un matériau qui

peut être magnétisé facilement. Vous souvenez-vous du télégraphe ?

À l'extrême de réception, il y a un morceau de métal avec un fil enroulé autour de lui. Ce morceau de métal se transforme en un aimant lorsque l'électricité se déplace à travers le fil. Le disque en a une petite version appelée « tête » montée sur un bras. Le bras tient la tête très près de la surface du disque en rotation, et le bras peut se balancer d'avant en arrière, de sorte que la tête puisse atteindre n'importe quel point de la surface du disque. Si vous mettez de l'électricité dans la tête, cela peut magnétiser la surface du disque. De plus, cela fonctionne dans l'autre sens; lorsque la tête passe au-dessus d'une zone aimantée, elle fait apparaître de l'électricité dans les fils enroulés autour de la tête. Ainsi, la tête peut soit écrire sur le disque, soit lire ce qui a été précédemment écrit sur le disque. Les bits des octets sont écrits les uns après les autres sur la surface du disque.

La surface du disque est divisée en une série d'anneaux, appelés pistes, très proches les uns des autres. La tête peut se déplacer sur la surface et s'arrêter sur n'importe laquelle des pistes. Chaque piste circulaire est généralement divisée en petits morceaux appelés secteurs. Puisqu'un disque a deux côtés, les deux côtés sont généralement recouverts du matériau magnétique et il y a une tête de chaque côté.



Dans la RAM, chaque octet a sa propre adresse. Sur un disque, il existe aussi un moyen de localiser les octets, mais c'est très différent.

Vous devez spécifier quelle tête, quelle piste et quel secteur où se trouve un bloc d'octets. C'est le type "d'adresse" que les données sur un disque ont, comme "Head 0, Track 57, Sector 15". Et à cette adresse, il n'y a pas qu'un seul octet, mais un bloc d'octets, typiquement plusieurs milliers. Pour les exemples de notre livre, puisque notre RAM est si petite, nous parlerons d'un disque qui stocke des blocs de 100 octets.

Lorsqu'un disque est lu ou écrit, il n'y a aucun moyen d'accéder à un octet individuel dans le bloc d'octets. Le bloc entier doit être transféré dans la RAM, travaillé dans la RAM, puis le bloc entier doit être réécrit sur le disque.

Le disque tourne rapidement, plus vite que ce ventilateur sur votre bureau ; de nombreux disques populaires tournent 7200 fois par minute, ce qui

est de 120 fois par seconde. C'est assez rapide, mais comparé au CPU, c'est encore assez lent. Pendant que le disque tourne environ une fois, l'horloge fera tic tac plus de huit millions de fois et notre processeur exécutera bien plus d'un million d'instructions.

Le disque, comme tout périphérique, est connecté à son propre adaptateur, qui à son tour est connecté au bus d'E/S. L'adaptateur de disque fait quelques choses. Il accepte les commandes pour sélectionner une tête, sélectionner une piste et sélectionner un secteur. Il accepte les commandes pour lire ou écrire dans le bloc d'octets à la tête, à la piste et au secteur actuellement sélectionnés.

Il y aura aussi probablement une commande où le CPU peut vérifier la position actuelle du bras et du disque.

La commande de sélection d'une tête peut être exécutée immédiatement, mais lorsqu'il reçoit une commande pour sélectionner une piste, il doit déplacer la tête vers cette piste, ce qui prend beaucoup de temps en termes de cycles d'instructions. Lorsqu'il reçoit une commande pour sélectionner un secteur, il doit attendre que ce secteur tourne jusqu'à l'endroit où se trouve la tête, ce qui prend également beaucoup de temps en termes de cycles d'instructions.

Lorsque le CPU a déterminé que la tête est arrivée à la piste et au secteur souhaités, les commandes d'E/S pour la lecture ou l'écriture peuvent être exécutées, et un octet à la fois sera transféré sur le bus d'E/S. Un programme qui lit ou écrit un bloc d'octets doit continuer le processus jusqu'à ce que tout le bloc d'octets soit terminé.

Avec notre système d'E/S simple, les octets individuels se déplacent entre le disque et un registre CPU. Le programme en cours d'exécution doit déplacer ces octets vers ou depuis la RAM, généralement dans des emplacements consécutifs.

C'est tout ce qu'un disque fait. Vous avez probablement utilisé un ordinateur qui avait un disque et n'aviez pas besoin de savoir quoi que ce soit sur les têtes, les pistes et les secteurs. Et c'est une bonne chose, car il est assez ennuyeux d'avoir affaire à un disque à ce niveau de détail. Nous verrons comment un disque est normalement utilisé plus tard dans le livre.

Autre note de langue : plusieurs mots signifient pratiquement la même chose, mais pour une raison quelconque, certains mots sont associés à certaines technologies.

Si vous voulez envoyer une lettre à quelqu'un, vous l'écrivez d'abord sur un morceau de papier, puis lorsque le destinataire reçoit la lettre, il la lit.

À l'époque des magnétophones, vous commençiez avec une bande vierge. Ensuite, vous enregistriez de la musique sur la bande. Quand vous vouliez réentendre la musique, vous mettiez la bande.

Lorsqu'il s'agit de disques informatiques, placer quelque chose sur le disque s'appelle écrire. Extraire quelque chose du disque s'appelle lire.

Mettre quelque chose dans la RAM s'appelle écrire ou stocker.
Obtenir quelque chose de la RAM s'appelle lire ou récupérer.

Mettre quelque chose dans un registre CPU est généralement appelé chargement.

Mettre de la musique sur un disque s'appelle parfois enregistrer, parfois graver. L'écoute d'un disque est encore généralement appelée lecture, mais si vous le copiez sur votre ordinateur, cela s'appelle l'extraction.

Écrire, enregistrer, stocker, charger et graver signifient à peu près la même chose. Lire, récupérer, jouer et extraire sont également très similaires. Ils veulent dire les mêmes choses, c'est juste une différence de mots.

Excusez-moi Madame

Il y a une autre chose que la plupart des ordinateurs ont dans leur système d'entrée/sortie. Un ordinateur n'a pas besoin de l'un d'entre eux pour être appelé un ordinateur, nous n'allons donc pas passer par toutes les portes nécessaires pour le construire. Mais c'est une chose très courante, nous allons donc décrire comment cela fonctionne.

Vous savez, si maman est dans la cuisine en train de remuer une marmite de soupe, et que le petit Joey entre en courant et dit "Je veux un verre de lait", maman pose la cuillère, va au placard, prend un verre, va à le réfrigérateur, versez le lait, donnez-le à Joey, puis elle retournera à la cuisinière, ramassera la cuillère et recommencera à remuer la soupe. L'agitation de la soupe a été interrompue en obtenant un verre de lait, puis l'agitation de la soupe a repris.

Cette chose que la plupart des ordinateurs ont, s'appelle une «interruption», et cela fonctionne très bien comme ce qui s'est passé avec maman et Joey.

Une interruption commence par un fil supplémentaire ajouté au bus d'E/S. Ce fil est utilisé par certains adaptateurs de périphériques pour faire savoir au CPU que c'est le bon moment pour le CPU d'effectuer une opération d'E/S, comme juste après que quelqu'un appuie sur une touche du clavier. Lorsqu'un adaptateur de périphérique active le bit d'interruption, la prochaine fois que le moteur pas à pas revient à l'étape 1, le cycle d'instruction suivant ne fera pas l'extraction et l'exécution habituelles, mais fera plutôt ce qui suit :

Étape 1

	déplacer le binaire 0 vers MAR
Étape 2	déplacer IAR vers RAM
Étape 3	déplacer le binaire 1 vers MAR
Étape 4	déplacer les drapeaux vers la RAM
Étape 5	déplacer le binaire 2 vers MAR
Étape 6	déplacer la RAM vers IAR
Étape 7	déplacer le binaire 3 vers MAR
Étape 8	déplacer la RAM vers les drapeaux

Le résultat de cette séquence est que l'IAR et les drapeaux actuels sont enregistrés dans les adresses RAM 0 et 1, et ils sont remplacés par le contenu des adresses d'octets RAM 2 et 3. Ensuite, le CPU revient à son extraction et exécution normales.

opération. Mais l'IAR a été remplacé ! Ainsi, la prochaine instruction sera récupérée à partir de n'importe quelle adresse dans l'octet 2 de la RAM.

En d'autres termes, ce que le CPU avait fait est enregistré et le CPU est envoyé pour faire autre chose. Si à la fin de cette nouvelle activité, le programme remet les octets 0 et 1 de la RAM dans l'IAR et les Flags, le CPU reprendra exactement là où il s'était arrêté, avant qu'il ne soit interrompu.

Ce système est très utile pour traiter les opérations d'E/S. Sans interruptions, le programme s'exécutant dans le CPU devrait s'assurer de vérifier régulièrement tous les périphériques sur le bus d'E/S. Avec les interruptions, le programme peut simplement faire ce pour quoi il est conçu, et le programme qui traite des choses comme la saisie au clavier sera appelé automatiquement selon les besoins par le système d'interruption.

Nous ne l'avons pas inclus dans notre processeur car cela rendrait simplement notre schéma de câblage de la section de contrôle trop grand. Il faudrait ajouter ce qui suit : deux étapes supplémentaires au moteur pas à pas, le câblage pour effectuer les 8 étapes ci-dessus à la place du cycle d'instructions normal, les chemins du registre Flags pour aller et venir du bus, une méthode d'envoi d'un 0 binaire , 1, 2 ou 3 à MAR, et une instruction qui restaure les octets 0 et 1 de la RAM dans le registre IAR et Flags.

Et c'est un système d'interruption. En ce qui concerne le langage, les concepteurs de l'ordinateur ont pris un verbe existant, « interrompre », et l'ont utilisé de trois manières : c'est un verbe dans « le clavier a interrompu le programme », c'est un adjectif dans « Ceci est l'interruption ». système », et c'est un nom dans « le CPU a exécuté une interruption ».

C'est tout les gens

Oui, c'est la fin de notre description d'un ordinateur.

C'est tout ce qu'il y a. Tout ce que vous voyez un ordinateur faire est une longue concaténation de ces opérations très simples, les opérations ADDing, NOTting, Shifting, ANDing, ORing, XORing d'octets, Stocking, Loading, Jumping et I/O, via l'exécution du code d'instruction de RAM. C'est ce qui fait d'un ordinateur un ordinateur. C'est la somme totale de l'intelligence d'un ordinateur. C'est tout ce dont un ordinateur est capable. C'est une machine qui fait exactement ce pour quoi elle est conçue, et rien de plus. Comme un marteau, c'est un outil conçu par l'homme pour accomplir des tâches définies par l'homme. Il fait sa tâche exactement comme prévu. Aussi comme un marteau, s'il est lancé sans discernement, il peut faire quelque chose d'imprévisible et de destructeur.

La variété de choses que l'ordinateur peut faire n'est limitée que par l'imagination et l'intelligence des personnes qui créent les programmes qu'ils doivent exécuter. Les personnes qui construisent les ordinateurs continuent de les rendre plus rapides, plus petits, moins chers et plus fiables.

Lorsque nous pensons à un ordinateur, nous pensons probablement à cette boîte qui se trouve sur un bureau et à laquelle sont attachés un clavier, une souris, un écran et une imprimante. Mais les ordinateurs sont utilisés dans de nombreux endroits. Il y a un ordinateur dans votre voiture qui contrôle le moteur. Il y a un ordinateur dans votre téléphone portable. Il y a un ordinateur dans la plupart des boîtiers de télévision par câble ou par satellite. Les choses qu'ils ont tous en commun sont qu'ils ont tous un processeur et une RAM. Les différences sont toutes dans les périphériques. Un téléphone portable a un petit clavier et un petit écran, un microphone et un haut-parleur, et une radio bidirectionnelle pour les périphériques. Votre voiture possède divers capteurs et commandes sur le moteur, ainsi que les cadres du tableau de bord pour les périphériques. La caisse enregistreuse d'un fast-food a un clavier amusant, un petit écran d'affichage et une petite imprimante pour les reçus.

Il y a des ordinateurs dans certains feux de circulation qui changent les lumières en fonction de l'heure de la journée et de la quantité de trafic qui traverse les capteurs intégrés à la chaussée. Mais le CPU et la RAM en font un ordinateur, les périphériques peuvent être très différents.

Pour le reste du livre, nous examinerons divers sujets liés à la compréhension de l'utilisation des ordinateurs, quelques mots intéressants liés aux ordinateurs, certaines de leurs faiblesses et quelques autres détails.

Matériel et logiciel

Vous avez entendu parler de matériel. Ce mot existe depuis longtemps. Il y a des quincailleries depuis un siècle ou plus. Je pense qu'une quincaillerie vendait à l'origine des choses qui étaient dures, comme des casseroles et des poêles, des tournevis, des pelles, des marteaux, des clous, des charrues, etc.

Peut-être que le « matériel » signifiait des choses qui étaient faites de métal. Aujourd'hui, certaines quincailleries ne vendent plus de casseroles et de poêles, mais elles vendent une grande variété de choses dures, comme des boulons et des tondeuses à gazon, ainsi que du bois et beaucoup de choses douces aussi, comme des tapis, du papier peint, de la peinture, etc. Mais ces choses douces sont pas appelé logiciel.

Le mot « logiciel » a été inventé quelque part dans les premiers jours de l'industrie informatique pour différencier l'ordinateur lui-même de l'état des bits qu'il contient.

