IUT - Département Informatique

ASR2-Système

Ordinateurs et Systèmes d'Exploitation

SYS ORDI

Table des matières

1 Le premier ordinateur

Les ordinateurs tels que nous les connaissons sont des objets qui s'incrivent dans une suite d'inventions et de combinaisons de technologies diverses.

Si on définit l'ordinateur comme

> un appareil électronique qui fait des calculs en suivant les instructions d'un programme enregistré

le premier ordinateur construit est très probablement le *Manchester Small Scale Experimental Machine (SSEM)*[1],

qui a tourné pour le première fois le 21 juin 1948. Il avait été réalisé par Tom Kilburn et Geoff Tootill, dans l'équipe de Freddie Williams, professeur d'électrotechnique à l'Université de Manchester.

Voir le reportage tourné par la BBC en 1948, sur `http://news.bbc.co.uk/2/hi/technology/7465115.stm`

1.1 Les calculateurs électroniques

Les calculateurs électroniques à programme existaient déjà depuis une quizaine d'années, mais leurs programmes n'étaient pas enregistrés en mémoire. Ils étaient soit externes (bande ou carte perforées), soit figés. Par exemple sur l'ENIAC (1946), des interrupteurs à tourner dans tableau de connexion pour réaliser les 0 et les 1 d'une mémoire morte. Cette manière de faire dérivait assez naturellement des machines à traiter les cartes perforées, utilisées depuis la fin du XIXe siècle. [2]

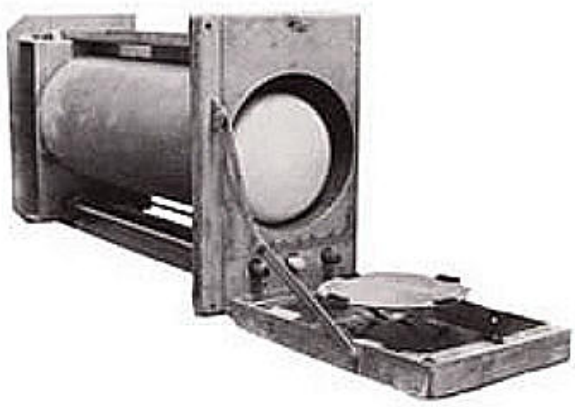
En réalité le SSEM était un prototype destiné à tester l'utilisabilité d'une mémoire à tube cathodique inventée par

1. surnommé "Baby", voir `http://www.computer50.org/mark1/new.baby.html`
2. Voir l'article de Wikipedia consacré à la mécanographie

F. Williams.

1.2 Mémoire à tube de Williams-Kilburn

Cette innovation utilisait un tube d'oscilloscope standard, dans lequel un faisceau d'électrons permettrait d'allumer des points de phosphore sur l'écran, avec une certaine rémanence. Le tube de Williams-Kilburn utilise la propriété suivante : quand le faisceau bombarde un point de l'écran, des électrons secondaires sont éjectés par le phosphore, en quantité différente selon que le point est ou non déjà allumé. En mesurant la tension sur une plaque métallique devant l'écran, on peut connaître l'état du point.



En 1947, l'équipe de Williams avait réussi à stocker 2048 bits sur un écran pendant des heures, ce qui promettait une technologie de mémoire rapide, bon marché, basée sur des composants standards, destinée aux calculateurs.

L'idée est donc venue assez naturellement de fabriquer un calculateur simple avec une mémoire à tube pour en tester la fiabilité dans une machine qui effectue plusieurs milliers de lectures/écritures par seconde. Jusque là, le tube avait été testé en addressant les bits par un jeu d'interrupteurs manuels...

1.3 Architecture et programmation

L'architecture du SSEM est très simple. En termes modernes, c'est une machine 32 bits, avec une mémoire de 32 mots de 32 bits (1024 bits stockés sur un tube), extensible à 8192.

Les calculs se font en nombre entiers en notation complément à deux.

Deux tubes étaient utilisés pour les registres spéciaux :

- l'un pour l'accumulateur A (32 bits) sur lequel se font les opérations
- l'autre pour CI (control instruction) qui contient d'adresse de l'instruction en cours, et PI (present instruction), l'instruction elle-même

Un dernier tube (sans plaque) dupliquait le premier, permettant de voir les bits en mémoire [3].

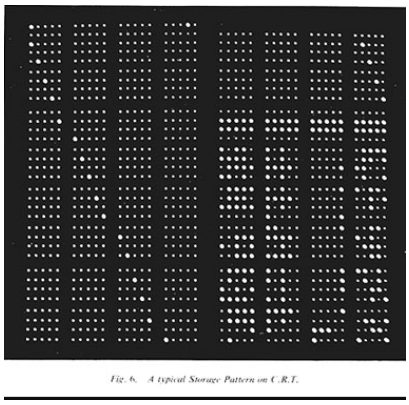


Fig. 6. A typical Storage Pattern on C.R.T.

La mémoire du SSEM

Le jeu d'instructions était réduit à 7 instructions d'un format unique : 3 bits pour le code opération, et 13 bits pour l'adresse S de l'opérande. les 16 derniers bits étaient inutilisés. Les 7 opérations étaient

- `LDN S` (load negative) `A = - Mem[S]`, qui charge dans l'accumulateur l'opposé du contenu d'un mot mémoire
- `SUB S` (subtract) `A = A - Mem[S]`, qui soustrait le contenu d'un mot
- `STO S` (store) `Mem[S] = A`, qui copie en mémoire le contenu de l'accumulateur
- `CMP` (compare) `If A < 0, CI = CI + 1`, qui saute l'instruction suivante si l'accumulateur est négatif,
- `JMP S` (jump) `CI = Mem[S] + 1`, qui provoque un saut indirect, à l'adresse contenue dans un mot de la mémoire.
- `JRP S` (jump relative) `CI = CI + Mem[S] + 1`, pour un saut indirect relatif.
- `HLT` (halt) qui arrête l'ordinateur.

