Université de Valenciennes et du Hainaut-Cambrésis

ARCHITECTURE DES ORDINATEURS

UVHC – ISTV – Licence 1 Informatique

Rabie Ben Atitallah

rabie.benatitallah@univ-valenciennes.fr

PROGRAMME

Séance 1 Un cours d'architecture pour des informaticiens

Séance 2 De l'électronique à l'informatique

Séance 3 Du binaire à l'information

Séance 4 Notion de circuit logique

Séance 5 Un modèle d'exécution : Von Neumann

Séance 6 Instructions machine

Séance 7 Comment mémoriser une donnée ?

Séance 8 Instruction et ordonnancement

CHAP. 6 : Instructions Machines

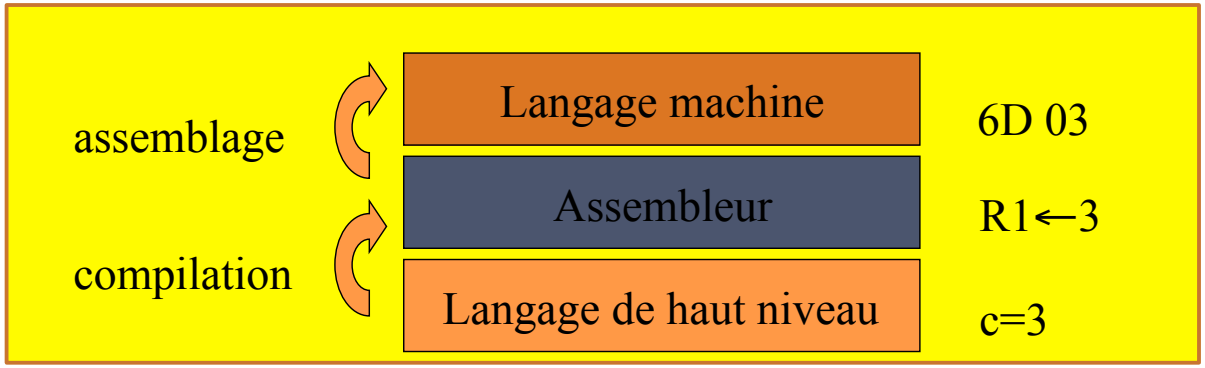
- Instructions
- Codage des instructions
- Modes d'adressage
- Branchement
- Pile
- Sous-routines

Objectifs

- Découvrir les instructions élémentaires du microprocesseur
- Comment on les réalise à l'intérieur du µP…
- Comment on contrôle le flot des instructions
  - Branchement
  - Sous routines
  - Passage de paramètres

Du langage à la machine

- **Programme = suite d'instructions**



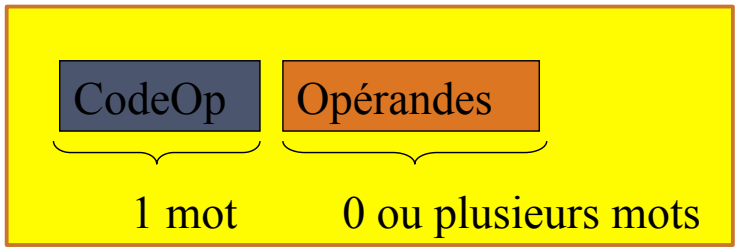
- **Trois types d'instructions**
  - **De rangement**
  - **De calcul**
  - **De branchement**

EXEMPLES D'INSTRUCTIONS MACHINE

R1←4	MOV R1,4 LD R1,4	Mettre la valeur 4 dans le registre R1
R1←R2	MOV R1,R2 LD R1,R2	Mettre la valeur de R2 dans le registre R1
R1←MEM(1515)	MOV R1,(1515)	Copie la valeur stockée en mémoire à l'@ 1515 dans R1
R1←R1+2	ADD R1,2	Additionne 2 à la valeur de R1 et range dans R1
R1←R1+1	INC R1	Incrémente de 1 la valeur de R1
R2←R2-R4	SUB R2,R4	Soustraie à R2 la valeur de R4 et range dans R2

CODAGE DES INSTRUCTIONS

- **Structure :**



- **On représente les instructions par des chaines de caractères => ce sont donc des chaines de bits!!!**
  - **MOV R1, 4 2 mots = 89 04 en hexadécimal**
  - **ADD R2, 3 2 mots = B2 03**

CODAGE DES INSTRUCTIONS

- **MOV R1, 4**
  - 89 04
  - 1000 1001 0000 0100
  - 10001 001 0000 0100
  - MOV R1 4

- **ADD R2, 3**
  - B2 03
  - 1011 0010 0000 0011
  - 10110 010 0000 0011
  - ADD R2 3

Chaque processeur possède son propre codage des instructions

INSTRUCTIONS EN MÉMOIRE

- **Von Neumann: on range les instructions dans la mémoire les unes derrière les autres…**
- **Si on range en mémoire à l'adresse 1515**

  MOV R1, 4 ADD R2, 3 il faut 4 mots mémoire de 8 bits ou 2 mots de 16 bits….