Le logiciel désigne la façon dont les bits sont activés ou désactivés dans un ordinateur, par opposition à l'ordinateur lui-même. N'oubliez pas que les bits peuvent être activés ou désactivés. Le mors a une place dans l'espace, il est fait de quelque chose, il existe dans l'espace, on le voit. Le bit est matériel. Que le bit soit allumé ou éteint est important, mais ce n'est pas une pièce séparée que vous vissez dans l'ordinateur, c'est la chose dans l'ordinateur qui est modifiable, la chose qui peut être moulée, c'est "doux" en ce sens qu'il peut changer, mais vous ne pouvez pas le prendre tout seul dans votre main. Cette chose s'appelle un logiciel.

Pensez à une cassette vidéo vierge. Ensuite, enregistrez un film dessus. Quelle est la différence entre la cassette vidéo vierge et la même cassette vidéo avec un film dessus ? Il a le même aspect, il pèse le même, vous ne pouvez pas voir de différence sur la surface de la bande. Cette surface est recouverte de particules très fines qui peuvent être magnétisées. Dans la bande vierge, toute la surface de la bande est magnétisée dans des directions aléatoires. Après avoir enregistré le film sur la bande, certains petits endroits sur la bande sont magnétisés dans un sens et d'autres petits endroits sont magnétisés dans l'autre sens. Rien n'est ajouté ou retiré de la bande, c'est juste la façon dont les particules magnétiques sont magnétisées. Lorsque vous insérez la cassette dans un magnétoscope, il lit un film. La bande est matérielle, le modèle des directions de magnétisation sur la bande est logiciel.

Dans un ordinateur, il y a un très grand nombre de bits. Comme nous l'avons vu, de nombreux bits doivent être définis de certaines manières pour que l'ordinateur fasse quelque chose d'utile. Les bits dans l'ordinateur sont toujours là. Si vous voulez que l'ordinateur fasse une certaine chose, vous activez ou désactivez ces bits selon le modèle qui fera que l'ordinateur fera ce que vous voulez qu'il fasse. Ce modèle est appelé logiciel. Ce n'est pas une chose physique, c'est juste le modèle dans lequel les bits sont définis.

Ainsi, la différence entre le matériel et le logiciel n'est pas comme le métal contre le caoutchouc. Le métal et le caoutchouc sont tous deux du matériel en ce qui concerne la définition de l'ordinateur.

Le matériel est quelque chose que vous pouvez prendre, voir, manipuler.

Le logiciel est la façon dont le matériel est configuré. Lorsque vous achetez un logiciel, il est enregistré sur quelque chose, généralement une sorte de disque. Le disque est matériel, le motif spécifique enregistré sur ce disque est logiciel. Un autre disque peut lui ressembler, mais avoir un logiciel complètement différent écrit dessus.

Une autre façon de voir la différence entre le matériel et le logiciel est de savoir à quel point il est facile de l'envoyer à distance.

Si vous avez un vase que vous voulez envoyer à votre tante Millie pour son anniversaire, vous devez emballer le vase dans une boîte et le faire transporter par un camion de chez vous à sa maison. Mais si vous voulez lui offrir de la musique en cadeau, vous pouvez aller au magasin, lui acheter un disque et l'envoyer par la poste, mais vous pouvez aussi lui acheter un chèque-cadeau sur Internet, lui envoyer un e-mail et lui faire télécharger la musique. Dans ce cas, la musique arrivera chez elle sans qu'un camion n'ait à s'y rendre. La musique sera transportée uniquement par le modèle d'électricité qui passe par la connexion Internet jusqu'à sa maison.

Une autre façon de voir la différence entre le matériel et le logiciel est la facilité avec laquelle il est possible de faire une copie de l'élément.

Si vous avez une tondeuse à gazon et que vous voulez une deuxième tondeuse à gazon, il n'y a pas de machine qui copiera la tondeuse à gazon. Vous pourriez photographier la tondeuse à gazon, mais vous n'auriez qu'une photographie plate d'une tondeuse à gazon. Vous ne pourriez tondre aucune pelouse avec la photo. Pour obtenir une vraie deuxième tondeuse à gazon, vous devez retourner à l'usine de tondeuses à gazon et en construire une autre avec du fer, du plastique, de la corde et tout ce dont sont faites les tondeuses à gazon. C'est

Matériel.

Le logiciel peut être facilement copié par la machine. Tout ce dont vous avez besoin est quelque chose qui peut lire le disque ou tout ce sur quoi il est enregistré, et quelque chose d'autre pour l'écrire sur un nouveau disque. Le nouveau sera comme l'original, il fera les mêmes choses. Si l'original est votre film préféré, la copie sera également votre film préféré. Si l'original est un programme qui préparera vos documents fiscaux, la copie le sera aussi.

Le logiciel n'est pas une chose physique, c'est juste la façon dont les choses physiques sont définies.

La définition de loin la plus couramment utilisée du terme « logiciel » consiste à faire référence à un ensemble de codes d'instructions informatiques. Je pense que la façon dont il a reçu ce nom est qu'une fois que vous avez construit un appareil aussi polyvalent qu'un ordinateur, il peut être fait pour faire beaucoup de choses différentes. Mais quand il n'y a pas d'instructions dedans, il ne peut rien faire. Le logiciel est donc une partie absolument nécessaire d'un ordinateur qui effectue une tâche. C'est une partie vitale de la machine totale, mais ce n'est pas comme les autres parties de la machine. Vous ne pouvez pas le peser ou le mesurer ou le ramasser avec une paire de pinces. Cela fait donc partie du « matériel », mais ce n'est pas du matériel. La seule chose qui reste à l'appeler est "logiciel".

Programmes

Comme mentionné précédemment, une série d'instructions dans la RAM s'appelle un programme.

Les programmes existent en plusieurs tailles. Généralement, un programme est un morceau de logiciel qui a tout le nécessaire pour effectuer une tâche spécifique. Un système serait quelque chose de plus grand, composé de plusieurs programmes. Un programme peut être composé de plusieurs parties plus petites appelées « routines ». Les routines peuvent à leur tour être constituées de sous-routines.

Il n'y a pas de définitions précises qui différencient système, programme, routine et sous-routine. Le programme est le terme général pour tous, la seule différence est leur taille et la façon dont ils sont

utilisé.

Il existe une autre distinction entre deux types de programmes qui n'est pas liée à leur taille. La plupart des ordinateurs personnels et professionnels ont un certain nombre de programmes installés sur eux. La plupart de ces programmes sont utilisés pour faire quelque chose que le propriétaire veut faire. Ceux-ci sont appelés programmes d'application car ils sont écrits pour appliquer l'ordinateur à un problème qui doit être résolu. Il existe un programme sur la plupart des ordinateurs qui n'est pas une application. Son rôle est de s'occuper de l'ordinateur lui-même et d'assister les programmes d'application. Ce programme qui n'est pas une application s'appelle le système d'exploitation.

Le système d'exploitation

Un « système d'exploitation », ou « OS » en abrégé, est un vaste programme qui comporte de nombreuses parties et plusieurs objectifs.

Son premier travail consiste à mettre l'ordinateur en marche lorsque vous l'allumez pour la première fois.

Une autre de ses tâches consiste à démarrer et à terminer les programmes d'application et à donner à chacun le temps de s'exécuter. C'est le "patron" de tous les autres programmes de cet ordinateur. Lorsque plus d'un programme est en RAM, c'est le système d'exploitation qui bascule entre eux. Il laisse un programme s'exécuter pendant une petite fraction de seconde, puis un autre programme, puis un autre programme. S'il y a dix programmes dans la RAM et que chacun s'exécute pendant un centième de seconde à la fois, chaque programme serait capable d'exécuter des millions d'instructions pendant ce temps, plusieurs fois par seconde. Il semblerait que les dix programmes s'exécutent simultanément parce que chacun peut faire quelque chose, plus vite que l'œil ne peut le voir.

Un système d'exploitation fournit également des services aux programmes d'application. Lorsqu'un programme d'application a besoin de lire ou d'écrire sur le disque, ou de dessiner des lettres à l'écran, il n'a pas à faire toutes les

instructions d'E/S compliquées nécessaires pour accomplir la tâche. Le système d'exploitation dispose d'un certain nombre de petites routines qu'il conserve en permanence dans la RAM à ces fins.

Tout ce qu'une application doit faire pour utiliser l'une de ces routines est de charger des informations dans les registres, puis de sauter à l'adresse de la bonne routine du système d'exploitation. Voici un exemple de la façon dont cela pourrait être fait. Disons que vous voulez dessiner un personnage sur l'écran. Tout d'abord, mettez le code ASCII du caractère souhaité dans R0.

Ensuite, placez les numéros de ligne et de colonne de l'endroit où vous souhaitez qu'il apparaisse à l'écran dans R1 et R2. Et voici la partie délicate : vous mettez l'adresse de la prochaine instruction de votre programme d'application, dans R3. Passez maintenant à la routine du système d'exploitation. La routine s'occupera de tous les détails du dessin du caractère sur l'écran, puis sa dernière instruction sera JMPR R3. Ainsi, ces routines peuvent être "appelées" à partir de n'importe quelle application, et une fois terminées, la routine reviendra à l'instruction suivante.

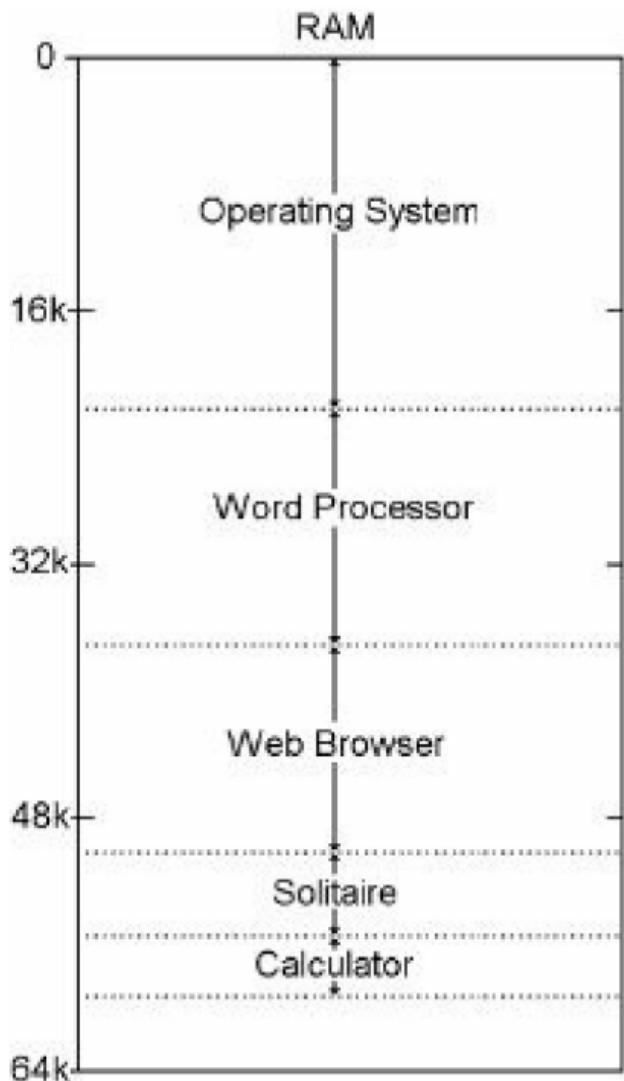
dans l'application qui l'a appelé.

Il existe plusieurs raisons pour lesquelles le système d'exploitation exécute toutes les fonctions d'E/S. La première est qu'il facilite l'écriture de programmes d'application, le programmeur n'a même pas besoin de savoir comment les périphériques fonctionnent réellement. Une autre raison est que cela gaspillerait beaucoup de RAM si chaque application avait sa propre copie de toutes les routines d'E/S.

L'une des raisons les plus importantes est que le système d'exploitation peut vérifier si le programme doit être autorisé à faire ce qu'il demande. Cela fait partie de l'autre tâche du système d'exploitation d'être le patron.

Le cœur du système d'exploitation est essentiellement une boucle d'instructions qui pose les questions suivantes : Dois-je saisir quoi que ce soit ? Dois-je sortir quelque chose ? Dois-je laisser un programme s'exécuter ? Puis ça recommence. Si la réponse à toutes ces questions est non, le processeur exécute simplement les instructions de cette boucle encore et encore, des millions de fois par seconde. Quand il y a quelque chose à faire, il saute au début du programme qui s'en occupe, et quand c'est fait, il revient à cette boucle où le système d'exploitation "attend" quelque chose d'autre à faire.

Voici un diagramme de notre plus grande version de RAM, montrant quelles parties de la RAM pourraient être occupées par un système d'exploitation et plusieurs autres programmes.



Dans la RAM de chaque programme, il y a tout le code d'instruction qui fait fonctionner le programme. Chaque programme peut être divisé en sa propre boucle principale et en de nombreuses routines utilisées pour les différentes tâches qu'il doit effectuer. Comme mentionné, le système d'exploitation possède également des routines qui peuvent être appelées par d'autres programmes.

Chaque programme utilise également une partie de son "espace d'adressage" pour les données sur lesquelles il travaille. La calculatrice, par exemple, doit avoir quelques octets où elle stocke les nombres que l'utilisateur y entre. Solitaire a besoin de quelques octets qui spécifient quelles cartes sont dans quelles positions. Le traitement de texte a besoin de RAM pour tous les codes ASCII qui composent le document sur lequel vous travaillez. Le système d'exploitation a également besoin d'octets où il peut stocker des polices, garder une trace de l'endroit où les programmes d'application ont été chargés, recevoir les données qu'il lit sur le disque et à de nombreuses autres fins.