Ce jeu d'instruction a été choisi parce qu'il était réalisable avec un minimum de circuits électroniques. Il ne simplifie évidemment pas la vie du programmeur. Par exemple, pour additionner deux nombres x et y situés aux adresses 20 et 21, il faut 4 opérations en passant par une variable temporaire (d'adresse 22)

```
0   LDN  20  ; A contient -x
1   SUB  21  ; A contient -x-y
2   STO  22  ; Mem[22] contient -x-y
3   LDN  22  ; A contient -(-x-y) = x+y
```

Voici un exemple plus complexe, écrit en utilisant des adresses symboliques : le calcul du maximum de deux nombres X et Y et le rangement dans Z.

```
# calculer la différence

0   LDN   X   ; A = -x
1   STO   TMP ; TMP = -x
2   LDN   TMP   ; A = x
3   SUB   Y

# selon la différence, charger -X ou -Y
# dans l'accumulateur
```

3. les bits de poids fort sont à droite, contrairement à la notaiton habituelle des nombres en binaire

```
4   CMP
5   JMP   I8 ; si A positif
6   LDN   Y
7   JMP   I9
8   LDN   X

# et envoyer l'opposé de l'accumulateur dans Z

9   STO   TMP
10  LDN   TMP
11  STO   Z
12  HLT

# Les variables

13 X   = 42
14 Y   = 15
15 TMP = 0
16 Z   = 0

# les adresses de saut
17 i8    = 7
18 i9    = 8
```

Remarque : pour aller à l'instruction 9, l'instruction 7 charge le mot d'adresse 17 dans le CI, auquel il ajoute 1 comme après toute instruction. C'est pourquoi le mot 17 contient 8, l'adresse qui précède celle de l'endroit où le programme doit se poursuivre.

1.4 Une démontration probante

Le programme précédent suffit à occuper plus de la moitié de la mémoire disponible sur le SSEM, qui n'a évidemment jamais servi à faire des calculs très complexes. Le clou du spectacle était un programme de 17 instructions [4] pour trouver le plus grand diviseur propre d'un nombre N, en essayant de le diviser successivement par N-1, N-2 etc., la division étant elle-même réalisée par soustraction successives.

Kilburn et Williams devant la console du SSEM

Il a fallu 52 minutes de calcul (et 3,5 millions d'opérations) pour établir que le plus grand facteur propre de 2^{18} était 2^{17}. Ce que tout le monde savait déjà évidemment. Sur l'écran phosphorescent, une puissance de deux est facile à lire : un point allumé sur une ligne éteinte.

Peu importe : l'objectif était de montrer que la mémoire était fiable : peu importe les calculs du moment que le résultat est correct après des millions d'opérations.

1.5 Les suites

Un vrai ordinateur est sorti de ces travaux, *Manchester Automatic Digital Machine* (MADM), opérationnel en avril 1949, et qui a donné naissance aux machines du constructeur Ferranti.

Quant aux tubes de Williams, ils ont été utilisés comme mémoires dans quelques ordinateurs célèbres (UNIVAC 1103, Whirlwind, IBM 701, IBM 702 ...) avant d'être rapidement supplantés par les mémoires à tores de ferrite, qui ont dominé le marché pendant 20 ans de 1955 à 1975, avant d'être remplacées par les mémoires à semi-conducteurs que nous utilisons aujourd'hui.

2 Structure d'un ordinateur simple

Schématiqument, un ordinateur est composé
- d'un **processeur**, circuit électronique capable d'exécuter une à une (mais très vite) les instructions d'un programme
- d'une **mémoire centrale**, circuit qui sert à mémoriser les données et les programmes pendant leur exécution
- des **périphériques** : imprimante, carte réseau, carte graphique, disque dur etc. reliés par des contrôleurs d'interface.

Le rôle du Baby étant de tester la fiabilité des mémoires à tubes de Williams-Kilburn, il était dépourvu de périphérique.

Dans ce chapitre, nous présentons ces divers éléments de façon très simplifiée.

2.1 La mémoire

Différentes technologies ont été utilisées pour réaliser les mémoires des ordinateurs : bascules bistables à base de tubes (triodes), mémoires à tores de ferrite, à transistors, circuits intégrés etc.

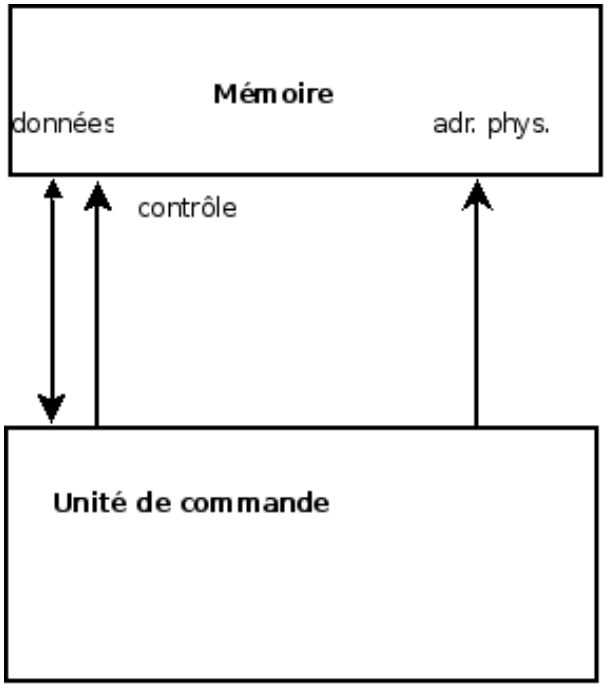
Indépendamment des technologies, la mémoire est un organe qui a pour fonction de stocker et restituer des "mots binaires" repérés par une `adresse`.

Les mots sont de taille fixe [5], par exemple l'ATLAS (réalisé en 1962 conjointement par l'université de Manchester, Ferranti et Plessey) avait une mémoire de 16384 mots de 48 bits. Les micro-processeurs des années 70 étaient souvent des machines à octets (mot = 8 bits), de nos jours ce sont des mots de 32 ou 64 bits.

La mémoire communique avec le reste de l'ordinateur par 3 bus (groupes de fils)

4. La légende de l'université de Manchester dit que c'est le seul programme que le professeur Williams ait jamais écrit.
5. Il y a eu bien sûr quelques exceptions, qui ont été des échecs. Dans l'histoire des ordinateurs, beaucoup de choses ont été essayées.

- le **bus de contrôle** (il faudrait dire bus de commande), qui indique à la mémoire l'opération que l'on veut effectuer : lecture ou écriture ;
- le **bus de données**, bidirectionnel, qui sert à émettre et recevoir les mots ;
- le **bus d'adresses**, qui indique à la mémoire l'adresse concernée.



La mémoire et ses trois bus

Opérations :

- lecture : pour consulter l'adresse A en mémoire, le processeur place le nombre A sur le bus d'adressses, et envoie le signal de contrôle "lecture". Après un petit délai de réponse, le contenu du mot d'adresse A est présent sur le bus de données.
- écriture : pour envoyer un mot M à l'adresse A, le processeur place A sur le bus d'adresses, M sur le bus de données et active l'ordre d'écriture.

