Le langage de la machine

ASR2 - Système

Semestre 2, année 2012-2013

Département informatique
IUT Bordeaux 1

Mars 2013

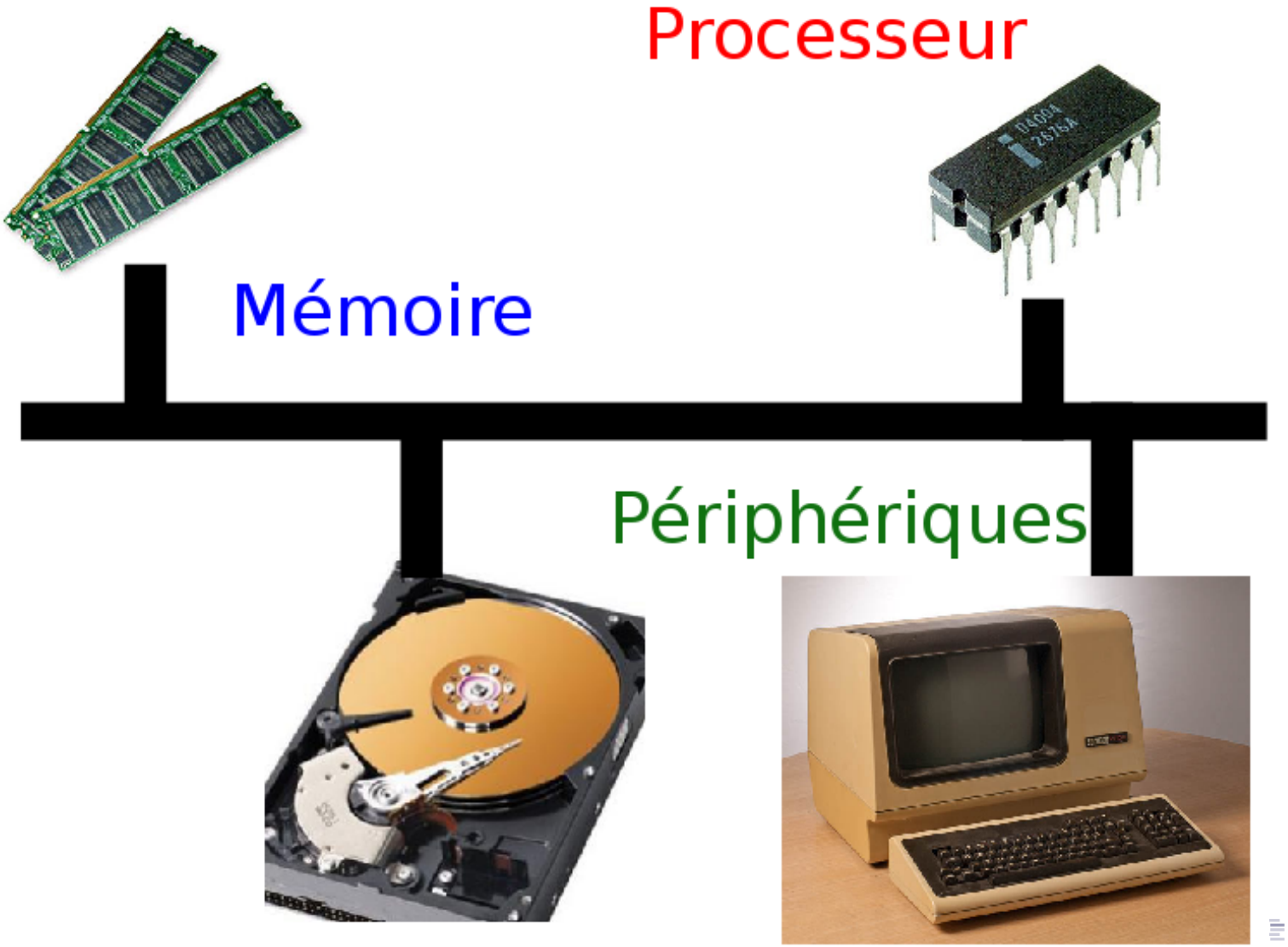
Première partie

Structure d'un ordinateur

Contenu

Structure d'un ordinateur

Un ordinateur comporte un *processeur*, de la *mémoire*, des *dispositifs d'entrée-sortie*.



Rôle des éléments

Ces éléments interagissent :

- la mémoire contient les données,
- le processeur exécute les instructions prises dans la mémoire ;
- ces instructions
  - effectuent des calculs,
  - prennent et placent des données en mémoire,
  - les envoient ou les lisent sur les dispositifs d'entrée-sortie
- les périphériques assurent
  - le stockage des données à long terme
  - la communication avec l'environnement

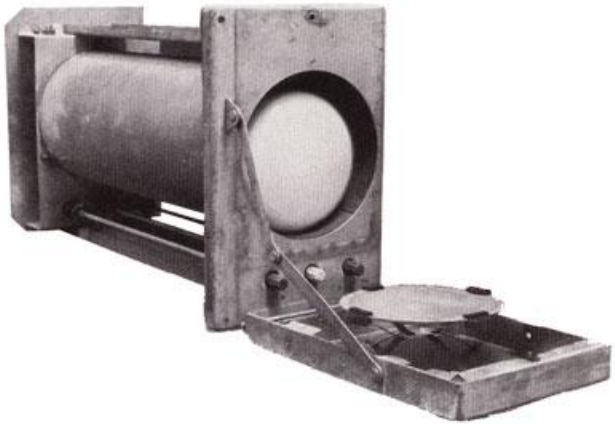
SSEM : le premier ordinateur

Small-Scale Experimental Machine, Université de Manchester, 1948

Première machine à architecture Von Neumann :
instructions et données enregistrés en mémoire.

SSEM, un calculateur expérimental

- banc de test pour une nouvelle technologie de mémoire : les tubes de Wilkins-Kilburn



un tube : 32 mots de 32 bits

- très limité
  - 330 diodes, 250 pentodes,
  - un accumulateur 32 bits
  - mémoire de 32 mots
  - 7 opérations, pas d'entrées-sorties

SSEM : démonstration concluante

Démonstration du 21 juin 1948

- programme de 17 instructions,
- 3,5 millions d'instruction en 52 minutes, soit 1,1 KIPS

- Fiabilité des tubes de Williams-Kilburn
  - des heures / millions d'opérations sans erreur !
  - employés dans le premier ordinateur d'IBM (1952)
    IBM 701 : 32 tubes de Williams
  - ensuite remplacés par les mémoires à tores de ferrite
  - et les mémoires à semi-conducteurs (fin années 70)
- Validation du concept d'ordinateur : calculateur à programme enregistré en mémoire vive
- Début d'une série d'ordinateurs britanniques :
  Mark 1, Ferranti Mark 1 (1951, premier ordinateur commercialisé),
  LEO I, II, et III (fabriqués par Lyons), etc.

De nos jours

Processeurs beaucoup plus complexes : plusieurs coeurs, des lignes de caches, des coprocesseurs etc.

Évolution du nombre de transistors par processeur

année	transistors	processeur
1971	2,300	Intel 4004, premier microprocesseur
1978	29,000	Intel 8086, premiers PC
1979	68,000	Motorola 68000
1989	1,180,000	Intel 80486
1993	3,100,000	Pentium
1997	9,500,000	Pentium III
2000	42,000,000	Pentium 4
2012	1,400,000,000	Quad-Core + GPU Core i7
2012	5,000,000,000	62-Core Xeon Phi

Source : `http://en.wikipedia.org/wiki/Transistor_count`

Deuxième partie

Structure d'un processeur

Contenu

Principes de base d'un processeur

Dans un processeur il y a

- des registres : circuits capables de mémoriser quelques bits d'information
- des circuits combinatoires (additionneur, comparateurs, ...),
- de la logique séquentielle pour gérer le déroulement des différentes phases des