Et c'est donc ce qui se passe à l'intérieur de votre ordinateur moyen. Il existe de nombreux programmes et zones de données différents dans la RAM. Le système d'exploitation saute à un programme, le programme saute à une routine, la routine saute à une sous-routine.

Chaque programme travaille sur ses données ou calcule quelque chose ou effectue une opération d'E/S. Au fur et à mesure que chacun se termine, il revient d'où il vient. Le CPU exécute une instruction d'un programme à la fois, et si elles sont écrites intelligemment, chaque programme fera son travail pièce par pièce, sans interférer avec le reste.

Si notre ordinateur avait inclus un "système d'interruption" comme nous l'avons décrit il y a quelques chapitres, chaque fois que quelqu'un appuierait sur une touche du clavier ou déplacerait la souris, il y aurait une interruption qui appellera une partie du système d'exploitation qui détermine quel I/O a provoqué l'interruption, puis appelle la routine appropriée pour s'occuper de ce qu'il s'agissait. Lorsque cela était fait, le processeur continuait avec l'instruction suivante du programme en cours d'exécution au moment de l'interruption.

Tout cela peut sembler très complexe, avec tant de millions et de milliards d'instructions exécutées en un clin d'œil. Il existe des façons d'organiser les programmes et de bonnes pratiques de programmation qui peuvent le rendre beaucoup plus compréhensible. Une étude de ceux-ci simplifierait le logiciel de la même manière que j'espère que ce livre a simplifié

le matériel. Mais ce serait le sujet d'un autre livre entier.

Langages Écrire

des programmes est très difficile à faire lorsque vous n'écrivez que des uns et des zéros, mais c'est le seul code que le CPU "comprend".

Qu'est-ce qu'une langue ? Une langue parlée, comme l'anglais, est un moyen de représenter des objets, des actions et des idées avec des sons. Une langue écrite est une façon de représenter les sons d'une langue parlée avec des symboles sur papier.

Cela ressemble à un autre code, et un code représentant un code. Nous ne pouvons tout simplement pas nous éloigner de ces choses !

Vous souvenez-vous de ce raccourci que nous avons utilisé lorsque nous examinions le code d'instruction du processeur et le câblage dans la section de contrôle ? Eh bien, c'est en fait quelque chose de plus qu'un simple outil pratique qui a été inventé pour ce livre. C'est un langage informatique. Voici quelques lignes de

ce:

Instruction Code		Langue		Sens
1000	AJOUTER		RA, RB	Ajouter
1001	rare SHR		RA, RB	décalage vers la droite
1010	rarb SHL		RA, RB	décalage vers la gauche
1011	rarb PAS		RA, RB	Non

Un langage informatique est une manière de représenter le code d'instruction.
Son but est de faciliter l'écriture de programmes informatiques.

Pour utiliser ce langage, vous écrivez le programme que vous voulez avec des caractères ASCII, et vous l'enregistrez dans un fichier.

Ensuite, vous chargez un programme spécial appelé "compilateur" dans la RAM et sautez à sa première instruction. Le compilateur lira le fichier ASCII, traduira chaque ligne dans le code d'instruction qu'il représente et écrira tous les octets du code d'instruction dans un second fichier. Le deuxième fichier peut ensuite être chargé dans la RAM, et lorsque le CPU passe à sa première instruction, le programme que vous avez écrit en ASCII fera, espérons-le, ce que vous voulez qu'il fasse.

Bien entendu, lorsque les ordinateurs ont été inventés pour la première fois, tous

les programmes devaient être écrits directement en uns et en zéros.

Puis quelqu'un s'est lassé de l'ennui de programmer de cette façon et a décidé d'écrire le premier compilateur. Ensuite, les programmes ont été écrits dans ce langage plus simple, puis traduits en code d'instruction par le compilateur. Avec le compilateur d'origine, vous pourriez même écrire un meilleur compilateur.

Donc, pour qu'un langage informatique existe, vous avez besoin de deux choses, un ensemble de mots qui composent le langage (un autre code) et un compilateur qui compile le langage écrit en code d'instruction informatique.

La langue que nous avons vue dans ce livre ne contient qu'environ 20 mots. Chaque mot correspond directement à l'une des instructions dont cet ordinateur est capable. Chaque ligne que vous écrivez donne lieu à une instruction informatique. Lorsque vous écrivez un programme de 87 lignes dans ce langage, le fichier de code d'instructions généré par le compilateur contiendra 87 instructions.

Ensuite, quelqu'un a inventé un langage de "niveau supérieur" dans lequel une ligne du langage pouvait aboutir à plusieurs instructions informatiques. Par exemple, notre ordinateur n'a pas d'instruction qui effectue une soustraction. Mais le

Le compilateur pourrait être conçu de manière à reconnaître un nouveau mot dans le langage tel que "SUB RA, RB", puis à générer le nombre d'instructions machine nécessaires pour que la soustraction se produise. Si vous pouvez comprendre comment faire quelque chose de fantaisiste avec 47 instructions, vous pouvez avoir un mot dans votre langue qui signifie cette chose fantaisiste.

Ensuite, quelqu'un a inventé un langage de niveau encore plus élevé où les mots qui composent le langage ne ressemblent même pas aux instructions réelles du processeur. Le compilateur a encore beaucoup de travail à faire, mais génère toujours un code d'instruction qui fait ce que signifient les mots dans cette langue. Quelques lignes d'un langage de niveau supérieur pourraient ressembler à ceci : Solde = 2 000 Taux d'intérêt = 0,034 Imprimer « Bonjour Joe, votre intérêt cette année est de : \$ »

Imprimer le solde X le taux d'intérêt

Le compilateur de ce langage lira ce programme de quatre lignes et générera un fichier qui pourrait facilement contenir

centaines d'octets de code d'instruction. Lorsque ce code d'instruction était chargé dans la RAM et exécuté, il affichait : Bonjour Joe, votre intérêt cette année est : 68 \$

L'écriture de logiciels dans des langages de niveau supérieur peut permettre d'en faire beaucoup plus en moins de temps, et le programmeur n'a plus besoin de savoir exactement comment l'ordinateur fonctionne réellement.

Il existe de nombreux langages informatiques. Certains langages sont conçus pour faire du travail scientifique, certains sont conçus à des fins commerciales, d'autres sont à usage plus général.

Les langages de niveau inférieur sont toujours les meilleurs à certaines fins.

Le système de fichiers

Comme nous l'avons vu précédemment, le fonctionnement réel d'un disque est assez étranger à la plupart des personnes qui utilisent un ordinateur.

Pour faciliter les choses, quelqu'un a inventé une idée appelée "fichier". Un dossier est censé être similaire au type de dossiers papier que les gens utilisent dans les bureaux. Un dossier papier est une feuille de carton pliée en deux et placée dans un classeur. Ce dossier a un onglet sur lequel vous pouvez écrire une sorte de nom pour le dossier, puis vous pouvez mettre un ou plusieurs morceaux de papier dans le dossier.

Un fichier informatique est une chaîne d'octets qui peut être de n'importe quelle longueur, d'un octet jusqu'à tous les octets disponibles sur le disque. Un fichier a aussi un nom. Un disque peut contenir plusieurs fichiers, chacun avec son propre nom.

Bien sûr, ces fichiers ne sont qu'une idée. Pour faire fonctionner un système de fichiers, le système d'exploitation fournit un tas de logiciels qui font apparaître le disque comme un classeur au lieu d'avoir des têtes, des pistes, des secteurs et des blocs d'octets.

Ce système de fichiers permet aux programmes d'application d'utiliser facilement le disque. Les applications peuvent demander au système d'exploitation de créer, lire, écrire ou effacer quelque chose appelé un fichier.

Tout ce que l'application a besoin de savoir, c'est le nom du fichier. Vous l'ouvrez, demandez des octets, envoyez-lui des octets, agrandissez ou réduisez le fichier, fermez le fichier.

Le système d'exploitation utilise une partie du disque pour maintenir une liste de noms de fichiers, ainsi que la longueur de chaque fichier et l'adresse du disque (tête, piste, secteur) du premier secteur des données. Si le fichier est plus petit qu'un secteur de disque, c'est tout ce dont vous avez besoin, mais si le fichier est plus grand qu'un secteur, il existe également une liste contenant autant d'adresses de type de disque que nécessaire pour contenir le fichier.

Le programme d'application dit de créer un fichier avec le nom "lettre à Jane". Ensuite, l'utilisateur tape la lettre à Jane et l'enregistre. Le programme indique au système d'exploitation où se trouve la lettre dans la RAM et sa durée, et le système d'exploitation l'écrit sur le disque dans le ou les secteurs appropriés et met à jour la longueur du fichier et toutes les listes nécessaires de type de disque.

adresses.

Pour utiliser le système de fichiers, il y aura une sorte de règles que le programme d'application devra suivre. Si vous souhaitez écrire des octets sur le disque, vous devez indiquer au système d'exploitation le nom du fichier, l'adresse RAM des octets que vous souhaitez écrire et le nombre d'octets à écrire. En règle générale, vous placeriez toutes ces informations dans une série d'octets quelque part dans la RAM, puis vous placeriez l'adresse RAM du premier octet de ces informations dans l'un des registres, puis vous exécuteriez une instruction Jump qui saute à une routine dans le Système d'exploitation qui écrit des fichiers sur le disque. Tous les détails sont pris en charge par cette routine, qui fait partie du système d'exploitation.

Si vous demandez au système d'exploitation de regarder votre disque, il vous montrera une liste de tous les noms de fichiers, et généralement leurs tailles et la date et l'heure auxquelles ils ont été écrits pour la dernière fois.

Vous pouvez stocker toutes sortes de choses dans des fichiers. Les fichiers ont généralement des noms composés de deux parties séparées par un point comme « xxxx.yyy ». La partie avant le point est une sorte de nom comme "lettre à Jane", et la partie après le point est une sorte de type comme "doc" qui est l'abréviation de "document". La partie avant le point vous dit quelque chose sur le contenu du fichier.

La partie après le point vous indique quel type de données est contenu dans ce fichier, en d'autres termes, quel code il

les usages.

Le type de fichier indique à vous et au système d'exploitation quel code les données du fichier utilisent. Dans un système d'exploitation populaire, ".txt" signifie texte, ce qui signifie que le fichier contient ASCII. Un ".bmp" signifie BitMaP, qui est une image. Un ".exe" signifie exécutable, ce qui signifie qu'il s'agit d'un programme et qu'il contient donc un code d'instruction.

Si vous demandez au système d'exploitation quels programmes sont disponibles pour être exécutés, il vous montrera une liste des fichiers qui se terminent par ".exe". Si vous demandez une liste d'images que vous pouvez regarder, il vous montrera une liste de fichiers qui se terminent par ".bmp".

Il existe de nombreux types de fichiers possibles, n'importe quel programme peut inventer son propre type et utiliser n'importe quel code ou combinaison de codes.

les erreurs

L'ordinateur est une machine assez complexe qui fait une série de choses simples les unes après les autres très rapidement.

Quelles sortes de choses pourraient mal tourner ici?

Au début de l'informatique, lorsque chaque porte de l'ordinateur était relativement coûteuse à construire, il y avait parfois des composants qui avaient en fait des pièces mobiles pour établir des connexions électriques. Deux morceaux de métal devaient se toucher pour que l'électricité aille là où les constructeurs voulaient qu'elle aille. Parfois, lorsque la machine cessait de fonctionner correctement, le réparateur regardait à l'intérieur pour découvrir ce qui n'allait pas, et il découvrait qu'une araignée s'était glissée à l'intérieur de la machine et s'était coincée entre deux de ces pièces de métal censées toucher chacun des autres. Ensuite, lorsqu'un morceau de métal se déplaçait pour toucher l'autre, l'araignée était sur le chemin et ils ne se touchaient pas. Ainsi, l'électricité n'arriverait pas là où elle devait aller et la machine ne fonctionnerait plus correctement. Le réparateur supprimait le bogue, nettoyait les contacts et signalait « Il y avait un bogue dans l'ordinateur ». Et il voulait littéralement dire un bug.

Au fil du temps, chaque fois qu'un ordinateur semblait fonctionner de manière incorrecte, les gens disaient que l'ordinateur avait un bogue. Il existe deux grandes catégories de bugs informatiques : matériels et logiciels.

Un bogue matériel signifie en fait que l'ordinateur est en panne. Cela pourrait être aussi grave que vous allumez l'ordinateur et qu'il prend feu, car il y a un octet dans la RAM où un bit est toujours désactivé.

Maintenant, un bit dans la RAM qui refuse de changer peut être un problème ou non. Si l'octet où se trouve ce bit n'est jamais utilisé, l'ordinateur fonctionnera très bien. Si cet octet fait partie d'un endroit où un nom est stocké, le nom peut être changé de "Joe" à "Jod". Si cet octet contient des instructions de programme, une instruction XOR peut être remplacée par une instruction JMP. Ensuite, lorsque le programme arrivera à cette instruction, il ne fera pas le XOR comme il est censé le faire, mais il sautera plutôt ailleurs et commencera à exécuter tout ce qui se trouve au nouvel emplacement comme s'il

était une série d'instructions. Le contenu de ces octets déterminera ce qui se passera ensuite, mais ce sera presque certainement aussi faux qu'un train qui tombe de sa voie.

