Architectures matérielles et robotique, systèmes et réseaux

Jean Dominique CECCALDI - Philippe CLOUP -Marc ELDIN - Luc VINCENT

Contenus	Capacités attendues	Commentaires
Modèle d'architecture séquentielle (von Neumann)	Distinguer les rôles et les caractéristiques des différents constituants d'une machine. Dérouler l'exécution d'une séquence d'instructions simples du type langage machine.	La présentation se limite aux concepts généraux. On distingue les architectures monoprocesseur et les architectures multiprocesseur. Des activités débranchées sont proposées. Les circuits combinatoires réalisent des fonctions booléennes.

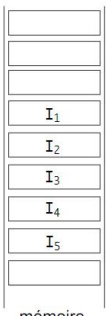
Situation dans la progression

Principe de fonctionnement d'un ordinateur

Modèle de Von Neumann

microprocesseur

Il s'agit d'un programme :
- charger a dans le registre r1 — Instruction 1
- calculer r1 = r1 * 2 — Instruction 2
- charger b dans le registre r2 — Instruction 3
- calculer r1 = r1 + r2 — Instruction 4
- ranger r1 dans c — Instruction 5

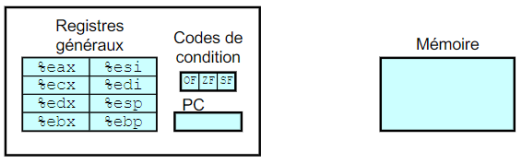


mémoire

Présentation du processeur Y86

Une version ultra-simplifiée des processeurs Intel x86

- Version « pédagogique » élaborée par Carnegie Mellon University
  - https://csapp.cs.cmu.edu
- Jeu d'instructions simplifié inspiré des processeurs x86 32 bits
  - Langage = assembleur Y86
- 8 registres généraux 32 bits
  - %eax, %ecx, …
- 1 compteur ordinal (Program Counter)
- De la mémoire RAM



Extrait du poly de Raymond Namyst (DIU)

Thématique de la séquence 1

Exécuter un programme simple d'addition.

Connaissances mises en fonctionnement

Contenus	Capacités attendues
Écriture d'un entier positif dans une base $b \geqslant 2$	Passer de la représentation d'une base dans une autre.
Représentation binaire d'un entier relatif	Utiliser le complément à 2.

Connaissances visées

Contenus	Capacités attendues
Modèle d'architecture séquentielle (von Neumann)	Distinguer les rôles et les caractéristiques des différents constituants d'une machine. Dérouler l'exécution d'une séquence d'instructions simples du type langage machine.

Durée 1 heure

Difficultés probables

Outils

- Y86 Simulator

Résolutions de taches

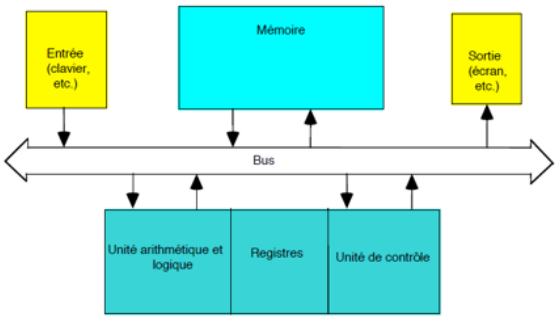
- Interprétation du code objet

Déroulement

- Configuration
  - Travail individuel sur ordinateur
- Support
  - Document pdf
  - Démonstration du programme
- Rôle enseignant
  - Présentation
  - Réponse au questionnement individuel autour de l'usage du logiciel, repérage des valeurs.

La programmation en langage assembleur

La majorité des ordinateurs peuvent être décrits par le schéma suivant.