instructions

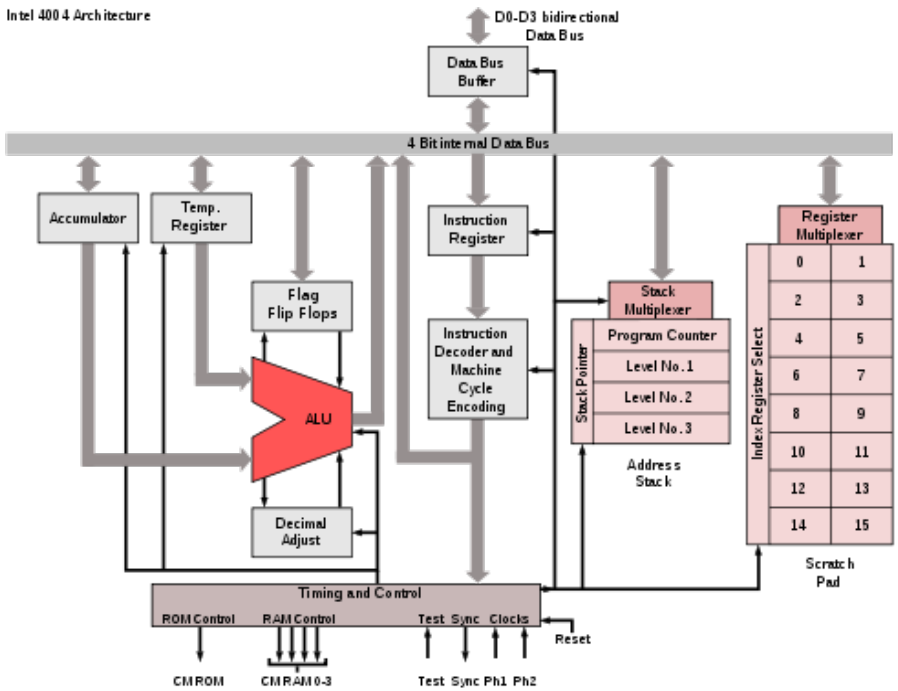


Illustration : architecture du INTEL 4004

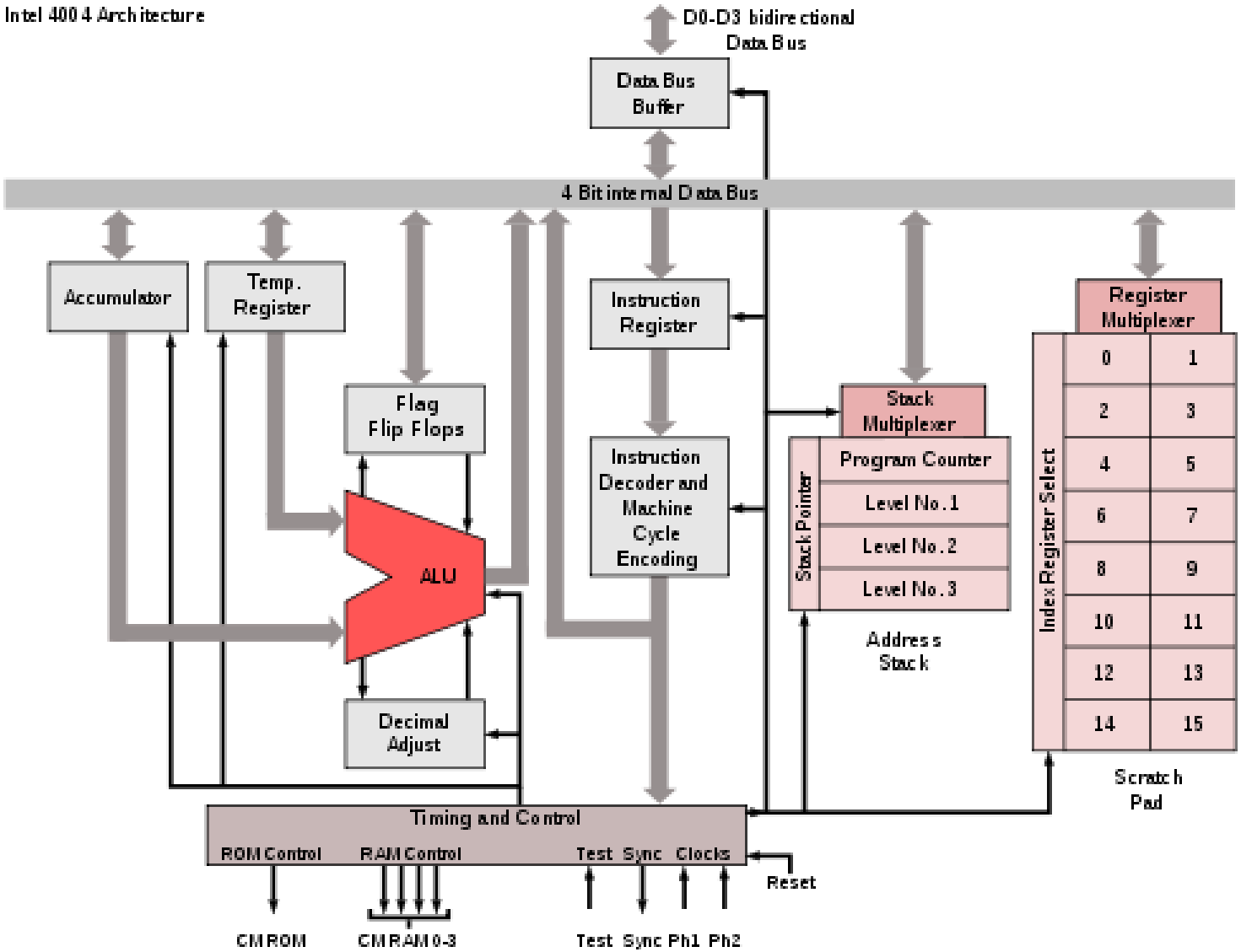
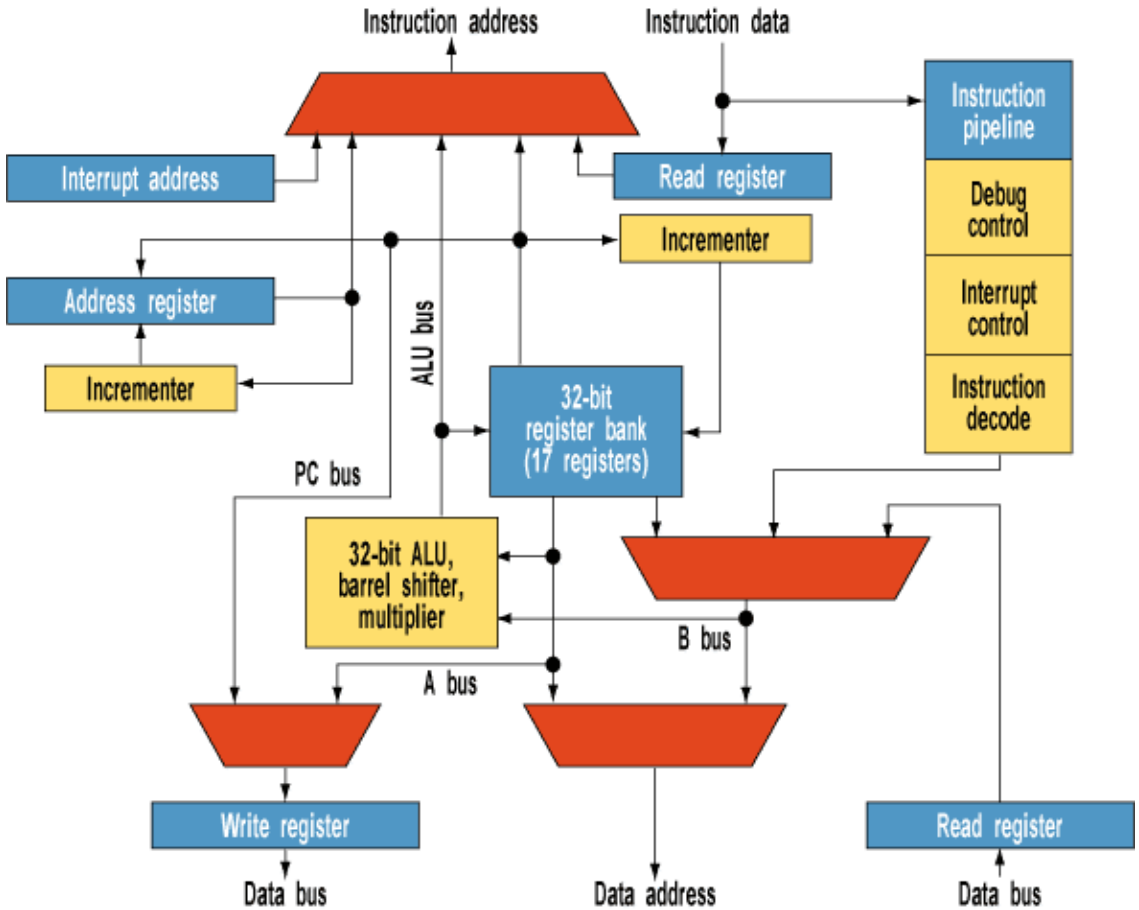


Illustration : Architecture ARM, Cortex M3



The Cortex M3's Thumbnail architecture looks like a conventional Arm processor. The differences are found in the Harvard architecture and the instruction decode that handles only Thumb and Thumb 2 instructions.

le “Modèle du programmeur”

Le programmeur n’a pas à connaitre tous ces détails, seulement

- le jeu d’instructions qu’il peut employer
  - les différents types d’instruction
  - leur effet sur les registres accessibles
- les registres auquel il a accès
  - le compteur de programme (adresse de la prochaine instruction)
  - les registres de travail,
  - les indicateurs de condition
  - ...

Troisième partie

Un processeur fictif

Contenu

- Utilisation de mnémoniques
- Réservation de données
- Utilisation d'étiquettes
- Conventions d'écriture des sources

Un processeur fictif

Eléments

- machine à mots de 16 bits, adresses sur 12 bits
- 1 accumulateur 16 bits
- compteur ordinal 12 bits
- jeu de 13 instructions sur 16 bits
  - arithmétiques : addition et soustraction 16 bits, complément à 2.
  - chargement et rangement directs et indirects
  - saut conditionnel et inconditionnel, appel de sous-programme
  - ...

Format des instructions

1 instruction = 16 bits. Format unique :

- code opération sur 4 bits (poids forts)
- opérande sur 12 bits

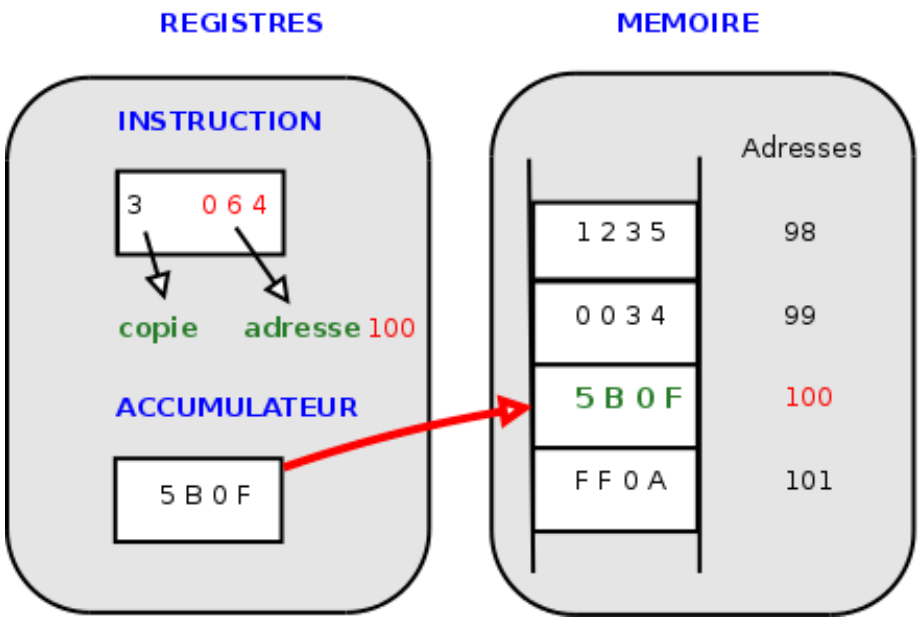
code 4 bits	opérande 12 bits
- - - -	- - - - - - - - - - - -

exemple

Le mot `0011 0000 0110 0100` (0x3064) peut représenter une instruction

- de code `0011` = 0x3
- d'opérande `0000 0110 0100` = 0x064 (100 en décimal)

qui signifie "*ranger le contenu de l'accumulateur dans le mot mémoire d'adresse 100*"



Instruction ou donnée ?