Si une porte est cassée dans le stepper, par exemple, de sorte que l'étape 4 ne s'allume jamais, alors l'ordinateur ne pourra pas vraiment fonctionner du tout. Il serait toujours capable de récupérer les instructions des étapes 1, 2 et 3, mais chaque instruction s'exécuterait de manière incorrecte. Certes, le programme créerait un gâchis après avoir «exécuté» seulement quelques instructions.

Les bogues logiciels peuvent prendre de nombreuses formes, mais ce sont tous en fin de compte des erreurs de programmeur. Il y a probablement beaucoup plus de façons d'écrire un programme incorrectement que correctement.

Certaines erreurs créent simplement une sorte de résultats incorrects, et d'autres erreurs provoquent un «plantage» de l'ordinateur.

Une de mes histoires préférées de programmeur stupide est celle-ci : Quelqu'un a acheté une voiture à crédit. Il a reçu un carnet de coupons avec le prêt, un coupon à envoyer avec chaque paiement. Mais lorsqu'il a effectué son premier paiement, il a accidentellement utilisé le dernier coupon du livre au lieu du premier. Quelques semaines plus tard, il a reçu une lettre générée par ordinateur de la société de prêt disant : "Merci d'avoir remboursé intégralement votre prêt, la prochaine fois que vous aurez besoin d'un prêt, veuillez nous utiliser à nouveau." De toute évidence, le programme vient de vérifier le numéro de coupon et s'il était égal au numéro de coupon le plus élevé du livre, passez à la routine pour un prêt entièrement remboursé. Il aurait dû au moins vérifier le solde restant sur le prêt avant de décider qu'il a été remboursé. Il s'agit d'une erreur subtile, elle pourrait ne pas être détectée par la société de prêt jusqu'à ce qu'elle ait vérifié ses livres des mois plus tard. L'ordinateur a fait exactement ce qu'on lui disait de faire, et la plupart du temps c'était suffisant, mais le programme n'a pas été écrit pour anticiper toutes les situations qui se produisent parfois dans

le vrai monde.

L'un des pires bogues logiciels est de rester coincé dans une boucle. Le programme exécute une série d'instructions, puis revient au début de la série et l'exécute encore et encore. Bien sûr, les boucles sont utilisées tout le temps en programmation, mais elles sont utilisées pour faire quelque chose qui a un nombre fini d'étapes similaires.

Il peut se répéter jusqu'à ce que 50 octets aient été déplacés quelque part, ou continuer à vérifier que l'utilisateur appuie sur une touche du clavier. Mais l'ordinateur sortira de la boucle à un moment donné et passera à sa tâche suivante. Mais s'il y a une sorte d'erreur de programmation où il y a une boucle sans issue, l'ordinateur semblera complètement bloqué. Cela s'appelle parfois être "bloqué", l'ensemble de l'ordinateur peut avoir besoin d'être éteint et redémarré pour sortir de la boucle et revenir à un fonctionnement utile.

Il y a toutes sortes d'erreurs qui aboutissent au fait que le CPU essaie d'exécuter autre chose que le code d'instruction.

Supposons que votre programme réside aux adresses 10 à 150 et que vous ayez des données ASCII telles que des noms et des numéros de téléphone aux adresses 151 à 210. Si le programme est écrit de manière incorrecte de sorte que, dans certaines conditions, il sautera à l'adresse 180, il continuera simplement à récupérer et à exécuter les octets à partir de l'adresse 180. Si 180-189 était rempli avec l'ASCII pour "Jane Smith", le "programme" exécutera maintenant des ordures complètes, une série d'octets qui n'ont pas été conçus pour être Instruction Code. Il peut se mettre dans une boucle, revenir quelque part dans le programme ou émettre la commande pour effacer le lecteur de disque. Et il fera des ordures à sa grande vitesse habituelle. Si vous regardiez les modèles dans les octets, vous pouviez voir ce que cela ferait, mais cela pourrait être à peu près n'importe quoi. Si le nom à l'adresse 180 était "Bill Jones", il ferait quelque chose de complètement différent. Puisqu'il n'est pas conçu pour être utile, il est fort probable qu'il continuera à faire un plus gros gâchis de ce qui est en mémoire jusqu'à ce que l'ordinateur doive être éteint pour l'arrêter.

Un autre type d'erreur pourrait se produire si un programme écrivait accidentellement "John Smith" à l'endroit où une police était stockée. Dans ce cas, chaque lettre "E" qui a obtenu

dessiné sur l'écran par la suite ressemblerait à ceci : '.'



L'ordinateur exécute des centaines de millions d'instructions chaque seconde, et il suffit d'une seule mauvaise instruction pour que tout s'arrête brutalement. Par conséquent, le sujet de la programmation d'ordinateurs d'une manière qui sera complètement "sans bogue" est quelque chose

qui attire beaucoup d'attention. Presque toute la programmation est effectuée avec des langages, et les compilateurs de ces langages sont conçus pour générer un code d'instruction qui évite les types d'erreurs les plus graves et pour avertir le programmeur si certaines bonnes pratiques de programmation sont violées. Pourtant, les compilateurs peuvent avoir des erreurs, et ils ne pourront jamais repérer une erreur comme celle ci-dessus avec le prêt automobile.

Comme vous pouvez le voir, l'ordinateur et ses logiciels sont des choses assez fragiles. Chaque porte doit fonctionner à chaque fois et chaque instruction exécutée doit être correcte.

Lorsque vous considérez toutes les choses qui pourraient mal tourner, le pourcentage élevé de choses qui vont normalement bien est en fait assez impressionnant.

Maladies informatiques ?

Un autre endroit où des caractéristiques humaines sont attribuées aux ordinateurs est ce qu'on appelle un virus informatique. Cela implique que les ordinateurs peuvent tomber malades et tomber malades. Vont-ils commencer à tousser et à éternuer ?

Vont-ils attraper un rhume ou la varicelle ? Qu'est-ce qu'un virus informatique exactement ?

Un virus informatique est un programme écrit par quelqu'un qui veut faire du mal à vous et à votre ordinateur. C'est un programme qui va faire une sorte de mal à votre ordinateur lorsqu'il s'exécute. La motivation des personnes qui écrivent des programmes antivirus va du simple défi technique de voir si l'on est capable de le faire, au désir de faire tomber l'économie du monde entier. Dans tous les cas, les personnes qui font de telles choses n'ont pas votre meilleur intérêt à l'esprit.

Comment un ordinateur « attrape-t-il » un virus ? Un programme antivirus doit être placé dans votre RAM et votre ordinateur doit accéder au programme antivirus et l'exécuter. Lorsqu'il s'exécute, il localise un fichier qui se trouve déjà sur votre disque dur et qui contient un programme exécuté régulièrement par votre ordinateur, comme une partie du système d'exploitation.

Une fois que le programme antivirus a localisé ce fichier, il copie le programme antivirus à la fin de ce fichier et insère une instruction de saut au début du fichier qui provoque un saut vers l'endroit où se trouve le programme antivirus. Maintenant, votre ordinateur a un virus.

Lorsqu'un ordinateur infecté par un virus est en cours d'exécution, il fait tout ce qu'il est censé faire, mais chaque fois qu'il exécute le programme contenant le virus, l'instruction de saut insérée entraîne l'exécution du programme antivirus à la place.

Maintenant, le virus fera généralement quelque chose de simple, comme vérifier une date pré-déterminée, et s'il ne s'agit pas d'une correspondance, le programme antivirus reviendra au début du fichier où le programme du système d'exploitation existe toujours.

Ainsi, votre ordinateur apparaîtra tout à fait normal, il n'y a que quelques instructions supplémentaires exécutées au cours de ses opérations régulières. Le virus est considéré comme dormant à ce stade. Mais quand cette date arrive, et que le virus

'décide' de faire ce qui est dans le reste de son programme, ça peut être n'importe quoi. Lorsque le programme antivirus est en cours d'exécution, il peut faire tout le mal auquel la personne qui l'a écrit pourrait penser. Il peut effacer des fichiers sur votre disque ou les envoyer ailleurs via Internet. Un virus humoristique ferait, de temps à autre, donner l'impression que les lettres à l'écran se détachaient et tombaient en tas au bas de l'écran.

Voici un exemple de comment attraper un virus. Disons que vous avez un ami qui trouve un film amusant sur Internet. Cela le fait rire et il pense que vous l'apprécierez aussi, alors il vous envoie le fichier du film par e-mail.

Vous recevez le fichier vidéo et le jouez, et vous l'appréciez.

Il y a deux choses différentes qui auraient pu se produire ici. Si votre ami vous a envoyé un fichier nommé "funny.mov" et que votre système d'exploitation inclut un programme qui lit les fichiers ".mov", alors le système d'exploitation chargera ce programme dans la RAM et ce programme lira les images dans le fichier "funny.mov". mov" et affichez-les sur votre écran. C'est très bien, le programme qui a fonctionné était quelque chose qui était déjà sur votre ordinateur. Le fichier "funny.mov" vient de fournir une série d'images qui ont été affichées sur votre écran.

Mais si votre ami vous a envoyé un fichier nommé "funny.exe", alors lorsque vous demandez au système d'exploitation de lire le film, il chargerà "funny.exe" dans la RAM et passera à sa première instruction.

Vous avez maintenant un programme en cours d'exécution sur votre ordinateur qui vient d'ailleurs. S'il s'agit d'un programme antivirus, il lira probablement le film pour que vous ne soupçonnez rien, mais il peut faire tout ce qu'il veut sur les fichiers de votre disque pendant que vous regardez le film. Il s'installera probablement de lui-même et entrera dans un état inactif pendant des jours ou des semaines, et vous ne saurez même pas que votre ordinateur est « infecté ». Mais tôt ou tard, il prendra vie et fera les dégâts pour lesquels il a été conçu

faire.

Ce type de programme malveillant est appelé virus car son fonctionnement est similaire à celui des virus réels qui infectent les êtres vivants. Un vrai virus est une chose plus petite qu'un animal unicellulaire. Il n'est pas tout à fait considéré comme vivant car le virus par lui-même ne peut pas se reproduire. Ils se reproduisent cependant en

envahir une cellule de quelque chose de vivant. Une fois dans la cellule, le virus utilise les mécanismes de cette cellule pour faire des copies de lui-même, qui peuvent ensuite continuer et infecter d'autres cellules.

Le virus informatique ne peut pas non plus se reproduire ou faire quoi que ce soit d'autre par lui-même. Il doit entrer dans un ordinateur et se faire exécuter une fois par ce processeur. Lorsqu'il s'exécute pour la première fois, il s'insère quelque part dans le système d'exploitation afin qu'il soit ensuite exécuté régulièrement. Ces instructions feront tout dommage qu'elles sont censées causer à l'ordinateur sur lequel elles s'exécutent, et elles feront aussi généralement quelque chose qui est conçu pour propager le virus à d'autres ordinateurs.

Micrologiciel

Bien sûr, la RAM est un élément essentiel de tout ordinateur. La possibilité d'écrire des octets dans la RAM et de les relire fait partie intégrante du fonctionnement de la machine.

Mais sur certains ordinateurs, certaines sections de la RAM ne sont écrites qu'au démarrage de l'ordinateur, et par la suite, ces sections restent inchangées pendant le fonctionnement de l'ordinateur. Cela pourrait être vrai sur n'importe quel ordinateur qui exécute toujours le même programme. Peut-être que la moitié de la RAM est utilisée pour contenir le programme, et l'autre moitié de la RAM est utilisée pour contenir les données sur lesquelles le programme travaille. La moitié avec le programme doit être chargée à un moment donné, mais après cela, le CPU n'a plus qu'à lire les octets du programme pour les récupérer et les exécuter.

Lorsque vous avez ce genre de situation, vous pouvez construire la moitié de la RAM de votre ordinateur de manière normale, et avec l'autre moitié, vous ignorez les portes NAND et connectez simplement chaque bit directement à un on ou un off dans le modèle de votre

programme.

Bien sûr, vous ne pouvez pas écrire dans la RAM pré-câblée, mais vous pouvez très bien lire à partir de celle-ci. Ce type de RAM a reçu le nom de Read Only Memory, ou ROM en abrégé. Vous l'utilisez de la même manière que vous utilisez la RAM, mais vous ne lisez qu'à partir de celle-ci.

Il y a deux avantages à la ROM. Au début des ordinateurs, lorsque la RAM était très chère, la ROM était beaucoup moins chère que la RAM.

L'autre avantage est que vous n'avez plus besoin de charger le programme dans la RAM lorsque vous allumez l'ordinateur pour la première fois. Il est déjà présent dans la ROM, prêt à être exécuté par le CPU.

Le point ici est un nouveau mot. Étant donné que le logiciel a été nommé "soft" parce qu'il est modifiable, en ce qui concerne la ROM, vous avez toujours un motif dans les bits, mais ils ne sont plus si soft. Vous ne pouvez pas écrire dans une ROM, vous ne pouvez pas changer les bits. Et c'est ainsi que ce type de mémoire est devenu connu sous le nom de "firmware". C'est un logiciel qui est écrit en permanence dans le matériel.

Mais ce n'est pas la fin de l'histoire. La ROM décrite ci-dessus devait être construite de cette façon en usine. Au fil des ans, cette idée a été améliorée et rendue plus facile à utiliser.

L'avancée suivante a eu lieu lorsque quelqu'un a eu la brillante idée de créer une ROM où chaque bit était activé en usine, mais il y avait un moyen d'y écrire avec beaucoup de puissance qui pouvait brûler des connexions individuelles, en changeant des bits individuels en off . Ainsi, cette ROM pourrait être programmée après avoir quitté l'usine. Cela s'appelait « ROM programmable » ou « PROM » en abrégé.