Remarque : sur certaines machines (c'était le cas du processeur 8088 qui équipait les premiers PC d'IBM [6]), les bus de données et d'adresses sont *multiplexés*, ils partagent des fils. L'avantage est de minimiser le nombre de connexions entre circuits intégrés (et du nombre de pattes), l'inconvénient est que les données et les adresses ne sont pas être transmis en même temps, ce qui se fait au détriment des performances. Un signal de commande supplémentaire précise si l'information qui circule est une adresse ou une donnée.

2.2 Le processeur

Un processeur est un dispositif électronique relativement simple, composé de circuits logiques divers. Il communique avec la mémoire (voir plus haut) et les périphériques par des bus de donées, d'adresse et de commande.

Son rôle est d'exécuter, les unes après les autres, des instructions qui sont stockées en mémoire.

2.2.1 Instructions et registres

Le **compteur de programme** [7] est un registre [8] qui contient l'adresse de la prochaine instruction à exécuter. C'est un *compteur* parce que, la plupart du temps, on va lui ajouter 1 pour passer à l'instruction suivante.

La première action du processeur est de lire en mémoire le mot qui contient cette instruction, et de la placer dans un **registre d'instruction**, où il sera décodé. [9].

Par exemple, on aura peut être lu le mot de 32 bits **00011000010000110000000000101010** qui, sur un PowerPC 32 bits, se décompose en **001110 00010 00011 00000000 000101010** ce qui représente

- les 6 premiers bits : le code de l'opération "ajouter une constante"
- un numéro de registre destination (registre 2) sur 5 bits
- un numéro de registre source (registre 3)
- une constante (42) codée sur 16 bits

et qui signifie : ajoutez au registre de travail numéro 3 la valeur 42, et placez le résultat dans le registre 2, ce qu'on écrirait en *langage d'assemblage*

```
addi  2,3,42
```

L'exécution de l'opération ci-dessus fera appel à différents autres circuits

- les **registres généraux**, qui stockent des valeurs intermédiaires (il y en a 32 sur le PowerPC). C'est la généralisation de l'accumulateur A du SSEM.
- une **unité arithmétique**, qui sera ici chargée de s'occuper de l'addition.

Certaines opérations permettront des transferts entre registres et mémoire, par exemple

```
stw  5,0,1234
```

envoie (*store*) le contenu du registre général 5 à l'adresse 1234 de la mémoire.

Il y a également des instructions pour comparer le contenu de registres, et d'autres qui changent le cours de l'exécution si une condition est remplie, en indiquant le numéro de la prochaine instruction à exécuter.

En fait, les instructions de comparaison positionnent des indicateurs booléens dans un **registre de condition** qui mémorisent le résultat de la comparaison (inférieur, supérieur, égal ?)

6. Le processeur 8086 d'INTEL possédait des bus séparés, le 8088 qui en était dérivé était à la fois plus cher à fabriquer (c'est un 8086 avec en plus des circuits de multiplexage/démultiplexage) et moins performant. Ceci dit, le 8086 nécessitait une famille de circuits associés 16 bits (contrôleurs d'interruption, etc), alors que le 8088 pouvait se contenter des circuits 8 bits qui étaient déjà produits en masse pour les microprocesseurs 8080 et 8085 qui équipaient les micro-ordinateurs les plus courants de l'époque (sous CP/M).

7. ou pointeur d'instruction, compteur ordinal...

8. en électronique numérique, un *registre* est un circuit qui mémorise quelques bits d'information. Il est possible d'y charger une valeur, et de la relire ensuite.

9. La terminologie du SSEM les désignait par CI (current instruction) et PI (present instruction)

En fonction de ces indicateurs, les **instructions de branchement conditionnel** incrémentent le compteur de programme, ou lui affectent une autre adresse.

Rappel : sur le SSEM, le bit de signe de l'accumulateur était l'unique indicateur de condition, utilisé par l'instruction `CMP`.

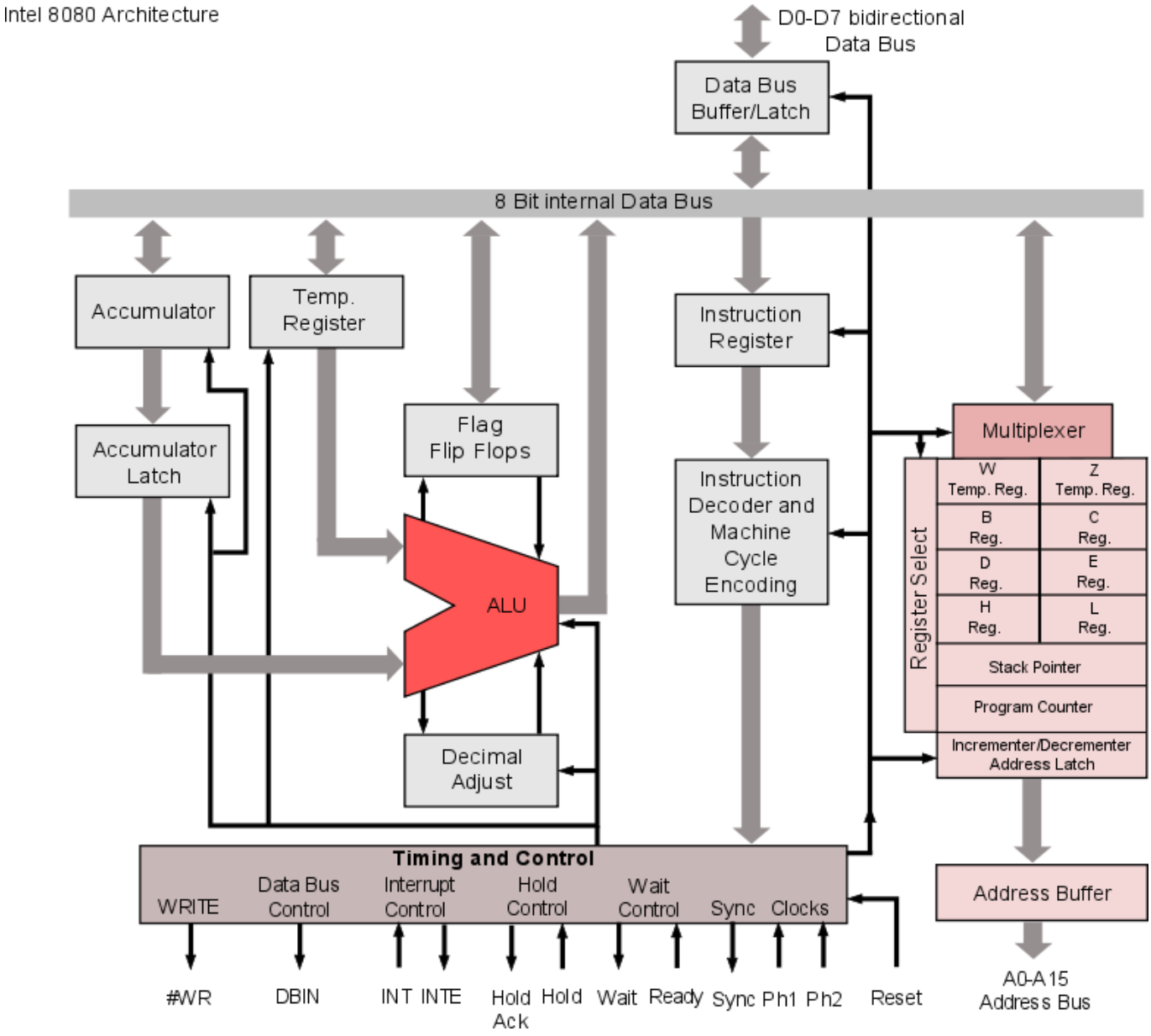
À ces opérations s'ajoutent des **instructions d'entrée-sortie**, permettant le dialogue avec des circuits **contrôleurs de périphériques**, ainsi que des instructions spéciales que nous verrons plus tard.