Le mot 0x3064 représente :

- une instruction (code 3, opérande 100)
- le nombre +12388 en binaire complément à 2

La signification d'un mot en mémoire dépend de ce qu'on en fait.

Le jeu d'instructions

	Mnémonique	Description	Action	Cp =
0	`loadi` *imm12*	chargement immediat	`Acc = ext(imm12)`	`Cp + 1`
1	`load` *adr12*	chargement direct	`Acc = M[adr12]`	`Cp + 1`
2	`loadx` *adr12*	chargement indirect	`Acc = M[M[adr12]]`	`Cp + 1`
3	`store` *adr12*	rangement direct	`M[adr12] = Acc`	`Cp + 1`
4	`storex` *adr12*	rangement indirect	`M[M[adr12]] = Acc`	`Cp + 1`
5	`add` *adr12*	addition	`Acc += M[adr12]`	`Cp + 1`
6	`sub` *adr12*	soustraction	`Acc -= M[adr12]`	`Cp + 1`
7	`jmp` *adr12*	saut inconditionnel		`adr12`
8	`jneg` *adr12*	saut si négatif		si `Acc < 0` alors `adr12` sinon `Cp+1`
9	`jzero` *adr12*	saut si zero		si `Acc==0` alors `adr12` sinon `Cp+1`
A	`jmpx` *adr12*	saut indirect		`M[adr12]`
B	`call` *adr12*	appel	`M[adr12] = Cp+1`	`M[adr12]+1`
C	`halt 0`	arrêt		

Les classes d'instructions

4 classes :

Transferts

pour charger une valeur dans l'accumulateur
ou placer le contenu de l'accumulateur en mémoire (load, store).

Arithmétique

addition et soustraction (add, sub)

Branchements

pour continuer à une adresse donnéee (jump, call)

Divers

halt

Programmes

- Charger un programme, c'est remplir la mémoire avec un contenu : instructions et données.

Exemple de programme)

```
0009 5005 6006 3007 C000 0005 0003 0000
```

- l'exécution commence (par convention) au premier mot :
  - le premier mot contient 0009, qui correspond à "loadi 9" (charger la valeur immédiate 9 dans l'accumulateur)
  - le second mot contient 5005, soit "add 5' (ajouter le mot d'adresse 5 à l'accumulateur)
  - ...

Utilisation de mnémoniques

Exemple de programme

```
0009 5005 6006 3007 C000 0005 0003 0000
```

Traduisons les 5 premiers mots en utilisant les codes mnémoniques des opérations

adresse	contenu	mnémonique	opérande
0	0009	loadi	9
1	5005	add	5
2	6006	sub	6
3	3007	store	7
4	C000	halt	0

En clair, ce programme charge la valeur 9 dans l'accumulateur, lui ajoute le contenu du mot d'adresse 5, retranche celui de l'adresse 6 et range le résultat à l'adresse 7. Et il s'arrête.

Réservation de mots

Exemple de programme

```
0009 5005 6006 3007 C000 0005 0003 0000
```

aux adresses 5, 6, et 7 on trouve les valeurs 5, 3 et 0,

adresse	contenu	directive	opérande
5	0005	word	5
6	0003	word	3
7	0000	word	0

La *directive* `word` indique la réservation d'un mot mémoire, avec sa valeur initiale.

Étiquettes symboliques

Il est commode de désigner les adresses par des noms symboliques, les étiquettes :

```
    load    9
    add     5
    sub     6
    store   7
    halt    0

    word    5
    word    3
    word    0
```

```
            load    9
            add     premier
            sub     second
            store   resultat
            halt    0

premier     word    5
second      word    3
resultat    word    0
```

Assemblage

Le programmeur écrit ses programmes en langage d'assemblage.
Le code source comporte

- des instructions
- des directives de réservation
- des commentaires

qui font apparaître

- des codes mnémoniques
- des étiquettes

La traduction de ce code source est faite par un assembleur.

Conventions d'écriture

Sur chaque ligne

- l'étiquette est facultative.
  En colonne 1 si elle est présente.
  Si elle est absente, la ligne commence un espace (au moins)

```
debut loadi 100      # étiquette et instruction
      sub   truc     # instruction sans étiquette
```

- si l'étiquette est seule, elle se rapporte au prochain mot

```
fin                  # étiquette seule
      halt 0
```

Quatrième partie

Programmation

Contenu

Instructions de base

Pour commencer :

chargement	immédiat	loadi valeur
chargement	direct	load adresse
rangement	direct	store adresse
addition	directe	add adresse
soustraction	directe	sub adresse
arrêt		halt 0

Attention ne pas confondre les opérandes immédiats et directs

- `loadi 100` charge *la constante 100* dans l'accumulateur
- `load 100` copie *le mot d'adresse 100* dans l'accumulateur

Prise en main du simulateur

`firefox ~/Bibliotheque/ASR2-systeme/WebSim16/index.html`

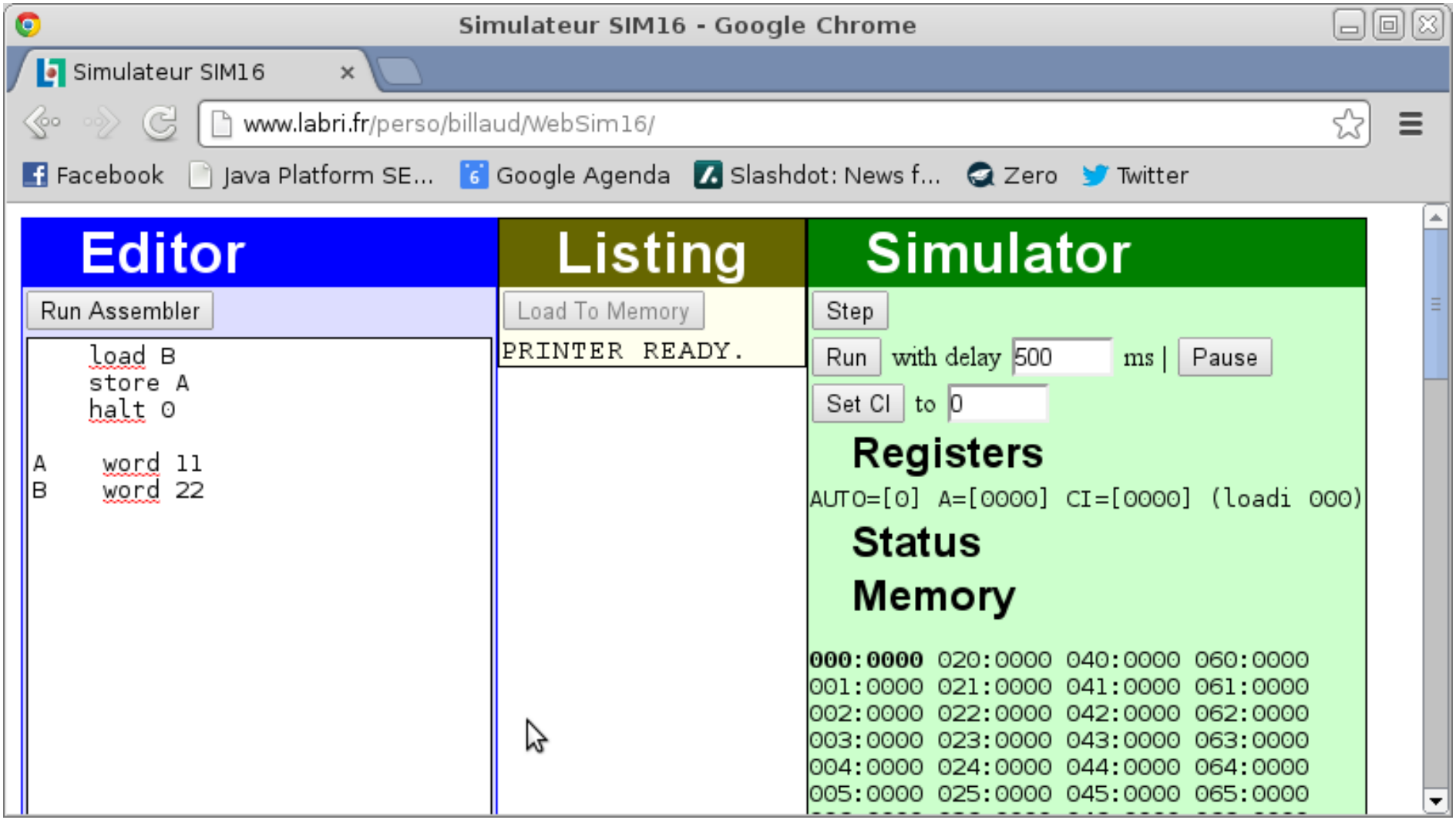
Exemple : traduction de l'affectation "A = B"

```
    load  B
    store A
    halt 0

A   word 11
B   word 22
```