Puis quelqu'un a compris comment créer une PROM qui réparerait toutes ces connexions brisées si elle était exposée à la lumière ultraviolette pendant une demi-heure. Cela s'appelait une «PROM effaçable», ou «EPROM» en abrégé.

Ensuite, quelqu'un a compris comment construire une EPROM qui pourrait être effacée en utilisant une alimentation supplémentaire sur un fil spécial intégré à l'EPROM. Cela s'appelait «Electrically Erasable PROM», ou «EEPROM» en abrégé. Un type particulier d'EEPROM porte le nom de "mémoire Flash". Il y a donc RAM, ROM, PROM, EPROM, EEPROM et Flash.

Ce sont tous les types de mémoire d'ordinateur. La chose qu'ils ont en commun est qu'ils permettent tous un accès aléatoire.

Ils fonctionnent tous de la même manière lorsqu'il s'agit d'adresser les octets et de lire les données qu'ils contiennent. La grande différence est que la RAM perd ses paramètres lorsque l'alimentation est coupée. Lorsque le courant revient, la RAM est pleine de tous les zéros. Les autres ont toujours leurs données après la mise hors tension et sous tension.

Vous pouvez alors demander : « Pourquoi les ordinateurs n'utilisent-ils pas l'EEPROM pour leur RAM ? Ensuite, le programme restait dans la RAM lorsque l'ordinateur était éteint. La réponse est qu'il faut beaucoup plus de temps pour écrire dans l'EEPROM que dans la RAM. Cela ralentirait énormément l'ordinateur. Si quelqu'un trouve comment créer une EEPROM aussi rapide et aussi bon marché et utilise la même puissance ou moins que la RAM, je suis sûr que ce sera fait.

Soit dit en passant, le mot ROM est également utilisé pour désigner tout type de stockage défini de manière permanente, tel qu'un disque préenregistré, comme dans "CD ROM", mais sa définition d'origine ne s'appliquait qu'à quelque chose qui fonctionnait comme RAM.

Bottes

Qu'est-ce que les bottes ont à voir avec les ordinateurs ? Eh bien, il y a une vieille phrase qui dit "relevez-vous par vos propres bottes". C'est une sorte de blague, cela fait littéralement référence aux sangles cousues dans de nombreuses bottes qui sont utilisées pour aider à tirer les bottes sur vos pieds. La blague est que si vous portez une telle paire de bottes et que vous voulez vous lever du sol, au lieu d'obtenir une échelle ou de grimper à une corde, vous pouvez vous faire décoller en tirant simplement assez fort sur ces bootstraps. Bien sûr, cela ne fonctionnerait que dans un dessin animé, mais l'expression en est venue à signifier faire quelque chose quand il n'y a aucun moyen apparent de le faire, ou faire quelque chose sans les outils qui seraient normalement utilisés, ou accomplir quelque chose par vous-même sans l'aide de personne. autre.

Dans un ordinateur, il existe un problème similaire à la nécessité de démarrer et de ne pas avoir d'outils disponibles pour l'accomplir. Lorsqu'un ordinateur fonctionne, la mémoire est pleine de programmes qui font quelque chose, et lorsque l'opérateur de l'ordinateur entre une commande pour démarrer un autre programme, le système d'exploitation localise le programme sur le disque, le charge en mémoire et saute au première instruction du programme. Maintenant, ce programme est en cours d'exécution.

Mais lorsque vous allumez un ordinateur pour la première fois, comment mettez-vous le système d'exploitation en mémoire ? Il faut qu'un programme s'exécute en mémoire pour dire au lecteur de disque d'envoyer un code d'instruction, et le programme doit écrire ce code en mémoire à un endroit approprié, puis passer à sa première instruction pour lancer le nouveau programme. Mais lorsque vous allumez l'ordinateur, chaque octet en mémoire est composé de zéros. Il n'y a aucune instruction en mémoire. C'est la situation impossible, vous avez besoin d'un programme en mémoire pour obtenir un programme en mémoire, mais il n'y a rien là-bas. Donc, pour que l'ordinateur démarre en premier lieu, l'ordinateur doit faire quelque chose d'impossible. Il doit se ressaisir par ses bootstraps !

Il y a longtemps, au début des ordinateurs, la machine avait des commutateurs et des boutons poussoirs sur le panneau avant qui permettaient à l'opérateur d'entrer des octets de données

directement dans les registres, et de là, dans la RAM.

Vous pouvez saisir manuellement un programme court de cette manière et le lancer. Ce programme, appelé "chargeur d'amorçage", serait le plus petit programme possible que vous puissiez écrire qui demanderait à l'ordinateur de lire les octets d'un périphérique, de les stocker dans la RAM, puis de passer à la première instruction. Lorsque le chargeur d'amorçage s'exécute, il charge un programme beaucoup plus volumineux en mémoire, comme les débuts d'un système d'exploitation, puis l'ordinateur deviendra utilisable.

De nos jours, il existe des moyens beaucoup plus simples de charger le premier programme dans l'ordinateur. En fait, cela se produit automatiquement immédiatement après la mise sous tension de l'ordinateur. Mais ce processus se produit toujours, et la première étape s'appelle "démarrer" ou "démarrer" et cela signifie seulement mettre le premier programme en mémoire et commencer à l'exécuter.

La solution la plus courante à ce problème comporte trois parties. Tout d'abord, l'IAR est conçu de sorte que lors de la première mise sous tension, au lieu que tous ses bits soient à zéro, son dernier bit sera à zéro, mais le reste de ses bits sera à un. Ainsi, pour notre petit ordinateur, la première instruction à récupérer sera à l'adresse 1111 1110. Deuxièmement, quelque chose comme les 32 derniers octets de la RAM (235-256) seront à la place de la ROM, câblés avec un programme simple qui accède au lecteur de disque , sélectionne la tête 0, la piste 0, le secteur 0, lit ce secteur dans la RAM, puis saute au premier octet de celui-ci. La troisième partie aurait alors intérêt à ce qu'il y ait un programme écrit sur ce premier secteur du disque. Ce secteur, soit dit en passant, s'appelle le "boot record".

Ce mot « démarrer » est devenu un verbe dans le langage informatique. Cela signifie charger un programme dans la RAM où il n'y a pas de programmes. Parfois, les gens l'utilisent pour signifier charger n'importe quel programme dans la RAM, mais sa signification originale ne s'appliquait qu'au chargement du premier programme dans un autre vide

RAM.

Numérique vs analogique

Vous avez sans doute entendu parler de ces termes. Il semble que tout ce qui est associé aux ordinateurs soit numérique, et tout le reste ne l'est pas. Mais ce n'est pas assez proche de la vérité.

Ce qu'ils signifient est assez simple, mais d'où ils viennent et comment ils se sont retrouvés dans leur utilisation actuelle n'est pas si simple.

Le mot «numérique» vient de chiffre, qui signifie doigts et orteils dans une langue ancienne, et puisque les doigts et les orteils ont été utilisés pour compter, numérique signifie avoir à voir avec les nombres. Aujourd'hui, les symboles individuels que nous utilisons pour écrire les nombres (0, 1, 2, 3, etc.) sont appelés chiffres. Dans l'ordinateur, nous représentons les nombres avec des bits et des octets. L'une des qualités des bits et des octets est leur nature non ambiguë. Un bit est activé ou désactivé ; il n'y a pas de zone grise entre les deux. Un octet est toujours dans l'un de ses 256 états ; il n'y a pas d'état entre deux nombres comme 123 et 124. Le fait que ces états

changer par étapes, c'est ce à quoi nous faisons référence lorsque nous parlons de numérique.

Le mot « analogique » vient du même endroit que « analogie » et « analogue », il a donc à voir avec la similitude entre deux choses. Dans le monde réel, la plupart des choses changent progressivement et continuellement, pas par étapes. Une voix peut être un cri ou un murmure ou absolument n'importe où entre les deux. Lorsqu'un téléphone convertit une voix en un équivalent électrique afin qu'elle puisse se déplacer à travers un fil vers un autre téléphone, cette électricité peut également varier partout entre être complètement allumée et complètement éteinte.

Le son et l'électricité sont deux choses très différentes, mais l'essence de la voix a été dupliquée avec l'électricité. Puisqu'ils sont similaires à cet égard, nous pouvons dire que le schéma électrique est un "analogique" de la voix. Bien que le sens de « analogique » provienne de ce facteur de « similitude », lorsque vous faites un analogue, vous faites généralement un analogue de quelque chose qui est continuellement variable. Cette idée de quelque chose étant continuellement variable est devenue la définition de l'analogique lorsque vous comparez le numérique et l'analogique.

Quelque chose qui est analogique peut être n'importe où dans le

l'intégralité d'une gamme, il n'y a pas d'étapes.

Les moyens numériques changent par étapes et les moyens analogiques changent de manière continue et régulière. Une autre façon de le dire est que le numérique signifie que les éléments qui composent un tout proviennent d'un nombre fini de choix, tandis que l'analogique signifie qu'une chose est faite de parties qui peuvent être sélectionnées parmi un nombre illimité de choix. Quelques exemples non informatiques peuvent aider à clarifier cela.

Si vous avez une plate-forme à trois pieds au-dessus du sol, vous pouvez soit construire des escaliers pour que les gens puissent y monter, soit une rampe. Sur la rampe, vous pouvez monter à n'importe quel niveau entre le sol et la plate-forme ; dans les escaliers, vous n'avez qu'autant de choix qu'il y a de marches. La rampe est analogique, les escaliers sont numériques.

Disons que vous voulez construire une passerelle dans votre jardin. Vous avez le choix de faire la passerelle hors de
en béton ou en briques. Si les briques sont trois
pouces de large, vous pouvez faire une allée de briques de 30 pouces de large ou 33 pouces de large, mais pas de 31 ou 32 pouces. Si vous faites l'allée en béton, vous pouvez la couler à la largeur de votre choix. Les briques sont numériques, le béton est analogique.

Si vous avez un vieux livre et une vieille peinture à l'huile, et que vous voulez faire une copie de chacun, il vous sera beaucoup plus facile de faire une copie du livre. Même si les pages du livre sont jaunies, que les coins sont écornés et qu'il y a des taches de saleté et des trous de vers à l'intérieur, tant que vous pouvez lire chaque lettre du livre, vous pouvez retaper le texte entier, exactement comme l'auteur l'a voulu. Avec la peinture à l'huile, les couleurs d'origine peuvent s'être estompées et être obscurcies par la saleté. L'emplacement exact de chaque poil dans chaque coup de pinceau, l'épaisseur de la peinture à chaque endroit, la façon dont les couleurs adjacentes se mélangent, pourraient tous être copiés dans les moindres détails, mais il y aurait inévitablement de légères différences. Chaque lettre du livre provient d'une liste d'un nombre spécifique de possibilités; les variations de couleurs de peinture et leurs positions sur la toile sont illimitées. Le livre est numérique, la peinture est analogique.

Voilà donc la différence entre l'analogique et le numérique. Le monde qui nous entoure est essentiellement analogique. La plupart des anciennes technologies étaient analogiques, comme le téléphone,

phonographe, radio, télévision, magnétophones et vidéocassettes. Curieusement, l'un des appareils les plus anciens, le télégraphe, était numérique. Maintenant que la technologie numérique est devenue très développée et peu coûteuse, les appareils analogiques sont remplacés un par un par des versions numériques qui accomplissent les mêmes choses.

Le son est une chose analogique. Un téléphone à l'ancienne est une machine analogique qui convertit le son analogique en un modèle électrique qui est un analogue du son, qui se déplace ensuite à travers un fil vers un autre téléphone. Un nouveau téléphone numérique prend le son analogique et le convertit en un code numérique. Ensuite, le code numérique se déplace vers un autre téléphone numérique où le code numérique est reconvertis en son analogique.

Pourquoi quelqu'un se donnerait-il la peine d'inventer un téléphone numérique alors que le téléphone analogique fonctionnait parfaitement ?

La réponse, bien sûr, est que même si le téléphone analogique fonctionnait, il n'était pas parfait. Lorsqu'un schéma électrique analogique parcourt de longues distances, de nombreuses choses peuvent lui arriver en cours de route. Il devient de plus en plus petit au fur et à mesure qu'il se déplace, il doit donc être amplifié, ce qui introduit du bruit, et lorsqu'il se rapproche d'autres équipements électriques, une partie du schéma de l'autre équipement peut se mélanger à la conversation. Plus le son va loin, plus il y a de bruit et de distorsion. Chaque modification de l'analogique de votre voix devient une partie du son qui sort à l'autre bout.

Entrez la technologie numérique à la rescoussse. Lorsque vous envoyez un code numérique sur de longues distances, les bits individuels sont soumis aux mêmes types de distorsion et de bruit, et ils changent légèrement. Cependant, peu importe si un bit n'est activé qu'à 97 % au lieu de 100 %. L'entrée d'une porte n'a besoin que de "savoir" si le bit est activé ou désactivé, elle doit "décider" entre ces deux choix uniquement. Tant qu'un bit est encore plus qu'à mi-chemin, la porte dans laquelle il entre agira exactement de la même manière que si le bit avait été complètement allumé. Par conséquent, le modèle numérique à la fin est aussi bon qu'il l'était au début, et lorsqu'il est reconvertis en analogique, il n'y a aucun bruit ni distorsion, on dirait que la personne a raison

la porte à côté.