Tous ces circuits fonctionnent sous le contrôle d'un **séquenceur**, qui enchaîne les différentes étapes :

- envoyer le contenu du PC sur le bus d'adresse, et un ordre de lecture
- copier la valeur présente sur le bus de données dans le registre d'instruction RI, et indiquer la fin de lecture
- décoder l'instruction contenue dans le RI
- l'exécuter. Si il s'agit de "ajouter 42 au registre 3" :
  - envoyer le contenu du registre 3, la valeur 42 et l'ordre d'addition l l'unité arithmétique (circuit de calcul)
  - copier le résultat dans le registre 3
  - ajouter 1 au PC
- et recommencer

2.2.2 Architecture interne d'un processeur

Le schéma ci-dessous montre l'architecture interne du processeur 8080 mis sur le marché par Intel en Avril 1974. C'est le second micro-processeur 8 bit, après le 8008.



On retrouve

- en haut, le bus de données de 8 bits

- en bas à droite le bus d'adresse 16 bits,
- à gauche l'accumulateur A
- au centre l'unité arithmétique et logique (ALU)
- le registre d'instruction
- à droite une "banque de registres", dont le compteur de programme, et des registres de travail (B,C,D ...)

2.2.3 Le modèle du programmeur

Dans un ordinateur, certains registres sont utilisable directement par le programmeur (comme le registre accumulateur par exemple), et d'autres ont un rôle interne (le registre d'instruction).

Ce qu'on appelle **modèle du programmeur**, c'est la partie qui est utilisable directement par le programmeur

- le compteur de programme,
- les registres généraux et spécialisés
- les registres de condition
- les différentes catégories d'instruction
  - chargement/ rangement en mémoire
  - arithmétique
  - logique : et, ou, décalages...
  - tests
  - branchements conditionnels et inconditionnels,
  - ...

Par exemple, dans le 8080, le registre C est utilisable comme opérande d'une instruction, par exemple `MOV A,C` (copie de C dans A), il fait donc partir du modèle contrairement au registre TEMP qui sert d'intermédiaire dans certaines instructions. Par exemple l'instruction `ADD B` (ajouter le contenu du registre B à l'accumulateur) se déroule en deux temps

1. copier B dans TEMP, et A dans le registre "latch" (autre registre temporaire), pour les présenter en entrée de l'UAL,
2. envoyer le résultat de LATCH+TEMP, sortant de l'UAL, dans A.

Exercice 1. Pourquoi y a-t-il deux étapes, et non une seule ?

2.2.4 La programmation en langage machine

Un exemple de programme en pseudo-assembleur vous montre le niveau de détail auquel il faut descendre quand on programme dans un *langage machine* : la séquence ci dessous, qui commence à l'adresse 100, calcule la somme des entiers de 1 à N, en supposant que N est dans le registre 3 et que le résultat doit aller dans le registre 4.

```
100    mettre la valeur 0 dans r4
101    comparer r3 et la valeur 0
102    si égal, aller à 106
103    ajouter r3 à r4
104    ajouter la valeur -1 à r3
105    aller à 101
106    ...
```

Après l'exécution de l'instruction d'adresse 102, le compteur de programme vaudra 103 ou 106, selon la valeur des indicateurs positionnés par l'instruction précédente (101).

Exercice 2. Ecrivez un programme du même type pour le SSEM, en essayant de le faire tenir sur 32 mots.

Comme vous le voyez, un processeur n'est pas bien compliqué. C'est un assemblage de quelques circuits de base : registres, additionneurs, etc. qui sait exécuter des instructions de base très élémentaires.

Ce qui est compliqué c'est de combiner des instructions aussi rudimentaires pour effectuer des traitements utiles (qui ne sont pas forcément simples, eux). Affronter cette complexité, c'est la spécificité du travail du programmeur.

2.3 Les périphériques

Enfin, dès les années 60, une grande variété de périphériques permet l'entrée, la sortie et le stockage des données, ainsi que la communication.

Sur les premiers ordinateurs, les périphériques courants étaient

- les lecteurs de rubans perforés, support fragile et peu commode hérité des téléscripteurs, abandonnés rapidement.
- Les lecteurs et perforateurs de cartes (hérités de la mécanographie),
- les imprimantes ;
- les lecteurs de bandes magnétiques.
- des terminaux interactifs : machines à écrire électriques, vore écrans cathodiques.

Assez rapidement on en est venu à utiliser un petit ordinateur auxiliaire dit "frontal" pour recopier les cartes perforées sur des bandes magnétiques, de lecture bien plus rapide. Et inversement, les résultats de l'ordinateur principal étaient transférés sur des bandes magnétiques que le petit ordinateur se chargeait de faire imprimer ou perforer.

Ainsi on économisait le temps précieux du gros ordinateur.

2.4 Les outils de programmation

Dans les premiers temps de l'informatique, le programmeur écrivait ses programmes en binaire, en utilisant la liste des instructions de l'ordinateur (instruction set) et en les codant lui même en binaire. Pour des programmes de quelques dizaines d'instructions, c'était encore envisageable.