Utilisation 1/4

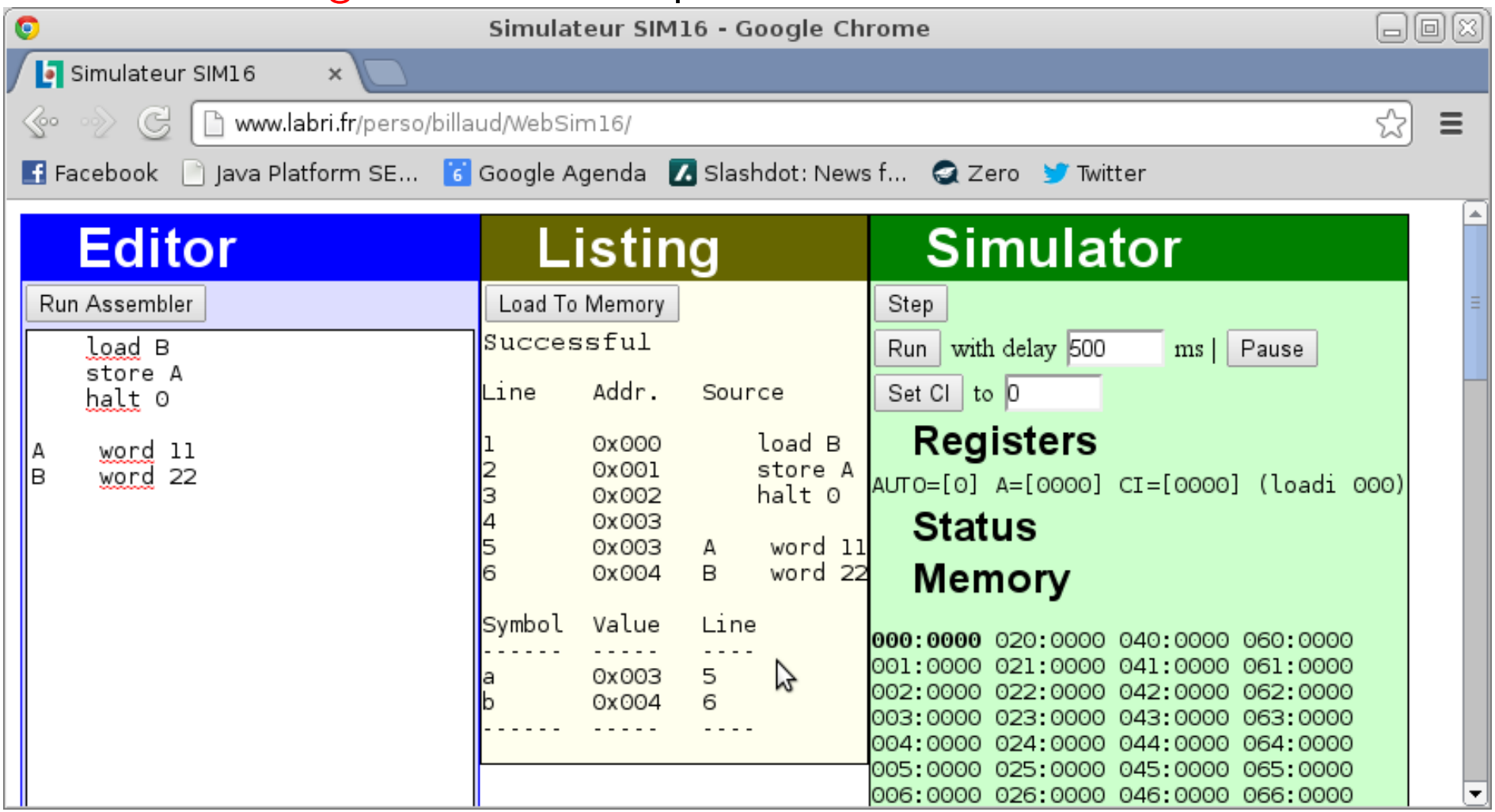
On tape le programme dans l'éditeur



et on lance l'assembleur...

Utilisation 2/4

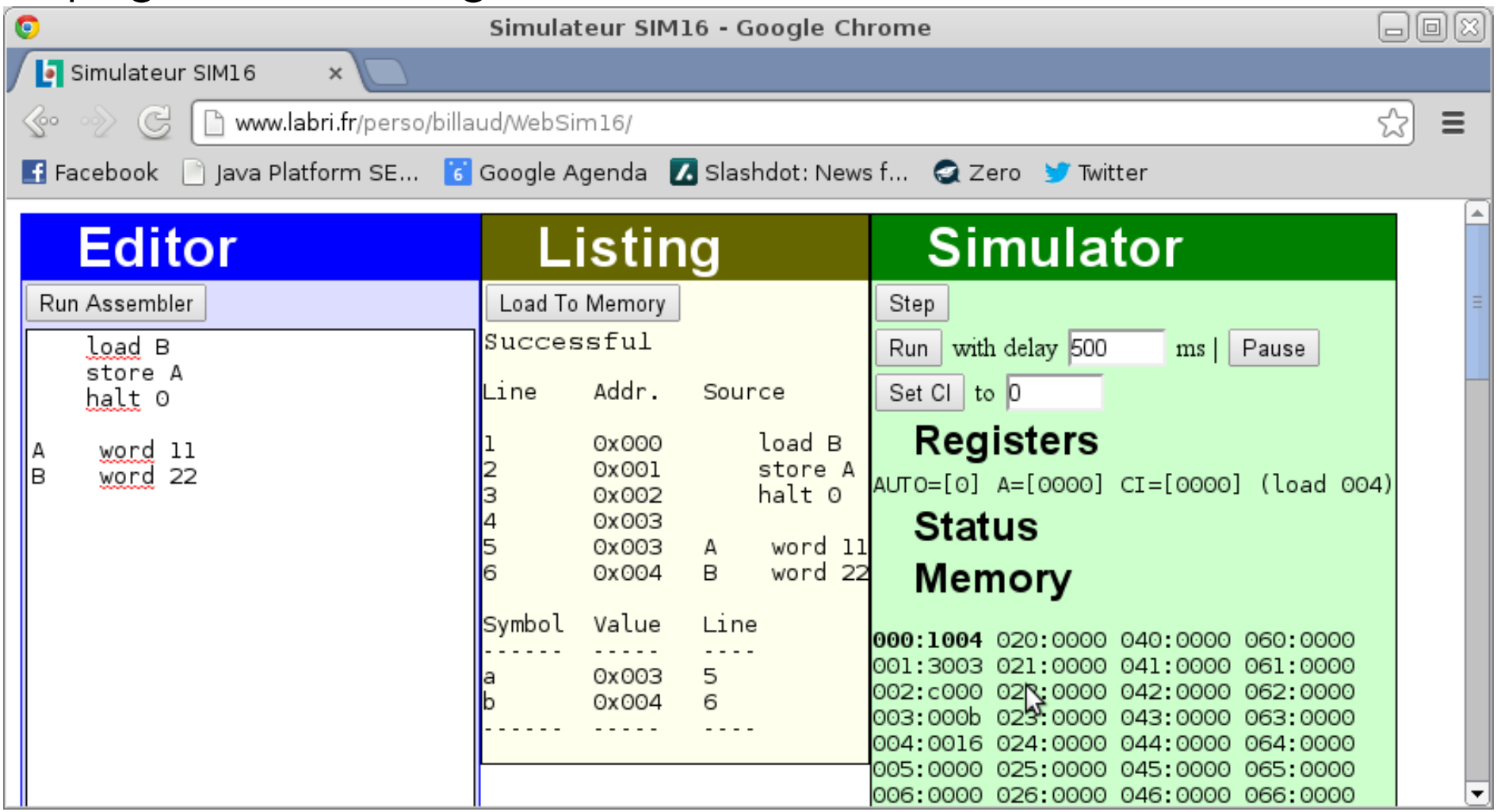
La fenêtre Listing montre un compte-rendu de la traduction.



On charge le code binaire en mémoire

Utilisation 3/4

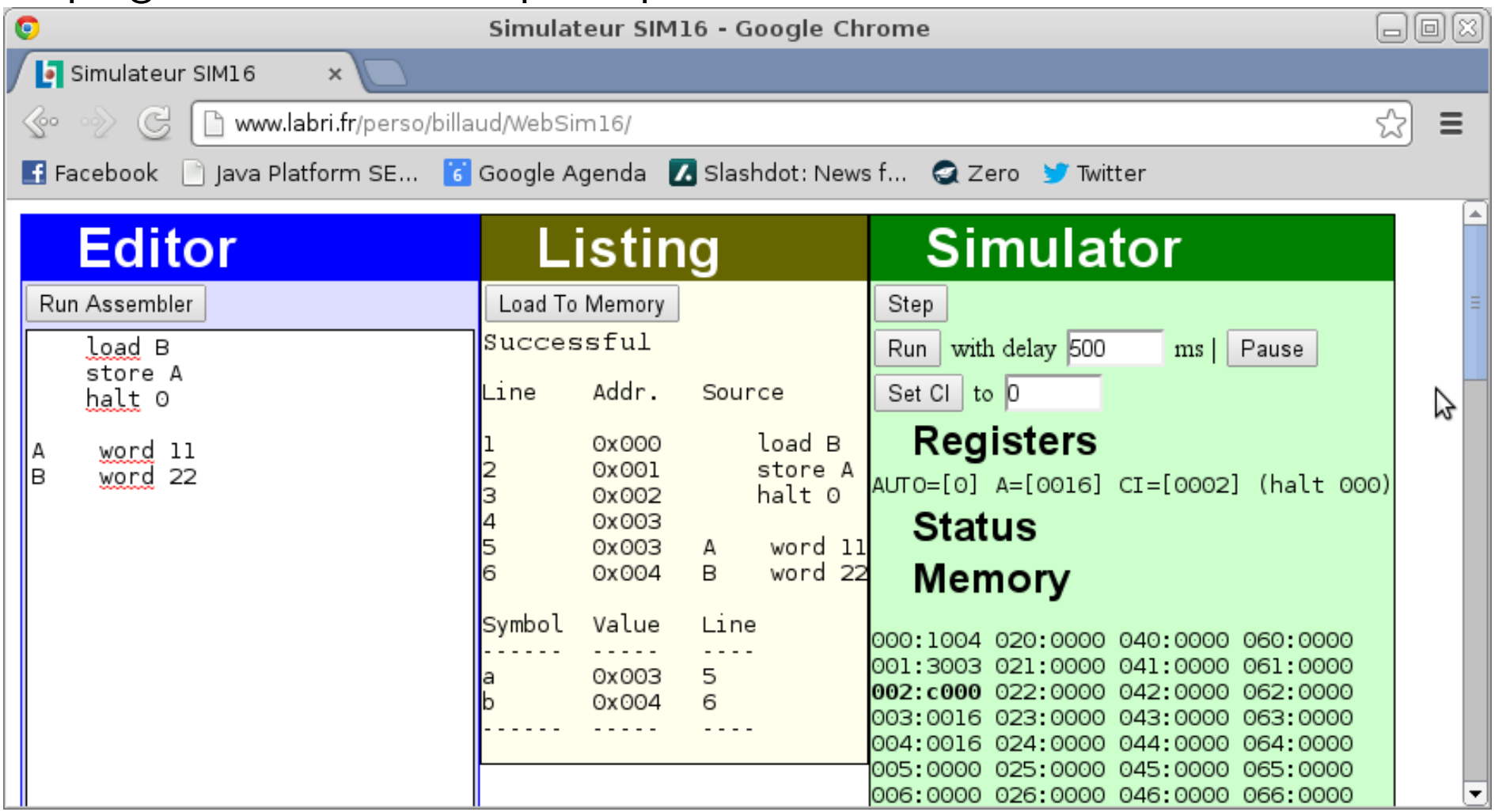
Le programme est chargé dans la denètres Simulator