Il y a des avantages et des inconvénients à chaque méthode, mais en général, les avantages de la technologie numérique l'emportent largement sur ses lacunes.

Le plus grand avantage du numérique est probablement lié à la fabrication de copies. Lorsque vous faites une copie de quelque chose comme un disque vinyle, vous pouvez l'enregistrer sur un magnétophone, ou je suppose que vous pouvez même obtenir tout l'équipement nécessaire pour graver un nouveau disque vinyle. Mais il y aura un certain degré de différence entre l'original et la copie. En premier lieu, toutes les machines ont des limites de précision. Une copie de n'importe quel objet physique peut être très proche de l'original, mais jamais tout à fait exacte. Deuxièmement, s'il y a des rayures ou des particules de poussière sur l'original, la copie aura alors des doublons de ces défauts. Troisièmement, la friction entre le disque et l'aiguille use en fait une infime quantité de vinyle à chaque fois que vous le jouez. Si vous utilisez un magnétophone, il y a toujours un faible niveau de « siflement » ajouté au son. Si vous faites une copie d'une copie, et une copie de cela, etc., les modifications seront de plus en plus importantes à chaque étape.

Lorsqu'il s'agit de quelque chose qui est numérique, tant que chaque bit qui était activé dans l'original est également activé dans la copie, nous obtenons une copie exacte à chaque fois. Vous pouvez faire une copie de la copie, et une copie de cela, etc., et chacun d'eux sera exactement le même que l'original.

Le numérique est définitivement la voie à suivre si vous voulez pouvoir faire un nombre illimité de copies et conserver quelque chose pour toujours.

L'ordinateur et les périphériques que nous avons construits sont entièrement numériques jusqu'à présent. Et si tout ce que nous voulions faire avec eux était des choses numériques telles que l'arithmétique et le langage écrit, nous pourrions en rester là. Cependant, si nous voulons que notre ordinateur joue de la musique et travaille avec des photographies en couleur, il y a encore une chose que nous devons regarder.

J'ai menti - En quelque sorte

Il y a un élément matériel dans un ordinateur qui n'est pas entièrement composé de portes NAND. Cette chose n'est pas vraiment nécessaire pour faire d'un ordinateur un ordinateur, mais la plupart des ordinateurs en ont quelques-uns. Ils sont utilisés pour passer de quelque chose d'analogique à quelque chose de numérique, ou de numérique à analogique.

Les yeux et les oreilles humaines réagissent aux choses analogiques. Les choses que nous entendons peuvent être fortes ou douces, les choses que nous voyons peuvent être claires ou sombres et être d'une multitude de couleurs.

L'écran d'affichage de l'ordinateur que nous avons décrit ci-dessus avait 320 x 200 ou 64 000 pixels. Mais chaque pixel n'avait qu'un bit pour lui dire quoi faire, être allumé ou éteint. C'est très bien pour afficher le langage écrit sur l'écran, ou cela pourrait être utilisé pour faire des dessins au trait, tout ce qui n'a que deux niveaux de luminosité. Mais nous avons tous vu des photographies sur des écrans d'ordinateur.

Tout d'abord, il doit y avoir un moyen de mettre différentes couleurs sur l'écran. Si vous sortez une loupe et regardez un écran d'ordinateur ou de télévision couleur, vous verrez que l'écran est en fait composé de petits points de trois couleurs différentes, bleu, rouge et vert.

Chaque pixel comporte trois parties, une pour chaque couleur. Lorsque l'adaptateur d'affichage scanne l'écran, il sélectionne les trois couleurs de chaque pixel en même temps.

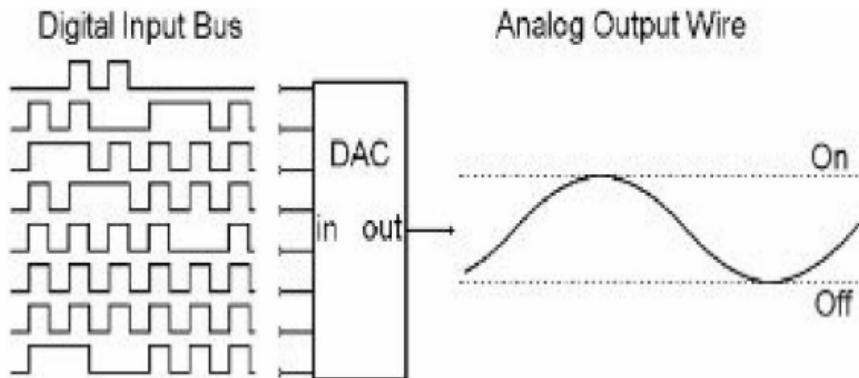
Pour qu'un ordinateur ait un écran couleur, il doit avoir trois bits pour chaque pixel, il devrait donc avoir trois fois la RAM afin de pouvoir contrôler les trois couleurs de chaque pixel individuellement. Avec trois bits, chaque couleur pourrait être entièrement activée ou désactivée, et chaque pixel aurait donc huit états possibles : noir, vert, rouge, bleu, vert et rouge (jaune), vert et bleu (cyan), bleu et rouge (magenta) et vert, bleu et rouge (blanc.)

Mais cela ne suffit toujours pas pour afficher une photographie. Pour ce faire, nous devons être en mesure de contrôler la luminosité de chaque couleur sur toute la plage entre complètement allumé et complètement éteint. Pour ce faire, nous avons besoin d'un nouveau type de pièce que nous décrirons sous peu, et nous avons besoin de plus de bits dans la RAM d'affichage. Au lieu d'un bit pour chaque couleur dans chaque

pixel, nous pourrions avoir un octet entier pour chaque couleur dans chaque pixel. C'est trois octets par pixel, pour un total de 192 000 octets de RAM rien que pour ce petit écran.

Avec ces octets, en utilisant le code binaire, vous pouvez spécifier 256 niveaux de luminosité pour chaque couleur de chaque pixel. Cela reviendrait à 16 777 216 états (ou couleurs) différents pour chaque pixel. C'est assez de variété pour afficher une photographie raisonnablement belle.

Pour que cela fonctionne - un nombre spécifiant 256 niveaux de luminosité différents - vous avez besoin d'une chose appelée "convertisseur numérique-analogique" ou "DAC" pour faire court. Un DAC a huit entrées numériques et une sortie analogique. La façon dont cela fonctionne est qu'il est câblé pour traiter l'entrée comme un nombre binaire, et la sortie a 256 niveaux entre off et on. La sortie a 256 gradations entre off et on, et elle va au niveau spécifié par le numéro d'entrée. Si l'entrée est un 128, la sortie sera à mi-chemin. Pour un 64 la sortie sera d'un quart de suite. Pour 0, la sortie sera complètement désactivée.



Pour faire fonctionner cet écran couleur, l'adaptateur d'affichage doit accéder à trois octets à la fois, les connecter à trois DAC et connecter les sorties des DAC aux trois couleurs du pixel actuel peint.

C'est ainsi que fonctionne un écran couleur.

Lorsque nous avons défini 'analogique' dans le dernier chapitre, nous avons dit

que c'était quelque chose qui variait continuellement de complètement éteint à complètement allumé. Mais notre DAC n'a en réalité que 256 niveaux différents sur sa sortie "analogique". C'est beaucoup plus proche d'être analogique qu'un peu, mais il y a encore des étapes. Ce que fait l'ordinateur, c'est se rapprocher d'une chose analogique par étapes suffisamment petites pour tromper le public visé. En ce qui concerne l'œil, 256 niveaux de luminosité différents suffisent.

Si quelque chose nécessite des étapes plus petites pour tromper le public visé, vous pouvez créer un DAC qui a 16 bits du côté numérique. Ainsi, vous pouvez présenter l'entrée numérique avec un nombre allant de 0 à 65535. Le côté analogique ne peut toujours varier que de complètement éteint à complètement allumé, mais la taille des étapes sera beaucoup plus petite puisqu'il y en a maintenant 65536.

En ce qui concerne l'oreille, elle peut entendre de très petites différences, et donc un DAC 16 bits est nécessaire pour un son de haute qualité.

Tous les sons, de la musique à la parole en passant par les coups de tonnerre, sont des vibrations de l'air. Ils varient dans la vitesse à laquelle l'air vibre et dans la façon exacte dont il vibre. L'oreille humaine peut entendre des vibrations d'environ 20 Hz à l'extrême basse jusqu'à

20 000 Hz (20 kHz) à l'extrême supérieure, c'est donc la gamme de vibrations que les ordinateurs sont conçus pour gérer.

Pour que toute machine électronique émette des sons, il existe un appareil appelé haut-parleur. Tout ce qu'un haut-parleur fait, c'est se déplacer d'avant en arrière dans l'air, ce qui fait vibrer l'air. S'il fait vibrer l'air exactement de la même manière que l'original qui a été enregistré, il sonnera exactement comme l'original.

Afin de stocker un son dans un ordinateur, la position du haut-parleur est divisée en 65536 positions possibles.

Puis une seconde est divisée en 44 100 parties. À chacune de ces parties de seconde, la position souhaitée du locuteur est stockée sous la forme d'un nombre à deux octets. C'est assez d'informations pour reproduire un son de très haute qualité.

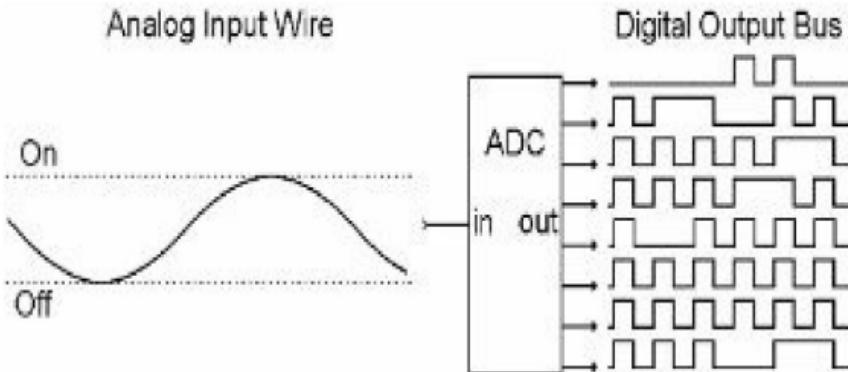
Pour jouer de la musique stéréo de qualité supérieure, un ordinateur aurait besoin d'un "périphérique audio". Cela aurait deux DAC 16 bits avec leurs sorties analogiques connectées aux haut-parleurs. Il aurait également sa propre horloge qui tourne à 44 100 Hz.

À chaque tick, il obtiendrait les deux prochains nombres à deux octets et les connecterait au côté numérique du

DAC.

En ce qui concerne la vitesse, ce serait 176 400 octets par seconde. C'est certainement rapide, mais rappelez-vous que l'horloge de notre ordinateur tourne un milliard de fois par seconde. Cela signifie que l'ordinateur peut envoyer quatre octets au périphérique audio, puis exécuter environ 4 000 instructions sur une autre tâche avant d'avoir besoin d'envoyer les quatre suivantes.

Pour aller dans l'autre sens, il existe un "convertisseur analogique-numérique" ou "ADC" en abrégé. Ceci est utilisé pour convertir le son d'un microphone en une série d'octets, ou pour un appareil photo pour convertir une image en une série d'octets. L'entrée a un fil qui peut être n'importe où de tout à fait à tout. L'ADC transforme ses sorties en un nombre compris entre 0 et 255 pour un ADC 8 bits ou entre 0 et 65 535 pour un ADC 16 bits. Ce nombre représente le degré d'activation ou de désactivation de l'entrée. La moitié correspond à 128 ou 32 768, un quart correspond à 64 ou 16 384, etc. Ce processus est tout simplement l'inverse de ce que fait un DAC.



Les DAC et ADC ne sont pas constitués de portes NAND, ils ont des composants électroniques comme les radios. Comment ils font ce qu'ils font n'est pas un sujet approprié pour ce livre. Alors peut-être ai-je menti en disant que tout dans un ordinateur est fait de portes NAND ? Eh bien, pas vraiment, car les DAC et ADC ne sont utilisés que dans certains types de périphériques, pas

dans l'ordinateur lui-même.

Divulgation complète

Nous avons construit un très petit ordinateur ici. Il s'agit du plus petit ordinateur qui puisse être inventé qui fasse tout le nécessaire pour être digne du nom d'ordinateur.

Je ne pense pas que quiconque ait construit un si petit ordinateur depuis 1952 environ, et personne n'a jamais construit cet ordinateur dans le monde réel.

Si jamais un vrai concepteur informatique lisait ce livre, je suis sûr qu'il s'arracherait les cheveux sur toutes les opportunités qui ont été manquées ici pour faire une meilleure machine. Mais encore une fois, l'objectif a été d'illustrer les principes informatiques aussi simplement que possible.

Il s'agit d'un ordinateur huit bits. Cela signifie que les registres du processeur sont huit bits, le bus est huit bits, et dans cette machine, même le registre d'adresse mémoire est huit bits.

Avec la plupart des ordinateurs qui sont réellement construits, alors que les octets individuels dans la RAM restent 8 bits, tout le reste est étendu à 16 bits, 32 bits ou 64 bits ou une combinaison de ceux-ci dans différentes parties de la machine.