Il est ensuite apparu que cette activité de codage, éminemment fastidieuse, était trop sujette à erreurs. Il était donc préférable d'utiliser un programme pour faire mécaniquement ce codage sans rique d'erreur.

L'histoire officielle dit que l'idée d'utiliser un programme de traduction vient de John Neumann en 1945, mais selon d'autres sources [10], Von Neumann était en fait initialement opposé à cette idée (émise par un étudiant), parce que cela gaspillait le précieux temps de calcul de l'ordinateur, alors qu'on pouvait très bien confier ce travail à des étudiants modestement rémunérés.

Un programme écrit en *langage d'assemblage* se présente comme une suite d'instructions utilisant les codes mnémoniques, comme `addi` pour "add immediate value" sur le PowerPC. Parfois c'est plus obscur, comme `lwzxu` (load word with zero indexed with update). L'*assembleur* est le programme de traduction [11], qui assemble les traductions de chaque instruction.

On est ensuite passé (au milieu des années 50) à la traduction automatique de formules mathématiques, puis de programmes complets, avec le langage FORTRAN (Formula translator). Là aussi, l'intérêt n'était pas évident pour tout le monde. Le même John Von Neumann a déclaré, quand on lui a présenté FORTRAN en 1954 « why would you want more than machine language ? »

Inversement, l'arrivée de langages de haut niveau a parfois soulevé un enthousiame excessif. En effet, il suffisait d'écrire

```
multiply PRIX-UNITAIRE
      by QUANTITE giving   PRIX.
add PRIX to TOTAL.
```

là où, autrefois, un professionnel barbu grassement rémunéré produisait des lignes de code absolument incompréhensibles

```
load   PU
mult   QTE
store  PRIX
add    TOT
store  TOT
```

y compris par son chef de service, incapable d'en vérifier la qualité.

Quand COBOL a été annoncé au début des années 60, certains y ont vu, un peu vite, la fin du métier de programmeur.

C'était en effet la fin d'un certain type de programmation, mais il reste que même si le code est écrit avec des phrases anglaises et semble facile à relire après une formation d'une semaine, la difficulté est en réalité dans l'algorithmique et dans l'organisation du code, composé d'une multitude d'opérations simples. La programmation reste un métier à part entière, qui ne s'improvise pas.

3 Utilisation en monotâche

Les faibles capacités des premiers ordinateurs (quelques dizaines de kilo-octets de mémoire) ne permettaient que de faire exécuter un programme à la fois.

Un opérateur était donc chargé de mettre le travail fourni par les utlisateurs (cartes perforées ou bande magnétique) dans la mémoire de la machine, de lancer l'exécution et de récupérer les résultats imprimés (ou enregistrer). Et de recommencer avec le travail suivant.

Chaque travail disposait donc de l'intégralité des ressources de la machine.

Exercice 3. Les premières machines, expérimentales, étaient utilisées en mono-tâche à la demande. Quand un utilisateur arrivait avec un travail à faire passer sur le calculateur, il devait attendre que les précédents aient libéré la place avant d'utiliser la machine.
Imaginons l'arrivée de 4 utilisateurs :
- John arrive à 8h00, avec un travail qui dure 40 mn
- Grace arrive à 8h10, avec un travail de 30 mn
- Alan arrive à 8h20, avec un travail de 1 h 5 mn
- Niklaus arrive à 8h40, avec un travail de 25 mn

1. Quel est le "temps de service" pour chaque utilisateur (durée entre son arrivée en salle d'attente et la fin de son travail), si ils passent dans l'ordre d'arrivée (politique FIFO, *first-in first-out*) ? Calculez le temps de service moyen.
2. Mêmes questions si les utilisateurs décident de faire passer en premier celui qui [a] a le travail le plus court (politique dite "du plus cours temps d'exécution").

a. parmi ceux qui sont présents

3.1 Moniteur d'enchaînement des travaux

Pour éviter de perdre du temps, on a vite eu l'idée d'automatiser l'enchaînement des travaux. Un petit programme, toujours présent en mémoire, assurait la lecture du travail suivant dès qu'un travail était terminé, et gagnait ainsi de précieuses minutes.

Cet embryon de système peut prendre la forme d'une boucle de quelques instructions pour copier en mémoire les cartes de l'exécutable à charger, avant de lui transférer le contrôle. Un programme utilisateur qui se termine normalement doit simplement relancer le moniteur.

En début de journée (et après chaque crash), l'opérateur manipule les interrupteurs de la console pour entrer ce "chargeur" en mémoire. Dans une version plus élaborée, c'est un dispositif électronique qui lit un ruban perforé contenant le chargeur : il suffit d'appuyer sur un bouton pour "recharger le chargeur".

Encore mieux : le programme sur bande perforée peut servir à charger un système plus volumineux, depuis un autre périphérique plus rapide (bande, disque ou tambour magnétique). C'est ce qu'on appelle le **bootstrapping**, ou **amorçage** : la procédure de démarrage d'un ordinateur, qui comporte notamment le chargement du programme initial, et qui peut se faire en plusieurs étapes.

Ce programme résident comportait aussi des sous-programmes (par exemple lecture-écriture sur bande

10. https://beacon.salemstate.edu/~tevans/VonNeuma.htm
11. mais on dit souvent, par métonymie, "programmer en assembleur"

magnétique, sur disque etc.) qui pouvaient être appelés par les programmes des utilisateurs, et qu'il n'était donc pas nécessaire de recharger avec chaque travail.

3.2 Planification de travaux

Dans les entreprises, des informaticiens étaient chargés de la planification de l'exploitation : un certain nombre de travaux devaient "passer" sur l'ordinateur, à eux de décider quand et dans quel ordre, en tenant compte de diverses contraintes :

- la taille mémoire de chaque programme, et celle de la machine,
- les périphériques utilisés : sur une installation à 3 lecteurs de bandes, on ne peut pas faire tourner en même temps 2 programmes qui ont chacun besoin de 2 lecteurs.
- les priorités définies par l'entreprise [12]

et en optimisant à la fois la satisfaction de chaque service demandeur, et la rentabilisation des matériels.

Exercice 4. Soit une installation avec un ordinateur monotâche. A partir de 8h00, on doit faire passer trois travaux A, B, C qui durent respectivement 1h, 30 min, et 45min. Et les utilisateurs sont évidemment pressés d'obtenir les résultats.
Quel est le temps d'attente moyen des utilisateurs si on les fait passer dans cet ordre sur l'ordinateur (monotâche). Dans l'ordre inverse ? Quel est l'ordre optimal ?