On peut lancer l'exécution (run)

Utilisation 4/4

Le programme se déroule pas à pas



Et on peut suivre l'évolution du contenu des registres et de la mémoire.

Exercices

À vous de jouer : traduisez les affectations

- A = A + B
- A = A + 1
- A = B + C - 1
- échange de deux variables ?

Bilan d'étape

Bravo !

- vous maîtrisez déjà la moitié (presque) des instructions
- vous savez les employer pour programmer
  - des expressions arithmétiques
  - de affectations

Sauts conditionnels et inconditionnel

Les instructions de saut

saut	inconditionnel	jmp adresse
saut	si accumulateur nul	jzero adress
saut	si accumulateur négatif	jneg adress

qui consultent l'accumulateur et agissent sur le registre "compteur de programme" (`Cp`).
Pour les deux sauts conditionnels, le déroulement se poursuit

- à l'adresse indiquée si la condition est vraie (`Cp = adresse`),
- en séquence sinon (`Cp = Cp + 1`).

Utilisation

Instructions rudimentaires, mais suffisantes pour réaliser

- des alternatives (si-alors, si-alors-sinon, ...)
- des boucles (tant-que, répéter, ...)

Exemple : calcul de la valeur absolue V d'un nombre X

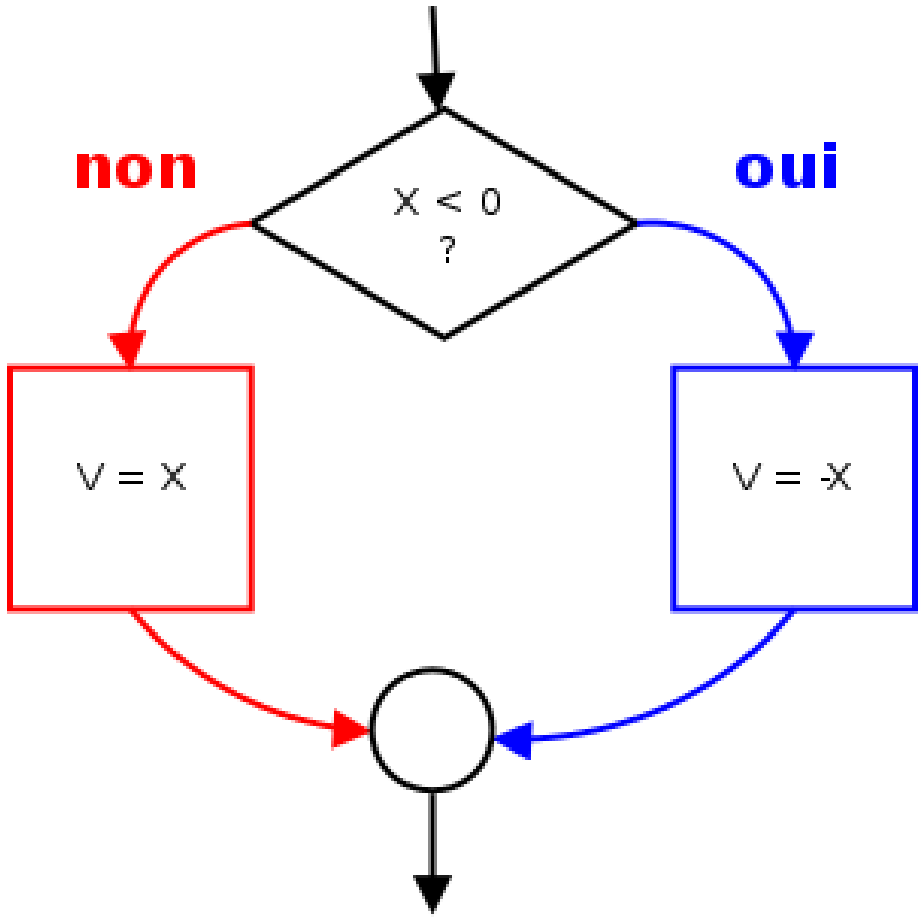
Algorithme structuré :

```
si X < 0
alors
   |  V = -X
sinon
   |  V =  X
```

Organigramme 1/4

L'algorithme

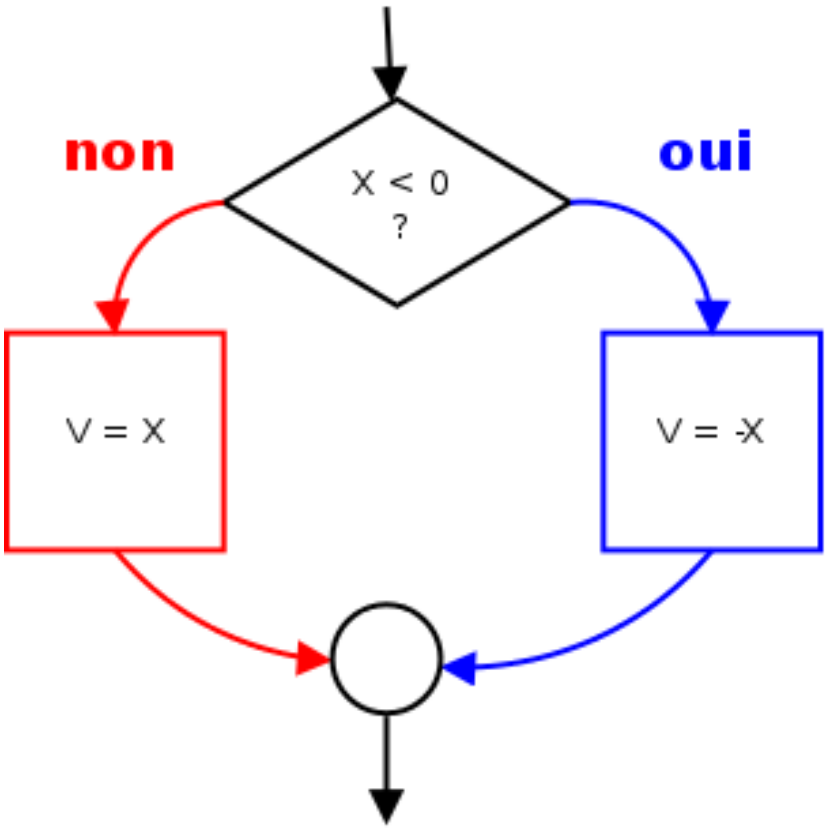
```
si X < 0
alors
    | V = -X
sinon
    | V = X
```