Notre RAM n'a que 256 octets, ce qui estridicullement petit, mais c'est tout ce que vous pouvez avoir avec un registre d'adresses mémoire à huit bits. Si vous utilisez 16 bits, vous pouvez avoir 65 536 octets de RAM (soit 64 Ko), si vous utilisez 24 bits, vous pouvez avoir 16 Mo, si vous utilisez 32 bits, vous pouvez avoir 4 gigaoctets de RAM.

Les vrais ordinateurs ont des choses que celui-ci n'a pas, mais ils ne sont pas capables de faire des choses que cet ordinateur ne peut pas faire.

Dans notre ordinateur, si vous voulez décaler un octet de trois bits vers la gauche, vous placeriez trois instructions de décalage vers la gauche dans votre programme. Dans la plupart des vrais ordinateurs, ils ont des décaleurs qui décaleront n'importe quel nombre de bits dans une instruction. Mais le résultat est le même, votre octet finit par avoir le même aspect dans les deux cas, le véritable ordinateur fait simplement le travail plus rapidement.

Dans notre ordinateur, l'additionneur peut additionner deux nombres à huit bits. Si vous voulez ajouter des nombres de 16 bits, vous devez

utiliser un logiciel pour le faire. Dans la plupart des ordinateurs, l'additionneur peut ajouter des nombres de 16 ou 32 bits en une seule instruction.

Encore une fois, les résultats sont les mêmes, l'un est juste plus rapide que l'autre.

Le moteur pas à pas de notre ordinateur est une simplification de quelque chose que la plupart des ordinateurs ont, appelé une "machine d'état". Les machines à états fournissent des étapes, mais démarrent l'instruction suivante dès que possible, font ce qui est nécessaire pour un système d'interruption, peuvent créer des instructions plus complexes, etc. le terme « pas à pas ». Alors oui, notre ordinateur est un ordinateur simple, petit et relativement lent, mais il peut faire tout ce que peuvent faire des machines plus compliquées. Les éléments qui agrandissent une machine plus grande sont conçus pour faire le travail plus rapidement, le faire en moins de cycles d'horloge, faire la même tâche avec moins d'instructions, fonctionner sur plusieurs octets en même temps.

Mais la nature de ce que font les machines est exactement la même. Chaque tâche qu'ils peuvent effectuer se résume à décaler, effectuer un AND, un OR, un XOR, un ADDing et un NOTing d'octets. Il n'y a pas d'autres types d'opérations fantaisistes qui ont été omis de ce livre.

Dans une machine plus grande, vous pouvez faire des additions, des soustractions, des multiplications et des divisions en une seule instruction.

C'est parce qu'ils ont un grand nombre de portes disposées en choses comme un «multiplicateur matériel». Il n'y a aucune raison de vous montrer les détails de la construction de l'un d'entre eux, c'est un travail très compliqué pour les quelques personnes qui ont besoin d'en construire un. C'est compréhensible, et tout se résume finalement aux portes NAND comme tout le reste. Mais nous avons vu comment faire toutes les opérations mathématiques avec juste un additionneur, un shifter, PAS des portes et quelques logiciels. Le multiplicateur matériel y parvient plus rapidement, mais les résultats sont exactement les mêmes.

Les plus grosses machines ont plus de registres, les registres sont chacun de plusieurs octets, ils ont des additionneurs qui peuvent additionner trois nombres en même temps, mais les instructions se résument toujours aux mêmes opérations simples. Votre compréhension des ordinateurs n'est pas petite parce que nous avons examiné un petit ordinateur.

Philosophie

Pourquoi avons-nous un chapitre intitulé « Philosophie » dans un livre sur les ordinateurs ? La seule chose dans ce livre qui se rapproche même d'une question philosophique est son titre, "Mais comment le sait-il?" Nous tenterons de répondre à cette question un peu plus tard.

Ce livre a été sur les ordinateurs que nous avons aujourd'hui. Mais qu'en est-il de l'avenir ? Alors que les ordinateurs et les logiciels continuent de progresser, dans combien de temps, voire jamais, le jour viendra-t-il où des robots informatisés parlants et ambulants apparaîtront et agiront comme des humains ? Viendra-t-il un jour où nous devrons décider de donner ou non à ces robots les mêmes droits légaux qu'aux personnes ? Les ordinateurs finiront-ils par conquérir le monde et remplaceront-ils complètement les gens ?

Pour répondre à ce genre de questions, les gens se réfèrent souvent à une question majeure en suspens dans le domaine de la philosophie depuis de nombreuses années.

La question est de savoir si l'homme est composé uniquement du corps structurel que nous pouvons voir et disséquer, ou s'il existe une composante spirituelle intégrale de chaque être humain qui explique les qualités de conscience, d'amour, d'honneur, de bonheur, de douleur, etc.

Cette question est bien au-delà de la portée de ce livre, et il reste une réponse peu convaincante malgré de nombreux livres faisant valoir chaque point de vue. Il y a des scientifiques qui disent que nous sommes sur la bonne voie pour construire des ordinateurs conscients, et cela arrivera. Il y a des gens dans les sciences humaines qui disent que c'est impossible parce qu'on ne peut pas fabriquer un esprit. Chaque camp a été incapable d'influencer l'autre.

Si nous définissons le cerveau comme ce drôle de morceau de viande grise entouré par le crâne, et définissons l'esprit comme tout ce qui est responsable de la conscience, de la mémoire, de la créativité, de la pensée et de tout ce que nous remarquons dans nos têtes, alors nous pouvons reformuler la grande question philosophique comme suit : "Le cerveau et l'esprit sont-ils une seule et même chose ?"

Ensuite, en ce qui concerne la question de la construction d'un robot humain convaincant, il y aurait deux

possibilités.

Si le cerveau et l'esprit sont la même chose, vous ne pourriez peut-être pas construire une personne synthétique aujourd'hui, mais avec le temps, vous pourriez éventuellement comprendre chaque structure et fonction du cerveau et construire quelque chose d'égale complexité qui généreraient véritable conscience, et cela devrait vraiment agir comme n'importe quelle autre personne.

Si le cerveau et l'esprit ne sont pas la même chose, alors construire un copain robot consistera toujours à simuler l'humanité, et non à construire quelque chose de qualité et de valeur égales.

Reformuler la question ne facilite pas la réponse, mais cette idée de séparer ce que nous savons des esprits de ce que nous savons des cerveaux peut être utile.

Au début, on s'est dit qu'on allait montrer comment fonctionnent les ordinateurs pour voir ce qu'ils sont capables de faire, et aussi ce qu'ils ne sont pas capables de faire. Nous allons prendre ce que nous savons sur les cerveaux et ce que nous savons sur les esprits et comparer chacun individuellement à nos nouvelles connaissances sur les ordinateurs. Ce faisant, nous pouvons rechercher des différences et des similitudes, et nous pouvons être en mesure de répondre à quelques questions moins controversées.

Les ordinateurs font certaines choses avec une grande facilité, comme additionner des colonnes de nombres. Un ordinateur peut faire des millions d'ajouts en une seule seconde. L'esprit peut à peine se souvenir de deux nombres en même temps, sans parler de les additionner sans crayon et papier.

L'esprit semble avoir la capacité de regarder et de considérer des quantités relativement importantes de données en même temps. Quand je pense à mon chat préféré, je peux voir à quoi il ressemble, entendre les sons de son ronronnement et de ses miaulements, sentir la douceur de sa fourrure et son poids lorsqu'il est ramassé. Ce sont quelques-unes des façons dont je connais mon animal de compagnie.

Qu'est-ce que cela signifierait pour notre ordinateur de penser à un chat ? Il pourrait avoir des images du chat et des sons du chat encodés dans des fichiers sur un disque rotatif ou dans la RAM. Est

cette réflexion ? Si vous exécutiez les octets de ces fichiers un par un via l'ALU, serait-ce une réflexion ? Si vous mettez l'image sur l'écran, serait-ce penser ?

Si vous jouiez les sons sur les haut-parleurs, serait-ce de la pensée ?

Les sons et les images encodés dans l'ordinateur ne sont que des modèles d'octets qui restent là où ils se trouvent. Ils ne ressemblent à rien et ne sonnent à rien tant qu'ils ne sont pas envoyés aux périphériques pour lesquels ils ont été conçus.

Et s'ils sont envoyés à l'écran et aux haut-parleurs, l'ordinateur ne les voit pas et ne les entend pas. Bien sûr, votre ordinateur pourrait avoir une caméra pointant vers l'écran et un microphone écoutant les sons, mais l'ordinateur ne verrait toujours pas d'image ou n'entendrait pas de son, il collecterait simplement plus de chaînes d'octets très similaires à celles envoyé à l'écran et aux haut-parleurs en premier lieu.

Il pourrait y avoir des programmes qui effectuent des opérations mathématiques sur les fichiers image afin de découvrir des modèles et de stocker les résultats de ces calculs dans d'autres fichiers. Il peut y avoir des fichiers qui relient un fichier image à d'autres fichiers image similaires, et des images à des sons, etc., créant ainsi plus de fichiers.

Mais peu importe la quantité de programmation appliquée aux fichiers image, il y a quelque chose que l'esprit peut faire pour lequel l'ordinateur n'a tout simplement aucune possibilité.

L'esprit peut considérer l'ensemble de quelque chose en même temps. Vous pouvez penser à l'ensemble du chat d'un coup. C'est un peu comme la différence entre le film et l'écran de télévision. Le film a des images entières, l'écran du téléviseur n'a qu'un pixel à la fois.

Vous pourriez dire que votre esprit travaille si vite que vous ne remarquez pas les détails, il s'intègre dans un tout, tout comme les pixels s'intègrent dans une image entière. Mais qu'est-ce que l'intégration ? Et quand c'est intégré, qu'est-ce que c'est et où est-ce que c'est ? Et qu'en est-il de l'ensemble intégré ?

Nous venons de voir tout ce qu'il y a dans un ordinateur. L'ordinateur se déplace d'un octet à la fois sur le bus. La chose la plus fantaisiste qu'il fasse est d'ajouter deux octets en un seul.

Tout le reste qu'il « fait » n'est rien de plus qu'un simple stockage d'octets. Un octet stocké ne fait rien d'autre que maintenir son propre réglage actuel.

Un ordinateur n'a tout simplement aucune fonctionnalité permettant d'intégrer les éléments d'une image dans quoi que ce soit d'autre, nulle part où stocker ce quelque chose d'autre et rien avec quoi le regarder.

Je ne dis pas que quelque chose ne pourrait pas être construit qui remplirait ces fonctions, je dis simplement que les ordinateurs tels que nous les connaissons aujourd'hui n'incluent actuellement aucun appareil de ce type.

Voici une autre question. Si un cerveau fonctionne comme un ordinateur, il doit avoir un programme pour que le processeur fonctionne. D'où viendrait ce programme ?

Bien que le cerveau ait des billions de cellules, le corps humain tout entier commence avec un ovule fécondé. Ainsi, tout programme du cerveau devrait être présent dans cette cellule unique, vraisemblablement dans l'ADN.

Les scientifiques ont maintenant décodé toute la séquence d'ADN de l'homme. L'ADN est intéressant en ce sens qu'il s'agit d'une longue chaîne de seulement quatre types de choses. C'est numérique ! Beaucoup de morceaux de cette chaîne sont utilisés pour faire des réactions chimiques pour fabriquer des protéines, etc. mais la majorité d'entre elles est appelée « ADN indésirable » parce que personne ne sait à quoi elle sert. Mais même si vous considérez que l'intégralité de l'ADN est consacrée aux logiciels informatiques, il pourrait y avoir environ un milliard d'instructions dans ce programme. Cela fait beaucoup de logiciels, mais l'ordinateur domestique moyen a probablement autant de logiciels chargés sur son disque dur, et ce ne serait pas suffisant pour faire fonctionner un être humain.

Certains ont dit que l'ordinateur humain se programme lui-même. En tant que programmeur moi-même, je ne peux tout simplement pas imaginer comment cela fonctionnerait. S'il est vrai qu'un programme peut accumuler des données et modifier son fonctionnement en fonction des données collectées, ce n'est pas la même chose que d'écrire un nouveau programme. Si jamais quelqu'un écrit ce programme qui peut écrire n'importe quel nouveau programme nécessaire, il y aura un grand nombre de programmeurs informatiques mis au chômage pour toujours.

Ensuite, il y a les types d'erreurs commises par les ordinateurs par rapport à celles commises par les gens. Si un ordinateur est bloqué dans une boucle, il semble s'être complètement arrêté.

Avez-vous déjà vu une personne marchant dans la rue s'arrêter soudainement de travailler ? Toutes les fonctions cessent simplement. La personne tomberait juste jusqu'à ce que son ordinateur redémarre d'une manière ou d'une autre. Les gens s'effondrent de temps en temps, mais c'est généralement parce qu'une autre partie s'est cassée, comme une crise cardiaque, et vous pouvez voir la personne reconnaître la douleur au fur et à mesure qu'elle la fait tomber. Mais si l'ordinateur humain restait coincé dans une boucle, il y aurait une perte de conscience instantanée et le corps tomberait complètement mou sans aucune lutte. Je n'ai jamais vu cela, mais si le cerveau fonctionnait comme un ordinateur, on s'attendrait à le voir assez régulièrement.

Ensuite, il y a la question de la vitesse. Comme nous l'avons vu, un simple ordinateur peut faire un milliard de choses en une seconde.

En ce qui concerne le cerveau, il a des nerfs qui ressemblent aux fils des ordinateurs. Les nerfs peuvent également transporter l'électricité d'un endroit à l'autre. Dans un ordinateur, les fils sortent des portes et entrent dans d'autres portes. Dans le cerveau, les nerfs sont reliés entre eux par des « synapses ».