3.3 Modifications matérielles nécessaires

Malheureusement, la coexistence en mémoire du "superviseur" et du travail utilisateur introduit de nouveaux problèmes. En effet, un programme utilisateur "buggé" (intentionnellement ou pas) peut

- altérer la partie de la mémoire réservée au superviseur, conduisant au plantage de la machine
- utiliser de façon incorrecte les instructions d'entrés-sorties (accès illégaux à des fichiers, périphériques endommagés, etc.)

Ceci a conduit à quelques modifications du processeur, suggérées dès la fin des années 50

- le processeur possède deux modes de fonctionnement "maître" (ou superviseur, ou privilégié) et le mode "esclave" (normal). Ceci est matérialisé par une bascule 1 bit.
  - le superviseur s'exécute en mode maître, et les programmes utilisateurs en mode esclave.
  - en mode maître, le processeur a accès à toute la mémoire, et peut exécuter toutes les instructions de la machine.
  - en mode esclave, la zone mémoire accessible par le processeur est restreinte : deux registres indiquent le début de cette zone, et sont constamment comparés avec le compteur ordinal.
  - en mode esclave, certaines instructions (par exemple les instructions d'entrée-sortie directes) ne peuvent par être exécutées [13].
- Quand un programme viole ces règles d'accès, une **exception** se produit : le contrôle est rendu au superviseur, à une adresse fixée au départ. Le superviseur examine donc la situation et décide des suites à donner (reprendre le programme, y mettre fin etc).
- pour appeler les sous-programmes du superviseur, un programme utilise une instruction spéciale ("syscall", trap logiciel, ...), qui provoque aussi une exception. Le superviseur se charge alors d'effectuer (en mode privilégié) l'opération demandée, avant de rendre la main au programme appelant.
- Quand une exception se produit, le contenu du compteur de programme est automatiquement sauvegardé, soit dans une pile en mémoire, soit dans un registre spécial. Ceci permet de reprendre éentuellement l'exécution là où elle en était arrêtée

3.4 Déroulement d'un appel système

Point de vue du programmeur d'application Par exemple, sous MS-DOS, pour faire afficher une chaine de caractères il fallait

- placer le nombre 9 dans le registre AH
- placer l'adresse de la chaîne dans la paire de registres DS :DX
- appeler l'instruction `INT 21H`

Déroulement L'exécution de l'interruption 33 (21H) déclenche l'appel au système d'exploitation, dans la **routine de traitement de l'interruption 21H**. Ce sous programme consulte le registre AH qui indique la fonction demandée : 9 = affichage d'une chaîne. Une fois cette fonction affectuée (par copie de caractères dans la mémoire de l'écran), le système place le code de retour 24H (36) dans le registre AL, et rend la main au programme utilisateur.

4 Profiter des temps morts : la multi-programmation

4.1 Motivation économique

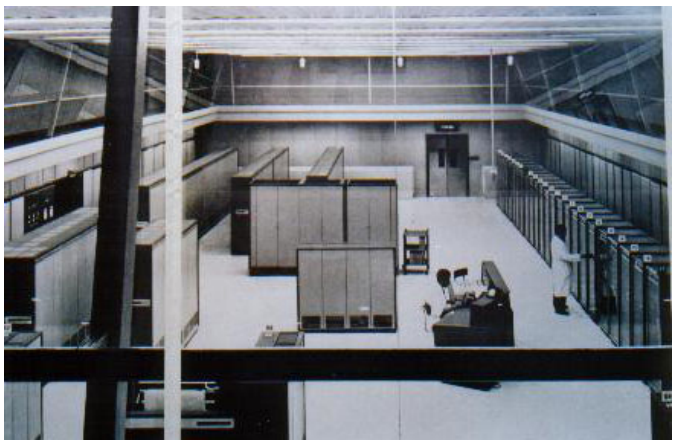
A l'époque (début des années 60) les ordinateurs coûtent une fortune, on essaie donc de les rentabiliser au maximum.

Les machines sont composées d'une unité centrale (processeur et mémoire) et de périphériques : lecteurs de cartes perforées, de bandes, imprimantes etc. Or on observe que les opérations d'entrées-sorties sont extrêmement lentes par rapport aux possibilités d'un processeur.

12. Du moins présentées comme telles par le Directeur Informatique, qui trouve là un moyen d'asseoir sa position stratégique dans l'entreprise en négociant sa collaboration avec d'autres Directeurs..

13. Dans le Stretch d'IBM, les ingénieurs ont oublié de rendre privilégiée l'instruction qui permet d epasser en mode maître. Doh !

Prenons par exemple le Gamma 60 dont le premier exemplaire a été livré par la société Bull à la SNCF en 1958, avec 6 imprimantes et 16 dérouleurs de bande, il occupait $360m^2$.



source : `http://histoireinform.com/Histoire/+infos2/chr4infa.htm`

Pour les périphériques :
- le lecteur de ruban fonctionnait à 300 caractères par seconde
- les cartes perforées de 80 colonnes étaient lues à 300 cartes/mn
- les imprimantes fonctionnaient à 300 lignes par minute

et une instruction (opération sur nombres de 10 chiffres) prenait de 100 à 500 microsecondes, soit des dizaines de milliers par seconde.

Dans ces conditions, il est clair que le processeur est le plus souvent en attente d'une E/S.

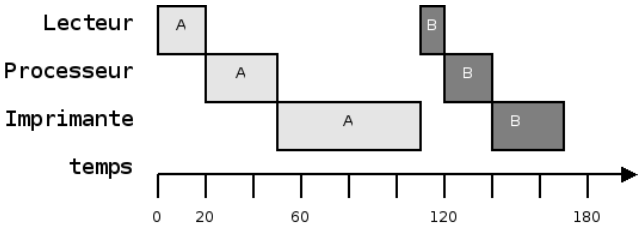
L'idée est donc de faire cohabiter plusieurs tâches dans la mémoire : quand la tâche "active" demande à lire des cartes sur le lecteur, on met à profit le temps libre du processeur pour faire avancer une autre tâche.

4.2 Étude d'un cas

Imaginons deux tâches A et B :
- la chargement de A depuis le lecteur de cartes dure 20 secondes, elle fait du calcul pendant 30 secondes, et l'impression des résultats prend 1 minute ;
- le chargement de la seconde B dure 10 secondes, son calcul 20 secondes et l'impression 30 secondes.