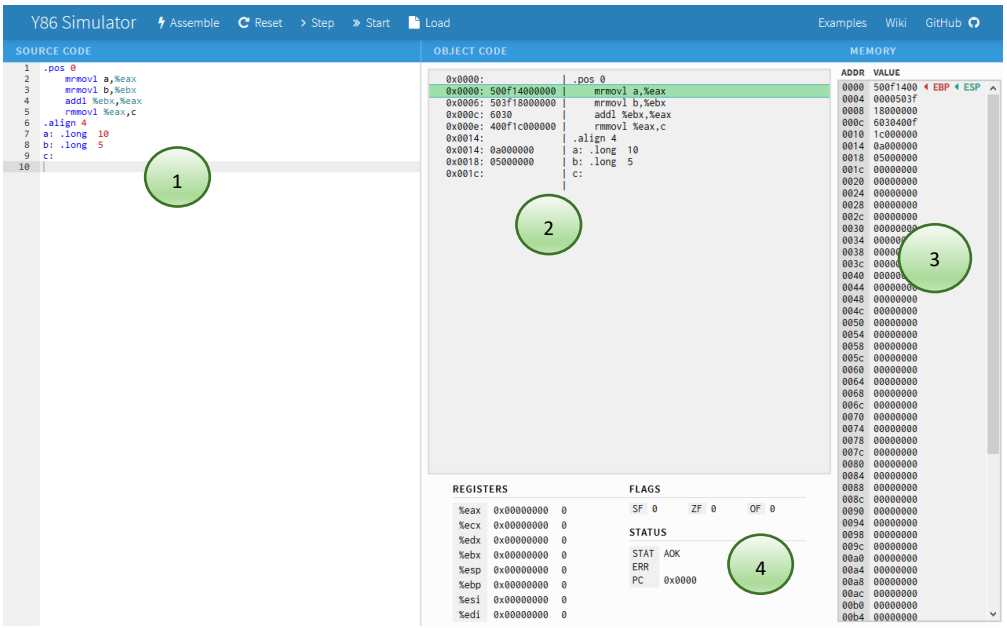


L'unité centrale de traitement exécute les instructions à la suite les unes des autres.

Un programme écrit en en langage évolué (Python) est traduit en langage machine pour pouvoir être exécuté.

```
1  a=10
2  b=5
3  c=a+b
4
```

Le logiciel Y86 va nous permettre de simuler le fonctionnement d'un microprocesseur et d'observer l'exécution de ce programme



Pour cela nous écrirons le programme en langage machine (« assembleur ») dans la zone code source (zone 1). Par appui sur le bouton Assemble ce code est transformé en code objet (zone 2) exécutable par la machine.

L'avancement du programme peut être réalisé instruction par instruction par l'appui sur Step ou dans son intégralité par l'appui sur Start. On pourra replacer le programme en situation initiale par appui sur Reset

La zone 3 montre le contenu de la mémoire et la zone 4 la valeur des registres et du microprocesseur.

Pour illustrer la différence entre langage évolué et langage d'assemblage, voici le programme équivalent en langage d'assemblage :

```
1  .pos 0
2      mrmovl a,%eax
3      mrmovl b,%ebx
4      addl %ebx,%eax
5      rmmovl %eax,c
6  .align 4
7  a: .long  10
8  b: .long  5
9  c:
10
```

Assembler le programme et repérer dans le code objet l'adresse mémoire qui contient la valeur 10

OBJECT CODE

```
0x0000:                | .pos 0
0x0000: 500f14000000   |     mrmovl a,%eax
0x0006: 503f18000000   |     mrmovl b,%ebx
0x000c: 6030           |     addl %ebx,%eax
0x000e: 400f1c000000   |     rmmovl %eax,c
0x0014:                | .align 4
0x0014: 0a000000       | a: .long  10
0x0018: 05000000       | b: .long  5
0x001c:                | c:
                       |
                       |
```

La case mémoire d'adresse 0x0014 contient la valeur 10

Exécuter la première instruction. Que s'est-il passé pour le microprocesseur ?

REGISTERS

%eax	0x0000000a	10
%ecx	0x00000000	0
%edx	0x00000000	0
%ebx	0x00000000	0
%esp	0x00000000	0
%ebp	0x00000000	0
%esi	0x00000000	0
%edi	0x00000000	0

FLAGS

SF 0 ZF 0 OF 0

STATUS

STAT AOK
ERR
PC 0x0006

Le registre %eax a pris la valeur 10 Le registre PC 0x0006 et pointe la prochaine instruction.

Exécuter le code pas à pas et repérer quelle est l'instruction qui réalise l'opération d'addition. Où se trouve le résultat de l'opération ?

C'est l'instruction **`addl %ebx,%eax`** qui réalise l'opération d'addition. Après cette instruction le résultat se trouve dans le registre %eax

A la fin du programme repérer dans quelle case mémoire se trouve le résultat.

Le résultat se trouve en `0x001c`

Choisir pour a la valeur hexadécimale `0xfedcba98` et pour b la valeur 1. Interpréter le contenu de la case mémoire ou on trouvera le résultat.

Le code `0x`fedcba99 est stocké avec les poids faibles en premier

MEMORY

ADDR	VALUE
0000	500f1400
0004	0000503f
0008	18000000
000c	6030400f
0010	1c000000
0014	98badcfe
0018	01000000
001c	99badcfe
0020	00000000

Modifier le code source pour réaliser une opération c=b-a avec a = 10 et b = 5 à l'aide de l'instruction **subl** puis interpréter le résultat obtenu en case `0x001c` après exécution du programme.

SOURCE CODE

```
1  .pos 0
2      mrmovl a,%eax
3      mrmovl b,%ebx
4      subl %eax,%ebx
5      rmmovl %ebx,c
6  .align 4
7  a: .long  10
8  b: .long  5
9  c:
10
```

En mémoire on peut lire 0xfbffffff ce qui correspond au mot résultat 0xfffffffb

Ce nombre est bien un nombre négatif car son bit de poids fort est à 1.

Pour les 4 bits de poids faible en binaire b s'écrit 1011

On peut calculer le code complément à 2 : 0x00000005

C'est donc bien le code de -5 qui est présent en mémoire.

0018	05000000
001c	fbffffff
0020	00000000

Thématique de la séquence 2

Etablir un lien entre la manipulation d'un tableau en python et son implantation en mémoire.