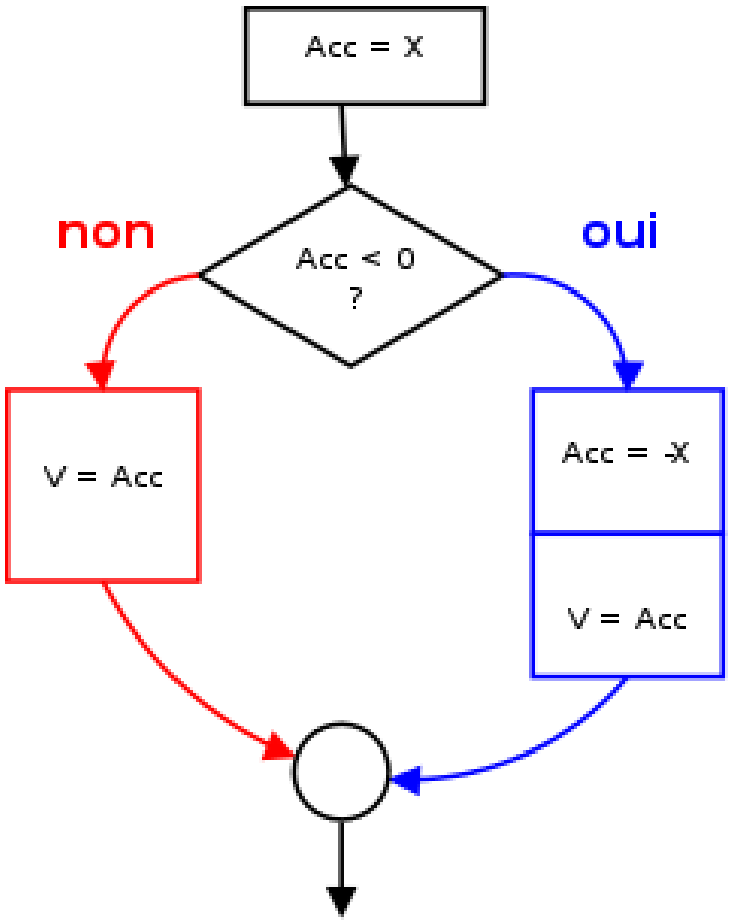


peut être représenté par un organigramme

Organigramme 2/4

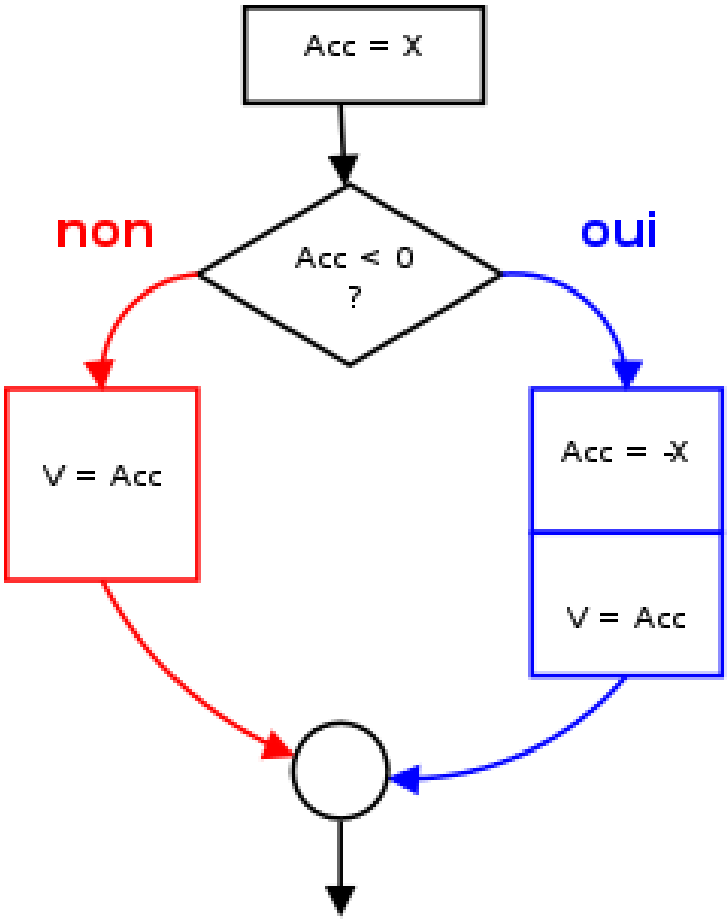
Faisons maintenant apparaître l'accumulateur :

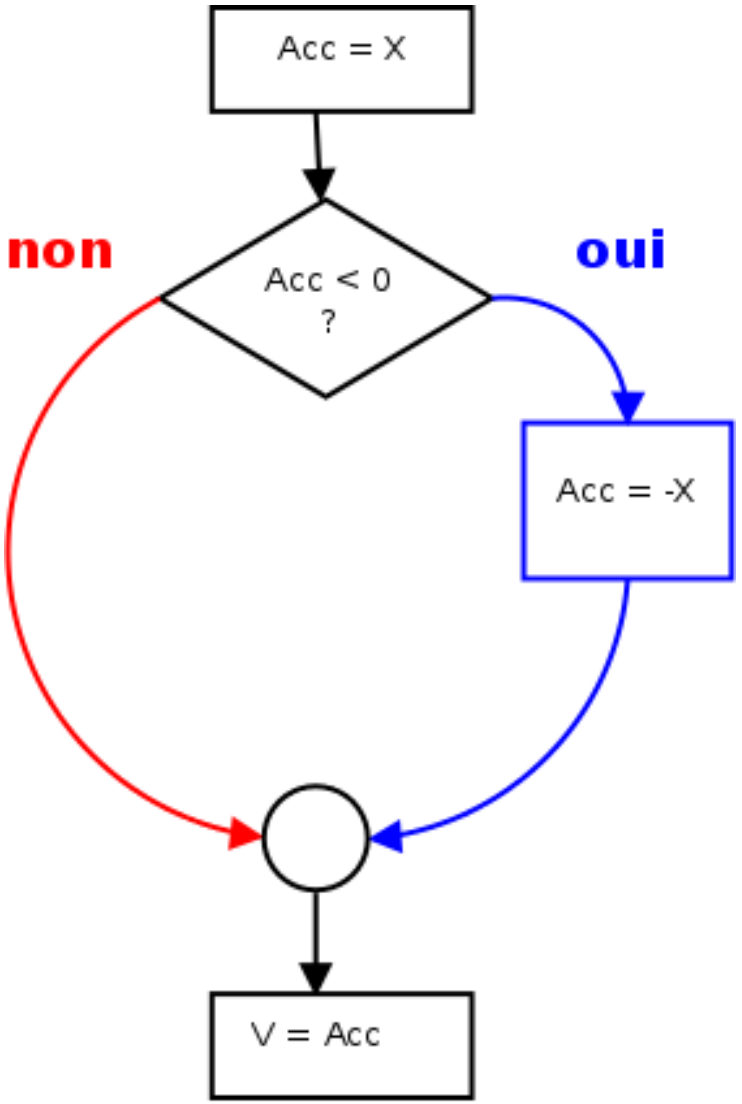




Organigramme 3/4

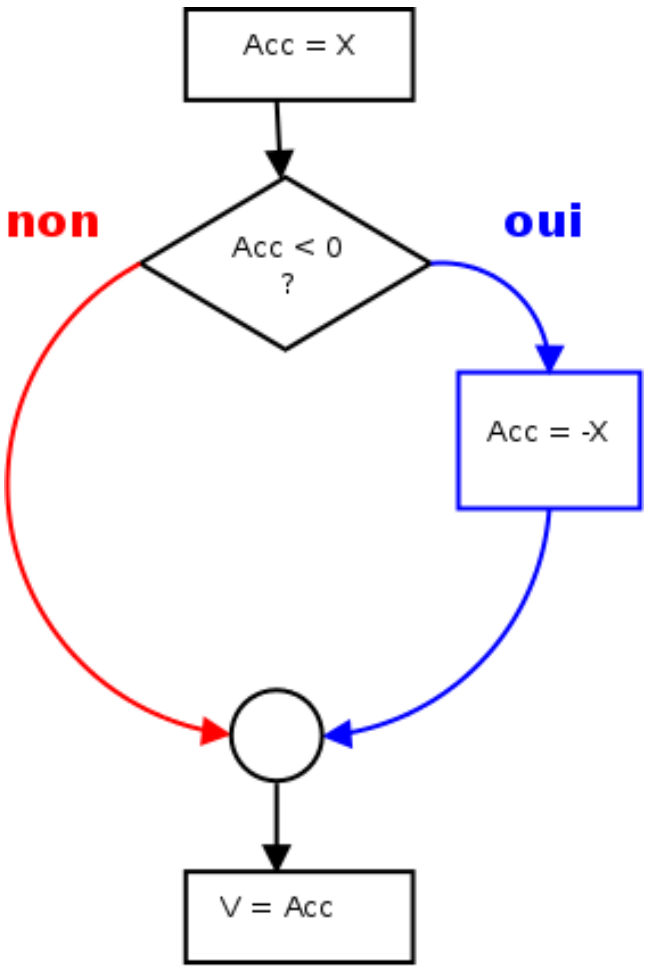
L'instruction "V = Acc" est la dernière des deux branches, on peut la "factoriser" :