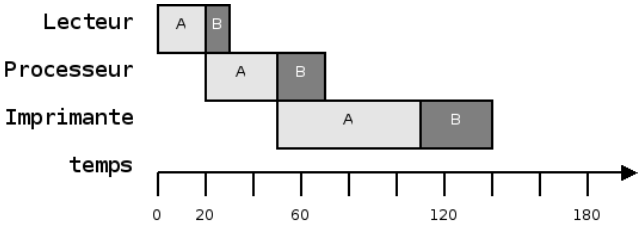
Le graphique ci-contre montre ce qui se passe sous le contrôle d'un "moniteur d'enchaînement de travaux". La tâche B n'est chargée en mémoire que quand A s'est terminée ($t = 110s$) et se termine à $t = 170s$.

Le processeur a travaillé $30 + 20 = 50s$, soit un taux d'occupation de $50/170 = 29,4\%$.



Exercice 5.
- calculez le taux d'occupation du lecteur de cartes
- calculez le taux d'occupation de l'imprimante.

Voici maintenant le déroulement dans un système multitâche ; la tâche B est chargée dès que le lecteur a été libéré, puis est exécutée quand le processeur est libre, etc.



Exercice 6.
- Calculez les taux d'occupation, comparez avec les chiffres précédents.
- Même question si on commence par exécuter B au lieu de A.
- Imaginons qu'il s'y ajoute une troisième tâche C semblable à B. Représentez le déroulement dans les deux cas (enchaînement séquentiel et multitâche). Comparez les chiffres.

4.3 La mise en oeuvre du multitâche

Pour mettre en oeuvre efficacement le multitâche, il faut que le processeur ne soit pas bloqué en attente des opérations d'entrée-sorties, qui doivent se dérouler en parallèle avec les calculs.

On confie donc le pilotage de périphériques à des circuits spécialisés, à qui le processeur enverra des commandes (requêtes d'E/S), et qui préviendront le processeur, par un **signal d'interruption**, quand la requête est terminée. Les interruptions sont traitées comme les exceptions vues plus haut.

Le fonctionnement par interruption décharge ainsi le processeur de la surveillance des périphériques, qui peut consacrer son temps à l'avancement des autres tâches.

L'idée des interruptions est apparue en 1955. La NASA possédait un Univac 1103 pour ses besoins de calculs scientifiques (traitement de données d'essais en tunnel de soufflerie avec Boeing) et ses applications administratives. [14]

Le programme de collecte de données était chargé en mémoire et présent pendant que les applications de gestion tournaient. Quand les tests en soufflerie étaient prêts, un bouton poussoir situé dans le hangar permettait d'activer le programme "instantanément" [15] : le contenu des différents registres de l'ordinateur (compteur ordinal, conditions, ...) était alors sauvegardé et le contrôle était transféré au programme de collecte de données.

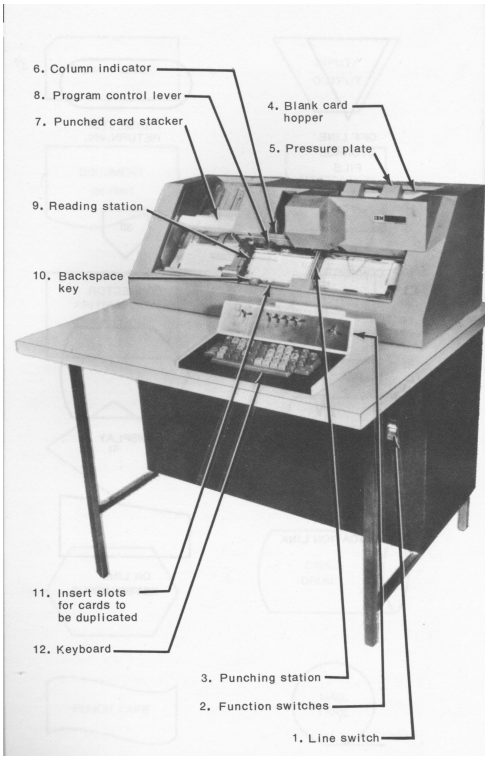
5 Fonctionnement d'un centre de calcul

Les premiers systèmes multi-tâches sont toujours destinés au *traitement par lots* (*batch processing*), mode d'exploitation qui est assez similaire à '*enchaînement automatique des travaux* décrit plus haut : on "enfourne" dans la machine une suite de travaux à réaliser, qui ressortent une fois terminés.

La différence, c'est qu'avec le multitâche, le travail N+1 peut être chargé en mémoire avant que le travail N ne soit terminé : on charge autant de programmes que possible pour remplir la mémoire, pour avoir un meilleur rendement et ne pas gaspiller le temps du processeur. Ils ne se terminent pas forcément dans l'ordre où ils ont commencé.

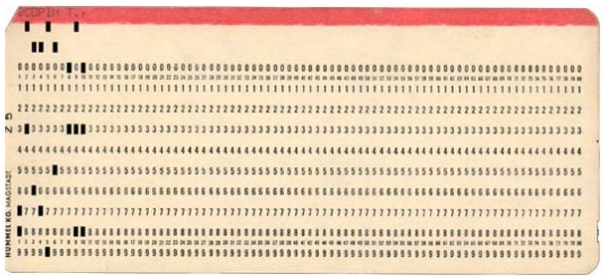
Dans les années 70, les étudiants en informatique de Bordeaux 1 travaillaient de la façon suivante

1. il fallait d'abord écrire le programme sur le papier
2. puis se rendre dans une salle où se trouvaient quelques perforatrices comme l'IBM 29 :



(source : `http://www.math-cs.gordon.edu/courses/cs323/FORTRAN/keypunch.html`)

pour transcrire le programme sur des cartes perforées [16]. Chaque carte contenait une ligne de 80 caractères.



3. les cartes, entourées par un élastique, étaient placés dans un bac qu'un étudiant (rémunéré) allait porter au Centre de Calcul Interuniversitaire, où se trouvait l'ordinateur IRIS 80, trois ou quatre fois par jour. Les bacs de cartes étaient alors confiés à un opérateur, qui les plaçait dans le lecteur de cartes, lançait le traitement et récupérait (bien plus tard) les cartes avec les listings de résultats.

14. `http://www.cs.clemson.edu/~mark/interrupts.html`
15. la solution précédente consistait à réserver l'utilisation de l'ordinateur à l'avance, et à téléphoner à l'opérateur pour qu'il lance le programme le moment donné
16. avec 4 perfos, il fallait évidemment faire la queue, un peu moins la nuit et le week-end

Source `http://www.feb-patrimoine.com/projet/iris80/iris80.htm`.