Ces synapses sont des espaces entre les nerfs où l'électricité d'un nerf crée une réaction chimique, qui amène ensuite le nerf suivant à créer sa propre électricité. Ces réactions chimiques sont douloureusement lentes.

Personne n'a montré que ces nerfs sont connectés comme les fils d'un ordinateur, mais leur manque de vitesse fait qu'il est très peu probable que cela fasse beaucoup de bien même si les connexions étaient similaires. Une fois que l'électricité a parcouru rapidement la cellule nerveuse, elle atteint la synapse, où la réaction chimique prend environ un cinq centième de seconde pour se terminer. Cela signifie que notre simple ordinateur construit à partir de portes NAND pourrait faire deux millions de choses en même temps qu'une seule chose pourrait être faite par un ordinateur construit à partir de nerfs et de synapses.

Un autre domaine où la différence entre l'esprit et les ordinateurs est assez évidente est celui de la reconnaissance des visages. L'esprit y est très bien. Si vous entrez dans une fête avec cinquante personnes présentes, vous saurez en quelques secondes si vous faites partie d'un

groupe d'amis ou d'inconnus. De nombreuses recherches ont été effectuées sur la façon dont les gens accomplissent cet exploit, et de nombreuses informations très intéressantes ont été découvertes.

Il y a aussi beaucoup de spéculations et il existe de nombreuses théories fascinantes sur les principes et mécanismes sous-jacents. Mais les structures et les fonctions complètes et exactes n'ont pas été découvertes.

Si vous donnez à un ordinateur un fichier image d'une personne, puis lui donnez à nouveau le même fichier, il peut comparer les deux fichiers octet par octet et voir que chaque octet dans un fichier est exactement égal à l'octet correspondant dans l'autre fichier. Mais si vous donnez à l'ordinateur deux photos de la même personne qui ont été prises à des moments différents, ou sous des angles différents, ou avec un éclairage différent, ou à des âges différents, alors les octets des deux fichiers ne correspondront pas octet par octet. Pour l'ordinateur, déterminer que ces deux fichiers représentent la même personne est une tâche énorme. Il doit exécuter des programmes très complexes qui exécutent des fonctions mathématiques avancées sur les fichiers pour y trouver des modèles, puis déterminer à quoi ces modèles pourraient ressembler sous différents angles, puis comparer ces choses à tous les autres visages qu'il a jamais stockés sur son disque, choisissez la correspondance la plus proche, puis déterminez si elle est suffisamment proche pour être la personne ou simplement quelqu'un qui lui ressemble.

Le fait est que les ordinateurs ont une méthode de traitement des images basée sur les principes sur lesquels fonctionnent les ordinateurs. L'utilisation de ces seuls principes n'a pas encore produit d'ordinateurs ou de logiciels capables de reconnaître un visage avec la rapidité et la précision de n'importe quelle personne ordinaire.

La reconnaissance vocale par les ordinateurs est une autre technologie qui a parcouru un long chemin, mais qui a encore beaucoup à faire pour rivaliser avec ce que l'esprit fait facilement.

Donc, en comparant un ordinateur à un cerveau, il semble tout simplement peu probable qu'ils fonctionnent sur les mêmes principes. Le cerveau est très lent, il n'y a aucun endroit où trouver le logiciel pour l'exécuter, et nous ne voyons pas les types de problèmes auxquels nous nous attendrions avec des erreurs de logiciel informatique.

En comparant un ordinateur à l'esprit, l'ordinateur est

beaucoup mieux en mathématiques, mais l'esprit est meilleur pour gérer les visages et les voix, et peut contempler l'intégralité d'une entité qu'il a déjà expérimentée.

Les livres et les films de science-fiction regorgent de machines qui lisent les pensées ou y implantent des idées, des vaisseaux spatiaux avec des ordinateurs parlants intégrés et des robots et androïdes réalistes. Ces machines ont des capacités variables et certaines des intrigues traitent du robot aux prises avec la conscience, la réalisation de soi, les émotions, etc. Ces machines semblent ne pas être complètes car ce ne sont que des machines et veulent désespérément devenir pleinement humaines. C'est en quelque sorte une version adulte du classique pour enfants "Pinocchio", l'histoire d'une marionnette qui veut devenir un vrai garçon.

Mais serait-il possible de construire de telles machines avec une version considérablement étendue de la technologie que nous avons utilisée pour construire notre simple ordinateur ?

L'optimisme est une bonne chose, et il ne faut pas l'écraser, mais un problème ne sera pas susceptible d'être résolu si vous utilisez une méthodologie ou une technologie qui n'est pas à la hauteur de ce problème. Dans le domaine de la médecine, certaines maladies ont été anéanties par les antibiotiques, d'autres peuvent être prévenues par des inoculations, mais d'autres continuent de tourmenter l'humanité malgré les meilleurs soins et des décennies de recherche. Et ne regardons même pas

dans des sujets comme la politique. Peut-être que plus de temps est tout ce qu'il faut, mais vous devez également envisager la possibilité que ces problèmes soient insolubles ou que la recherche ait cherché la réponse aux mauvais endroits.

Par exemple, de nombreuses visions du futur ont inclus des personnes voyageant dans des voitures volantes. En fait, plusieurs types de voitures volantes ont été construits. Mais ils sont chers, inefficaces, bruyants et très dangereux.

Ils fonctionnent sur les mêmes principes de base que les hélicoptères.

Si deux voitures volantes ont un accident mineur, tout le monde mourra lorsque les deux voitures s'écraseront sur la Terre. Ainsi, la technologie aéronautique d'aujourd'hui ne se traduira tout simplement pas par une voiture volante satisfaisante. À moins et jusqu'à ce que quelqu'un invente un dispositif anti-gravité bon marché et fiable, il n'y aura pas de marché de masse pour les voitures volantes et le trafic sur les routes ne sera pas soulagé.

Si vous voulez construire une machine qui fonctionne exactement comme une personne, la meilleure façon de le faire serait certainement de découvrir comment la personne travaille, puis de construire une machine qui fonctionne sur les mêmes principes, qui a des pièces qui font les mêmes choses, et est câblé de la même manière qu'une personne.

Lorsque Thomas Edison a inventé le phonographe, il traitait du sujet du son. Le son est une vibration de l'air. Il inventa donc un appareil qui captait les vibrations de l'air et les transformait en une rainure vibrante à la surface d'un cylindre de cire. Le son pourrait alors être recréé en transférant les vibrations dans la rainure dans l'air. Le fait est que pour recréer le son, il a découvert comment le son fonctionnait, puis a fabriqué une machine qui fonctionnait sur le même principe. Le son est une vibration, le sillon d'un phonographe est une vibration.

Beaucoup de recherches ont été faites sur le sujet de ce qui motive les gens. De nombreuses recherches ont été menées sur la manière de faire en sorte que les ordinateurs fassent ce que les gens font. Beaucoup de choses ont été découvertes et beaucoup de choses ont été inventées. Je ne veux pas minimiser le travail effectué ou les résultats obtenus dans ces domaines.

Mais il y a beaucoup de choses qui n'ont pas encore été découvertes ou inventées.

De nombreux cerveaux morts ont été disséqués et leurs parties ont été étudiées et classées. Le cerveau contient des cellules nerveuses qui transportent l'électricité d'un endroit à un autre. C'est une similitude entre les cerveaux et les ordinateurs. Mais la recherche sur le fonctionnement réel des cerveaux humains vivants est nécessairement limitée. La plupart des observations ont été faites lors d'interventions chirurgicales rendues nécessaires par un accident ou une maladie. De nombreuses observations ont été faites sur les changements de comportement après qu'une blessure ou une maladie ait désactivé certaines parties du cerveau. A partir de ces recherches, il a été possible d'associer certaines fonctions à certaines zones du cerveau.

Mais personne n'a découvert de bus, d'horloge, de registres, d'ALU ou de RAM. Le mécanisme exact de la mémoire dans le cerveau reste un mystère. Il a été démontré que les nerfs développent de nouvelles connexions au fil du temps, et on suppose que c'est le mécanisme de l'apprentissage, mais personne n'a été

capable de dire que ce nerf particulier remplit cette fonction exacte, comme nous pouvons le faire avec les fils individuels d'un ordinateur.

Tout ce qui entre dans un ordinateur est transformé en un code ou un autre. Le clavier génère un octet ASCII par frappe, un microphone génère 44 100 nombres binaires par seconde, une caméra couleur génère trois nombres binaires par pixel, 30 fois par seconde, etc. Personne n'a isolé l'utilisation de codes tels que l'ASCII, les nombres binaires, les polices ou un code d'instruction dans le cerveau. Ils sont peut-être là, mais ils n'ont pas été isolés. Personne n'a tracé une pensée ou localisé un souvenir de la même manière que nous pourrions suivre le fonctionnement d'un programme dans un ordinateur.

Il est largement admis que le cerveau fonctionne d'une manière beaucoup plus dispersée qu'un seul ordinateur, qu'il existe des milliers ou des milliards d'éléments informatiques qui coopèrent et partagent le travail. Mais de tels éléments n'ont pas encore été localisés. Dans le monde de l'informatique, cette idée est appelée "traitement parallèle" et des ordinateurs avec des dizaines ou des centaines de processeurs ont été construits. Mais ces ordinateurs n'ont toujours pas abouti à un substitut humain.

Considérez tout cela comme un puzzle. La façon dont les gens travaillent est une face du puzzle. Faire en sorte que les ordinateurs fassent des choses que les gens font est l'autre côté du puzzle. Les pièces du puzzle sont assemblées des deux côtés. Le problème est qu'au fur et à mesure que des progrès sont réalisés des deux côtés, il semble de plus en plus qu'il s'agit de deux puzzles différents, ils ne se rejoignent pas au milieu. Ils ne convergent pas en une seule image.

Les chercheurs sont très conscients de ces développements. Mais quand il s'agit de culture pop, les gens entendent parler de nouvelles inventions tout le temps et voient l'avenir dépeint dans les films de science-fiction, et la conclusion logique semble être que la recherche continuera à résoudre les problèmes un par un jusqu'à ce que dans 10 ou 20 ou 30 ans nous aurons nos amis électro-mécaniciens. Au siècle dernier, nous avons conquis l'électricité, le vol, les voyages dans l'espace, la chimie, l'énergie nucléaire, etc. Alors pourquoi pas le cerveau et/ou l'esprit ?

La recherche, cependant, en est encore au stade où chaque fois qu'une nouvelle réponse est trouvée, elle crée plus d'une nouvelle question.

Il semble donc que, quelle que soit la façon dont nous le regardons, ni le cerveau ni l'esprit ne fonctionnent sur les mêmes principes que les ordinateurs tels que nous les connaissons. Je dis « tels que nous les connaissons » parce qu'un autre type d'ordinateur pourrait être inventé à l'avenir. Mais tous les ordinateurs que nous avons aujourd'hui relèvent de la définition des « ordinateurs numériques à programme stocké », et tous les principes sur lesquels ils fonctionnent ont été présentés dans ce livre.

Pourtant, rien de tout cela ne « prouve » qu'un humain synthétique ne pourrait jamais être construit, cela signifie seulement que les principes informatiques tels que présentés dans ce livre ne sont pas suffisants pour le travail. Un type d'appareil complètement différent qui fonctionne selon un ensemble de principes complètement différent pourrait être en mesure de le faire. Mais nous ne pouvons pas commenter un tel appareil jusqu'à ce que quelqu'un en invente un.

Pour en revenir à une question plus simple, vous souvenez-vous de Joe et de la bouteille Thermos ? Il pensait que le Thermos avait une sorte de capteur de température, et un radiateur et un refroidisseur à l'intérieur. Mais même s'il avait eu toutes ces machines, il ne « saurait » toujours pas quoi faire, ce serait juste un appareil mécanique qui allumait le chauffage ou le refroidisseur en fonction de la température de la boisson qui y était placée.

Une paire de ciseaux est un appareil qui remplit une fonction lorsqu'il est conçu pour le faire. Vous mettez un doigt et un pouce dans les trous et pressez. Les lames à l'autre extrémité des ciseaux se déplacent ensemble et coupent du papier ou du tissu ou tout ce que vous avez placé sur leur chemin. Est-ce que les ciseaux « savent » comment découper des formes dans du papier ou comment faire une robe avec du tissu ? Bien sûr que non, ils font juste ce qu'on leur dit.

De même, les portes NAND ne "savent" pas ce qu'elles font, elles réagissent simplement à l'électricité ou à l'absence d'électricité placée sur leurs entrées. Si une porte ne sait rien, alors peu importe combien d'entre elles vous connectez ensemble, si l'une d'elles connaît absolument zéro, un million d'entre elles connaîtra également zéro.

Nous utilisons beaucoup de mots qui donnent des caractéristiques humaines à nos ordinateurs. On dit qu'il « sait » des choses. Nous disons qu'il "se souvient" des choses. Nous disons qu'il "voit" et "comprend". Même quelque chose d'aussi simple qu'un adaptateur de périphérique "écoute" son adresse pour qu'elle apparaisse sur l'E/S

bus, ou une instruction de saut « décide » quoi faire. Il n'y a rien de mal à cela tant que nous connaissons la vérité sur la question.

Maintenant que nous savons ce qu'il y a dans un ordinateur et comment il fonctionne, je pense qu'il est assez évident que la réponse à la question "Mais comment sait-il ?" est simplement "Il ne sait rien!"