Programme python

Soit le programme python « édité » avec http://pythontutor.com/

Exécuter ce programme et expliquer son fonctionnement :

Sur quoi pointe la flèche bleue ?

Dessiner une flèche qui correspond à la case modifiée par l'exemple.

Programme identique en Assembleur

Le programme identique au précédent codé sur « Y86 » correspondant est le suivant :

SOURCE CODE

```
1  .pos 0
2  Init:
3   irmovl 3, %eax
4   mrmovl indice, %edi
5   rmmovl %eax, tab(%edi)
6
7
8   halt
9
10 .align 4
11 indice: .long 8
12 tab:    .long 12
13         .long 10
14         .long 8
15         .long 0
16
```

```
0x0000:              | .pos 0
0x0000:              | Init:
0x0000: 30f003000000 |  irmovl 3, %eax
0x0006: 507f14000000 |  mrmovl indice, %edi
0x000c: 400718000000 |  rmmovl %eax, tab(%edi)
                     |
                     |
0x0012: 00           |  halt
                     |
0x0013:              | .align 4
0x0014: 08000000     | indice: .long 8
0x0018: 0c000000     | tab:    .long 12
0x001c: 0a000000     |         .long 10
0x0020: 08000000     |         .long 8
0x0024: 00000000     |         .long 0
                     |
```

ADDR	VALUE
0000	30f00300
0004	0000507f
0008	14000000
000c	40071800
0010	00000000
0014	08000000
0018	0c000000
001c	0a000000
0020	03000000
0024	00000000
0028	00000000
002c	00000000
0030	00000000
0034	00000000
0038	00000000

Exemple de questions posées

- Identifier la place du tableau en mémoire ?
- Repérer l'adresse mémoire du 1er élément du tableau, puis celle du second élément.
- Quelle étiquette permet de pointer sur la case à modifier ?
- En exécutant le programme pas à pas, que contient le registre **%eax** après exécution de la ligne 3 ?
- Quel est le rôle des lignes 4 et 5 ? On justifiera sa réponse en exécutant le programme pas à pas, en notant la valeur des registres.

Exercices Bilan

Modification simple du programme assembleur

Modifier le programme :

- Pour avoir 6 cases mémoire, initialisées à [6, 7, 8, 9, 10, 11] (ils seront codé en hexadécimal)
- Modifier le programme pour qu'en fin d'exécution la case 4 soit à 0x3.

Lecture d'une case

- Modifier le programme précédent pour qu'il accède à la case 5 du tableau, elle sera placée dans le registre %ecx, puis la valeur sera placée à l'adresse 128 (repérée par le label « **destination** »).

Autres modifications

On prend 10 cases mémoires par exemple et on demande :

1. D'en faire la somme (utilisation d'une boucle qui va lire tous les éléments en les ajoutant) et de ranger le résultat dans un registre ;
2. D'en donner le maximum et le minimum : mettre 0 dans un registre et le comparer à tous les autres ; modifier ce registre chaque fois qu'un nombre est supérieur ;
3. D'écrire une fonction qui renvoie le maximum ou le minimum ;
4. De les trier par ordre croissant en utilisant l'algorithme du trie à bulles à l'aide la fonction précédente ;

Lien avec l'Arduino et utilisation d'une matrice 8x8

On utilise une matrice 8x8 de Leds pour afficher des lettres correspondants à des nombres écrits en binaire.

Coco_Matrice_8x32_projet §

```
#include "LedControl.h" //Pour controler la matrice 8x8
//12=DataIn,11=CLK,10=LOAD nommé CS
LedControl lc=LedControl(12,11,10,1);
delaytime=1000;
void setup() {
  lc.shutdown(0,false);
  lc.setIntensity(0,1);
  lc.clearDisplay(0);
}
```

Coco_Matrice_8x32_projet | Arduino 1.8.9

Fichier Édition Croquis Outils Aide

Coco_Matrice_8x32_projet §

```
void writeArduinoOnMatrix() {
  byte a[5]={B01111110,B10001000,B10001000,B10001000,B01111110};
  byte r[5]={B00111110,B00010000,B00100000,B00100000,B00010000};
  byte d[5]={B00011100,B00100010,B00100010,B00010010,B11111110};
  byte u[5]={B00111100,B00000010,B00000010,B00000100,B00111110};
  byte i[5]={B00000000,B00100010,B10111110,B00000010,B00000000};
  byte n[5]={B00111110,B00010000,B00100000,B00100000,B00011110};
  byte o[5]={B00011100,B00100010,B00100010,B00100010,B00011100};
```

Coco_Matrice_8x32_projet §

```
  lc.setRow(0,0,a[0]);
  lc.setRow(0,1,a[1]);
  lc.setRow(0,2,a[2]);
  lc.setRow(0,3,a[3]);
  lc.setRow(0,4,a[4]);
  delay(delaytime);
  lc.setRow(0,0,r[0]);
  lc.setRow(0,1,r[1]);
  lc.setRow(0,2,r[2]);
```