4. il en profitait pour ramener les travaux précédents, avec les listings de résultats.
5. en récupérant son travail, l'étudiant constatait généralement qu'il manquait un point-virgule quelque part : ne restait plus qu'à trouver où, remplacer la carte fautive, et remettre le paquet dans le bac de départ, pour avoir le résultat quelques heures plus tard.

Dans ces conditions, il était évidemment préférable de réfléchir avant de taper, et de se relire soigneusement plusieurs fois avant de mettre les cartes dans le bac...

Les étudiants de troisième cycle, et les chercheurs en mathématiques et informatique, disposaient quant à eux d'une petite salle avec quelques **terminaux conversationnels** hétéroclites : telex Olivetti, terminaux à écran cathodiques (HP 2621), clavier avec imprimante thermique, écran graphique Textronix 4027, reliées directement au centre de calcul par des lignes à 9600 bit/s. De quoi travailler très confortablement.

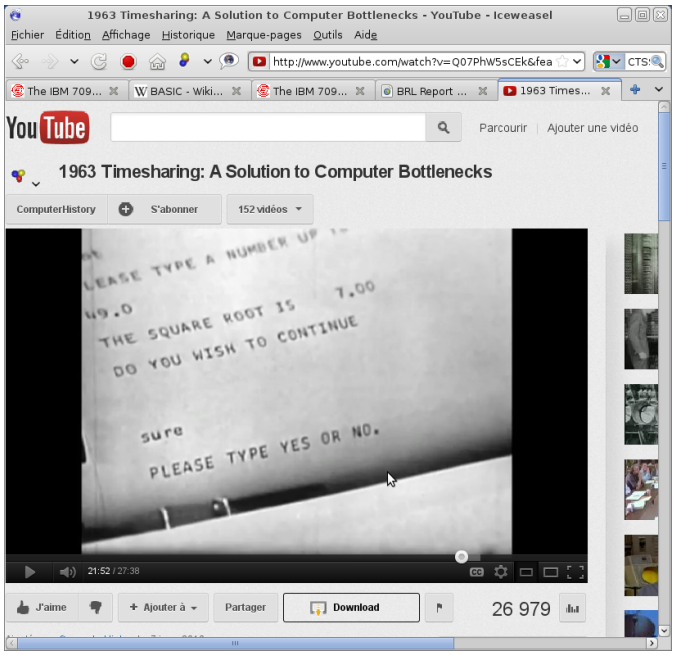
6 Travailler à plusieurs : le temps partagé

En effet un nouveau besoin est apparu avec l'utilisation de terminaux interactifs : telex transformés, machines à écrires électriques, et écrans alphanumériques. Au départ les utilisateurs peuvent soumettre de nouvelles tâches dans le traitement par lots, mais les programmes interactifs amènent une contrainte supplémentaire : chaque utilisateur doit avoir l'impression d'utiliser une machine "réactive" : si un collègue lance un programme de calcul lourd (quelques milliers de décimales de π)[17], ça ne doit pas empêcher les autres de travailler en monopolisant le temps du processeur.

C'est la prise en compte de cette contrainte qui conduit au *time sharing* : le temps du processeur est partagé "équitablement" entre les utilisateurs.

Parmi les premiers systèmes, le plus connu est CTSS (Compatible Time Sharing System) issu du projet MAC (Multi Access Computer) de John McCarthy au MIT[18]. Ce système multi-utilisateurs (à partir de 1961, en production en 1964) tournait sur un IBM 7094 modifié, avec une mémoire de 2 fois 32K mots de 36 bits. Lire par exemple "*The IBM 7094 and CTSS*" par Tom Van Vleck, `http://www.multicians.org/thvv/7094.html`

A voir absolument : le reportage "*1963 Timesharing : A Solution to Computer Bottlenecks*" (27 minutes) sur `http://www.youtube.com/watch?v=Q07PhW5sCEk`, avec une longue interview de Fernando J. Corbato, responsable du projet, qui explique le fonctionnement du temps partagé, suivie par une démonstration.



Une démonstration plus courte `http://www.youtube.com/watch?v=sjnmcKVnLi0` (*Robert Fano explains scientific computing*), à partir de 5 :20.

Exercice 7. Imaginons que 3 utilisateurs d'un système en temps partagé lancent en même temps des travaux qui nécessitent 10 minutes de calcul chacun.

1. si ces travaux sont envoyés dans une file d'attente pour être traités un par un, le premier utilisateur aura sa réponse dans 10 minutes, le second dans 20 minutes, et le troisième dans 30, d'où un temps d'attente moyen de 20 minutes ;
2. si ils se déroulent en temps partagé, ils dureront tous les trois 30 minutes.

Dans ce contexte, pourquoi les utilisateurs préfèrent-il quand même la seconde solution ?

17. ou un programme qui boucle, comme ça arrive parfois.
18. John McCarthy, décédé en octobre 2011, est un des pionniers de l'informatique : systèmes d'exploitation, intelligence artificielle, programmation symbolique et fonctionnelle, etc. Lire sa biographie sur Wikipedia. Parmi ses articles, un mémo "*A Time Sharing Operator Program for Our Projected IBM 709*" daté du 1er janvier 1959. Voir ses souvenirs sur le time-sharing dans `http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html`

7 De nos jours...

Les systèmes d'exploitation modernes[19] sont tous capables de faire exécuter plusieurs tâches en même temps. Sous Unix, la commande `top` vous permet de voir les tâches en cours : sur un ordinateur personnel vous constaterez qu'il en a au moins une bonne centaine, plusieurs milliers sur un serveur.

Les solutions qui permettent le fonctionnement en multi-tâches ont été trouvées, mises en place et généralisées, dès le début des années 60 : interruptions, partage de la mémoire, etc.

Dans les années 70 et 80, on a assisté à un recul apparent : les premiers micro-ordinateurs étaient destinés à un usage personnel.

Les contraintes de coût qui ne permettaient qu'une faible capacité mémoire (dizaines ou centaines de kilo-octets) expliquent la réapparition, pendant une dizaine d'années, des systèmes mono-tâches, comme CP/M (Kildall, 1977) et MS-DOS (1981).

Exercice 8. Avez-vous vraiment besoin d'un système multi-tâches, avec protection mémoire etc, dans votre smartphone ?

19. à l'exception de quelques systèmes embarqués ou de supercalculateurs