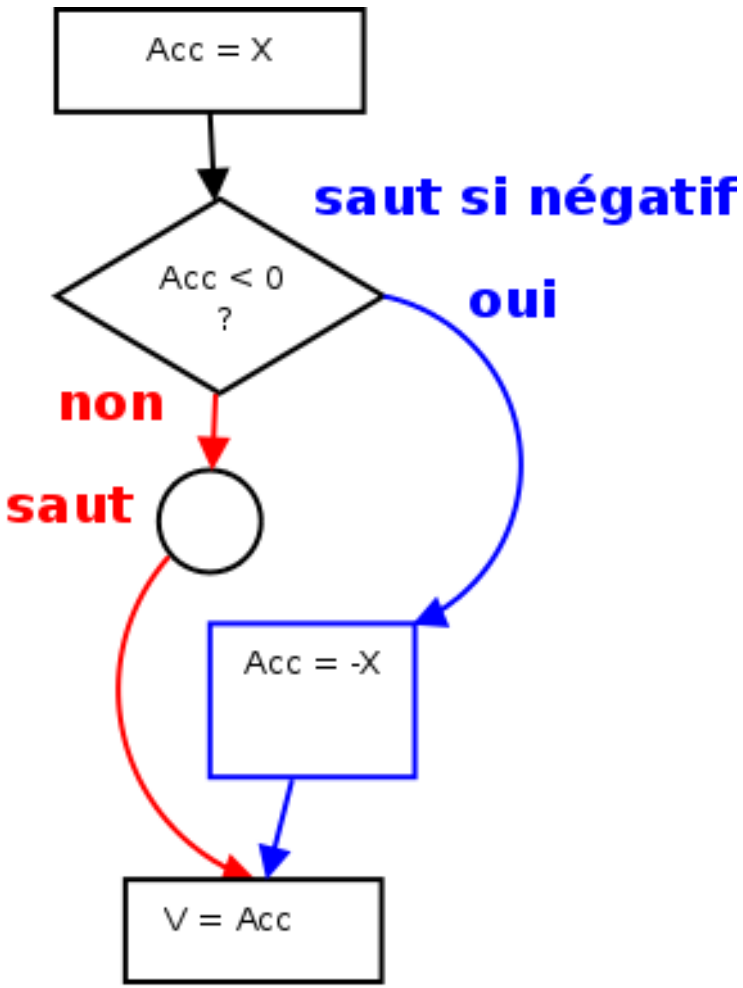




Organigramme 4/4

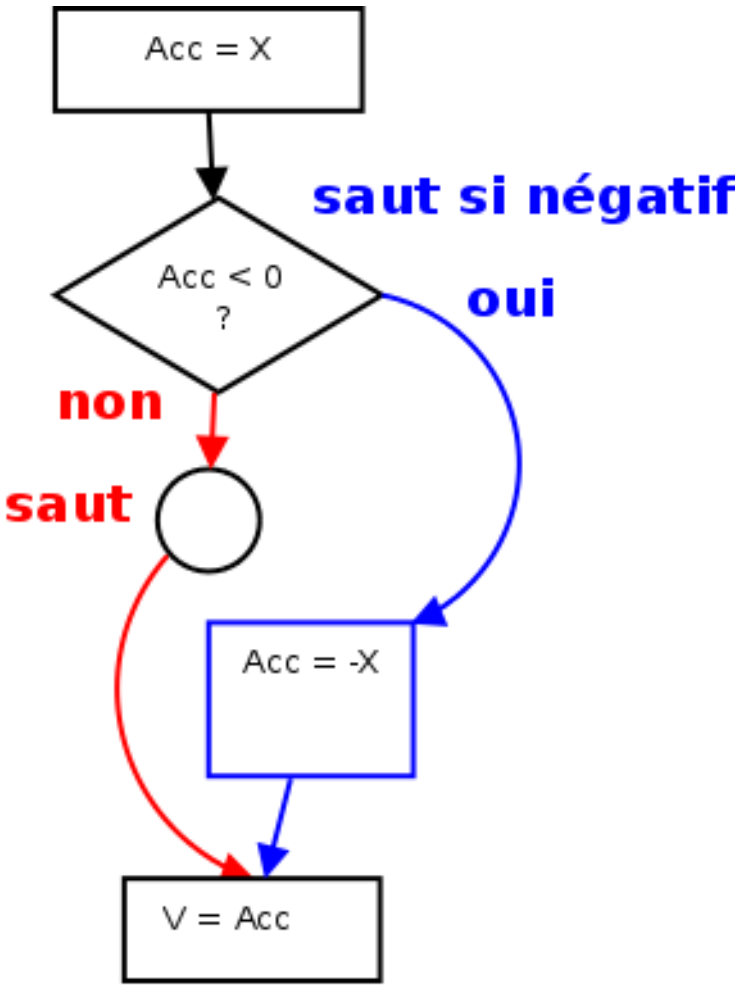
Si le contenu de l'accumulateur est négatif, l'exécution continue en séquence, il faut alors sauter à la fin.





de l'organigramme au programme

Il ne reste plus qu'à traduire




```
      load  X
      jneg    negatif
      jmp       fin
negatif
      loadi   0
      sub       X
fin
      store   V
```

Ca parait simple...

Commentaires

Quelques commentaires sont indispensables pour comprendre rapidement la structure du code

```
    load X          # Acc <- X
    jneg  negatif
    jmp    fin

negatif             # si Acc < 0 alors
    loadi  0        #       Acc <-  -X
    sub    X        #

fin
    store  V        # V <- Acc
```

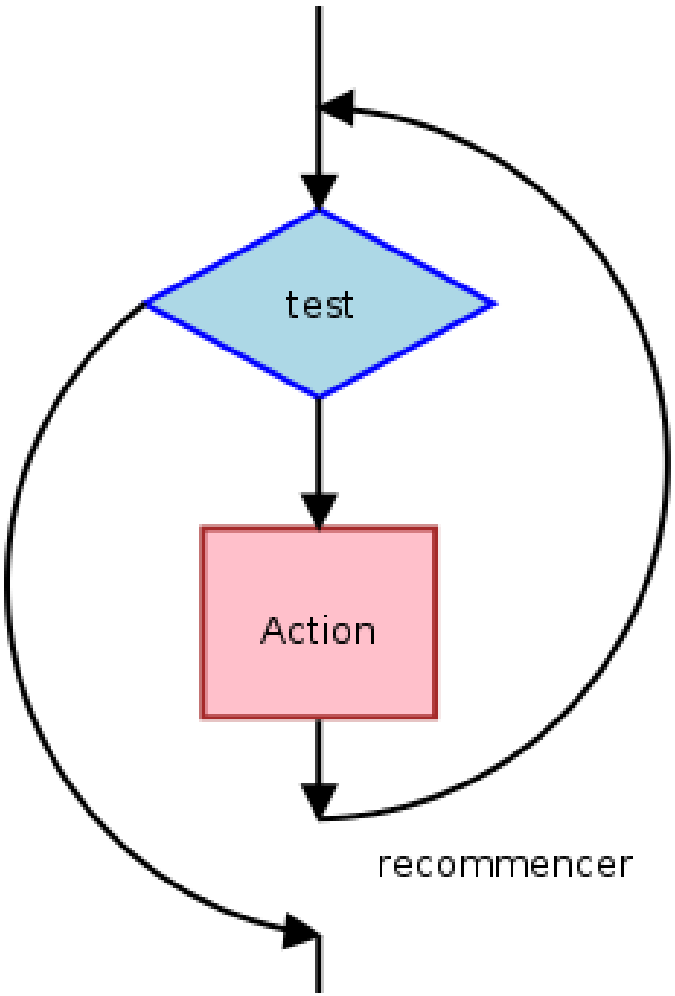
Exercices

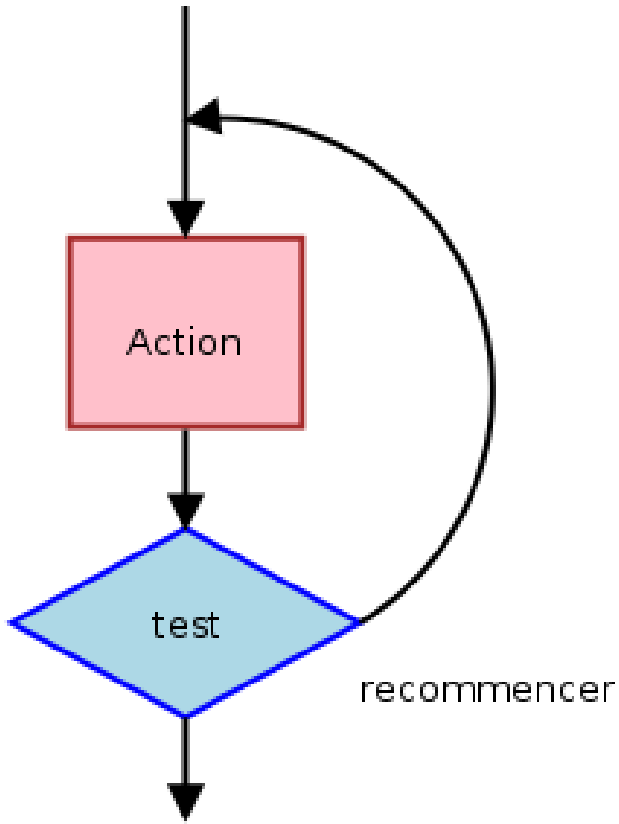
1. calcul du maximum M de deux nombres A et B
2. ordonner deux nombres A et B.

Indication : comparer, c'est étudier la différence...

Faire des boucles

Sous forme d'organigrammme, deux formes :





Avec test en tête (boucle "tant-que") ou test en fin (boucle "répéter").

Exemple : la somme des entiers de 1 à N

Algorithme

```
donnée      N nombre,
résultat    S nombre
variable    K nombre

début
    S = 0
    K = 1
    tant que K <= N
      faire
        |     S = S + K
        |     K = K + 1
fin
```

Pseudo-instructions

Séquences d'affectations + sauts conditionnels ou inconditionnels

Algorithme

```
début
    S = O
    K = 1
    tant que K <= N
     faire
        |    S = S + K
        |    K = K + 1
fin
```

Pseudo-code

```
      S = 0
      K = 1
BOUCLE
      si K > N aller à SUITE
      S = S + K
      K = K + 1
      aller à BOUCLE

SUITE
      ...
```

Le test revient à étudier le signe de la différence N-K.

Code assembleur

Le pseudo-code figure en commentaires

```
   loadi 0      # S = 0
   store S

   loadi 1      # K = 1
   store K

BOUCLE          # si K > N
   load  N
   sub   K
   jneg  SUITE #    aller à suite

   load  S      # S = S + K
   add   K
   store S

   loadi  1     # K = K + 1
   add    K
   store  K
   jmp    BOUCLE

SUITE
   halt 0

# variables

N   word   5
K   word   0
S   word   0
```

Exercices sur les boucles

Facile

1. programme qui multiplie deux valeurs (additions successives)
2. programme qui divise deux valeurs (soustractions successives) et fournit le quotient et le reste.

À la maison

1. programme qui calcule la factorielle d'un nombre.
2. programme qui trouve le plus petit diviseur non trivial d'un nombre (plus grand que 1).

Bilan d'étape

Bravo !

- vous maîtrisez maintenant 6+3 = 9 instructions sur 13
- vous savez les employer pour écrire des programmes avec
  - des affectations
  - des décisions
  - des boucles

Chargement/rangement indirect

Deux nouvelles instructions :

rangement	indirect	loadx adresse
rangement	indirect	storex adresse

qui réalisent des chargements /rangements indirects, à une adresse indiquée par une variable.

Elles nous permettront d'utiliser

- des tableaux
- des pointeurs
- ...

Exemple

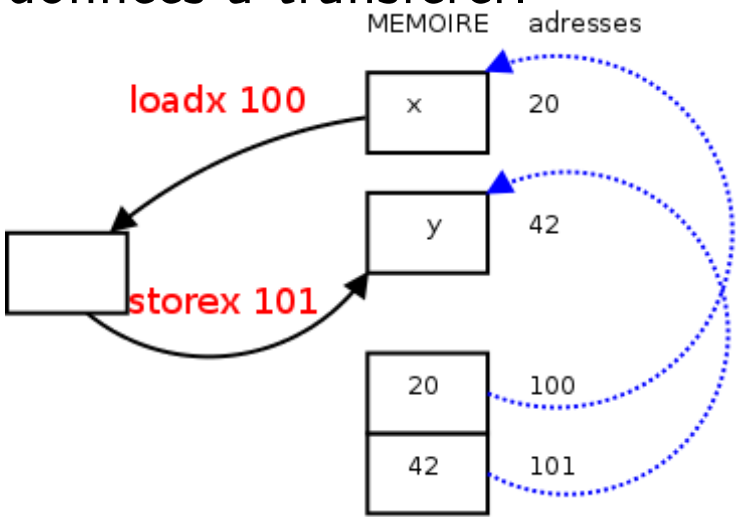
```
loadi    20
store    100
loadi    42
store    101

loadx    100
storex   101
```

copie le mot d'adresse 20 à l'adresse 42, comme le ferait

```
load     20
store    42
```

Les mots d'adresse 100 et 101 sont utilisés comme pointeurs vers les données à transférer.



Ils contiennent l'adresse des données effectives : les mots d'adresses respectives 20 et 42.

Synthèse : Les types d'opérandes

les trois instructions de chargement remplissent l'accumulateur avec un opérande différent :

- loadi *constante* : la constante figurant dans l'instruction (opérande immédiat)
- load *adresse* : la donnée située en mémoire, à l'adresse indiquée (opérande direct)
- loadx *adresse* : la donnée pointée par la valeur à l'adresse indiquée (opérande indirect).

```
loadi valeur          acc =          valeur
load  adresse         acc =     Mem[adresse]
loadx adresse         acc = Mem[Mem[adresse]]
```

Illustration

- loadi (immédiat) charge l'adresse d'une variable (valeur figurant dans l'instruction).
  Exemple `002B = loadi 42`
  signifie : `Acc = 42`
- load (direct) charge son contenu (contenu de la case mémoire).
  Exemple `102B = load 42`
  signifie : `Acc = Mem[42]`
- loadx (indirect) charge la donnée qu'elle pointe (indirection)
  Exemple `202B = loadx 42`
  signifie : `Acc = Mem[Mem[42]]`

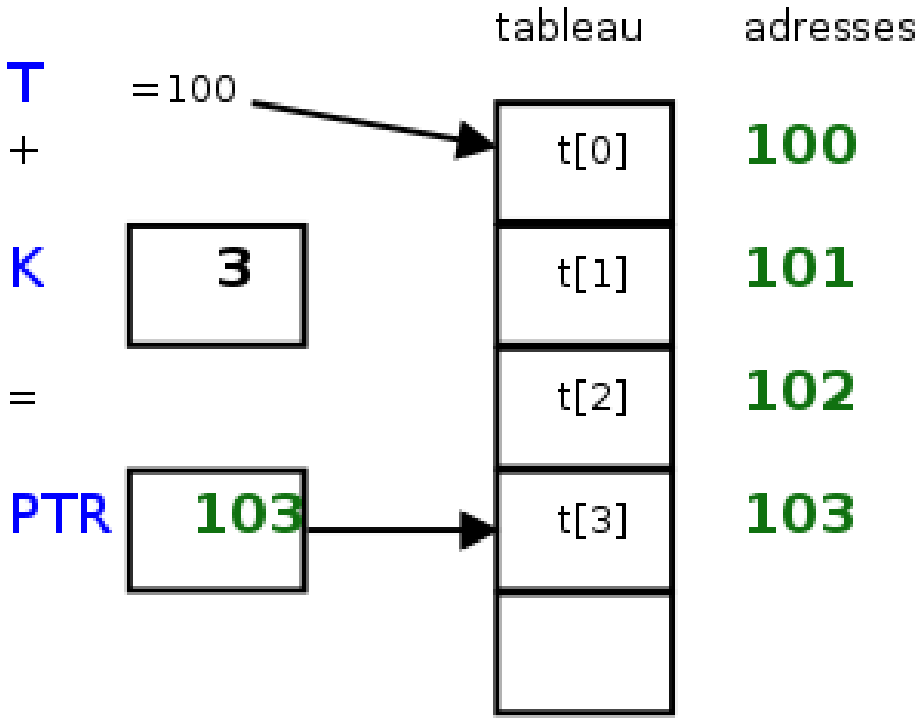
Tableaux

Soit T un tableau (indicé à partir de 0) qui commence à l'adresse 100

Pour accéder au K-ième élément de T, on ajoute

- l'adresse de base du tableau 100
- la valeur de l'indice K

ce qui donne l'adresse de l'élément `T[K]`

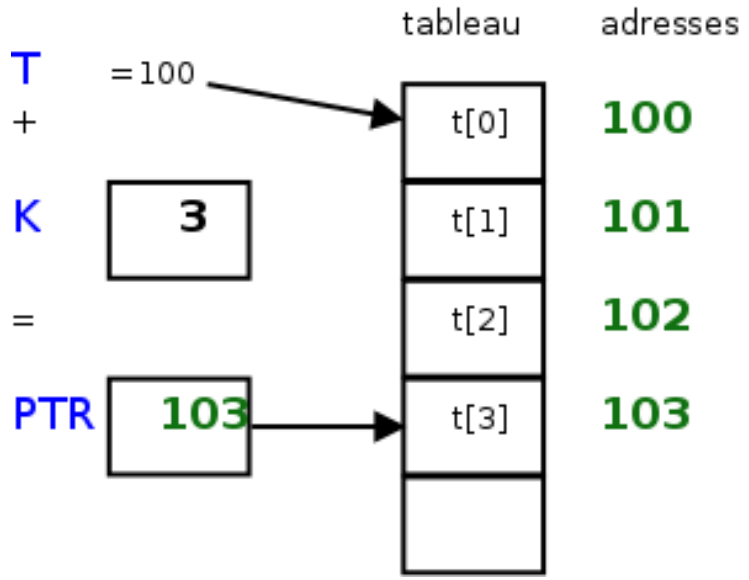


Rangée dans un pointeur `PTR`, cette valeur permet d'accéder à `T[K]` par indirection.

Accès à un élément de tableau

```
    loadi   T     # adr. de T[0]
    add     K
    store   PTR   # adr. de T[K]

    loadx   PTR   # acc = t[k]
    ...
    storex PTR    # t[k] = acc
    ...
K     word    3
PTR  word    0
T     word    11
      word    22
       ..
```



Exemple : somme des éléments d'un tableau

Algorithme en pseudo-code

```
S = 0
K = 0
tant que K != 10
faire
   |   S = S + T[K]
   |   K = K + 1
```

La programmation de la boucle n'a plus de secret pour vous
Il reste à réaliser `S = S + T[K]` :

```
    loadi   T
    add     K
    store   PTR

    loadx   PTR
    add     S
    store   S
```

Exemple : somme des éléments d'un tableau

```
        loadi  0
        store  S
        store  K
BOUCLE
        loadi  10          /  loadi  T
        sub    K          /   add    K
        jzero  FIN       /    store  PTR
        ...........     /     loadx  PTR
        loadi  1         \    add    S
        add    K          \   store  S
        store  K
        jmp    BOUCLE
FIN
        halt   0
```

Exercices

Simples

1. Remplissage d'un tableau avec les entiers de 0 à 9
2. Copie d'un tableau dans un autre
3. Maximum des éléments d'un tableau

Un peu plus longs...

1. Tri par sélection
2. Tri par insertion

Bilan d'étape

Bravo !

- vous maîtrisez maintenant 11 instructions sur 13
- et vous savez les employer pour écrire des programmes qui manipulent des tableaux et des pointeurs.
- pour finir, nous allons voir l'appel et le retour de sous-programmes.

Appel et retour

Les deux dernières instructions :

saut	indirect	jmpx adresse
rangement	indirect	storex adresse

servent à réaliser des sous-programmes :

- jmpx adresse fait aller à l'instruction pointée par le contenu de la case mémoire indiqués.
  Exemple, si le mot d'adresse 100 contient 25, un `jmpx 100` fait aller à 25.
- call appelle un sous-programme en
  - sauvegardant l'adresse de l'instruction suivante à l'adresse indiquée
  - poursuivant l'exécution à l'adresse + 1.

  En effet, un sous-programme commence par un mot réservé, qui contiendra l'adresse de retour, suivi par le code.
  Un exemple ?

Exemple de sous-programme

Séquence d'appel

```
    loadi   X
    call    DOUBLER
    store   XX
    ...
```

Le sous programme
(multiplie l'accumulateur par 2)

```
DOUBLER
   word 0  # adr. retour
   store TMP
   add   TMP
   jmpx  DOUBLER # retour
      ...
```

Cette manière de faire les appels était utilisée dans quelques machines (PDP/1, PDP/4, HP 1000....)

Passage de paramètres

Le passage de paramètres est une affaire de conventions.
Exemple : la fonction qui calcule le maximum de deux nombres
On peut décider que les paramètres seront fournis

- dans deux variables MAXP1 et MAXP2
- ou au sommet d'une pile d'exécution (tableau en fin de mémoire)

et que le résultat sera

- présent dans l'accumulateur
- présent dans une variable MAXRESULTAT
- placé sur la pile

Sans parler de passage par référence...

Exercices

Ecrivez

- un sous-programme de multiplication
- une fonction factorielle.
- un sous-programme de division,
- un programme qui teste si un nombre est premier.
- un programme qui remplit un tableau avec les 20 premiers nombres premiers

Conclusions

- Un jeu d'instructions très simple suffit à la programmation.
- Les langages de programmation ont hérité des concepts des premiers ordinateurs
  - instructions qui modifient les données contenues dans des "cases" : c'est la programmation impérative
  - indirection : pointeurs
- Des notions un peu délicates (comme les tableaux et les pointeurs en C/C++) se comprennent plus facilement quand on sait ce qui se passe dans la machine.

La suite

- La programmation en langage d'assemblage sera étudiée en détail avec des processeurs modernes (RISC à 3 registres).
  - plus d'instructions
  - moins fastidieux à utiliser
- Ce cours continue avec l'initiation au langage C.