  - 1515 = 89
  - 1516 = 04
  - 1517 = B2
  - 1518 = 03

  1515 = 8904

  1516= B203

COMBIEN D'OPÉRANDES?

- **3 opérandes**

- **2 opérandes**

- **1 opérandes**

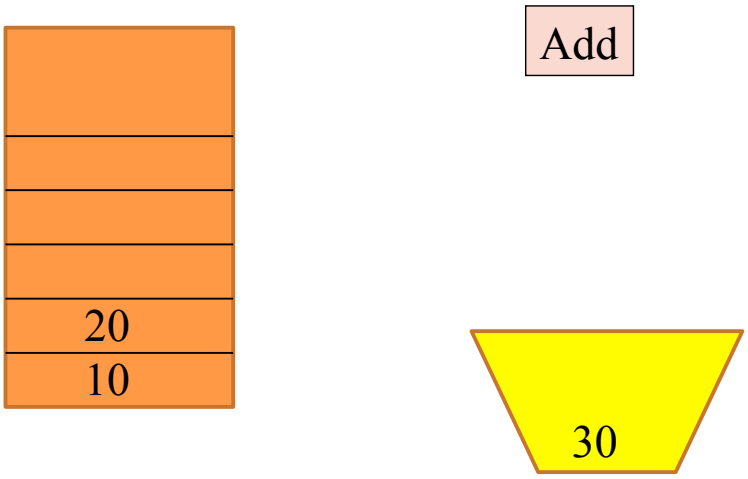
- **0 opérande**

Exemples d'instruction

- **ADD R2, R3, R4** R3 + R4 => R2
- **ADD R2, R3** R2 + R3 => R2
- **ADD R2** Acc + R2 => Acc
- **ADD** Sur la pile

Les machines à pile

- **On utilise une pile pour ranger les opérandes**
- **On passe en notation post-fixée (Polonaise)**



Modes d'adressage

- **Valeur immédiate**
  - MOV R1, 3E
- **Adressage direct registre**
  - MOV R1, R2
- **Adressage direct mémoire**
  - MOV R1, (1515)
- **Adressage indirect registre**
  - MOV R1, (R2)
- **Adressage indirect indexé**
  - MOV R1, (R3, R2)

D'AUTRES ADRESSAGES

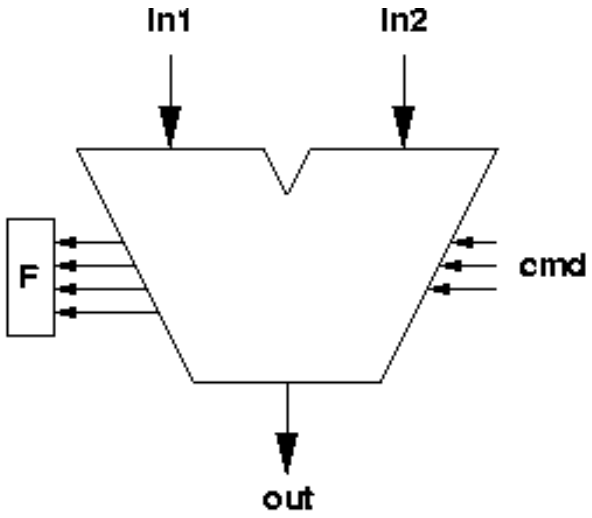
- **Post incrémenté**
  - MOV R1, (R4)+
- **Pré décrémenté**
  - MOV R1, -(R4)
- **Avec déplacement**
  - MOV R1, 10(R1)
- **Indirect indexé avec déplacement**
  - MOV R1, 10(R2, R3)

Instructions de branchement

- **Trois types de branchement**
  - Branchements simples
  - Branchements conditionnels
  - Branchements sous-routines
- **Deux types de références**
  - Absolu ou relatif
  - En relatif, on ne donne pas une adresse mais un déplacement…
- **Branchements simples :**
  - JMP 1789 ou JP 1789
  - C'est équivalent à un MOV dans PC

INSTRUCTIONS DE BRANCHEMENT

- **Branchement conditionnels**
  - **Sur quoi porte la condition ?**
  - **Lorsque l'on fait une opération arithmétique et logique, l'ALU positionne un registre de flags !**



INSTRUCTIONS DE BRANCHEMENT

- **Chaque bit du registre F est un « flag »**
  - **Ex :**
    - **Z le résultat est zéro**
    - **N le résultat est négatif**
    - **V l'opération a engendré un dépassement de capacité**
    - **Etc.**
- **La condition de saut porte sur les flags**
  - **JZ 1789**
    - **Saute à l'adresse 1789 si Z est positionné**

EXEMPLE 1

- En C :

```
if ( cpt == 10 )
    cpt = 0;
else
    cpt++;
```

- **La variable cpt est soit en mémoire soit dans un registre (ici on suppose que c'est dans R3).**

```
1515: CMP R3, 0A
1517: JNZ 1525; partie else
1520: MOV R3, 00
1522: JMP 1526          ; suite du programme
1525: INC R3
1526: …
```

EXEMPLE 2

- En C :

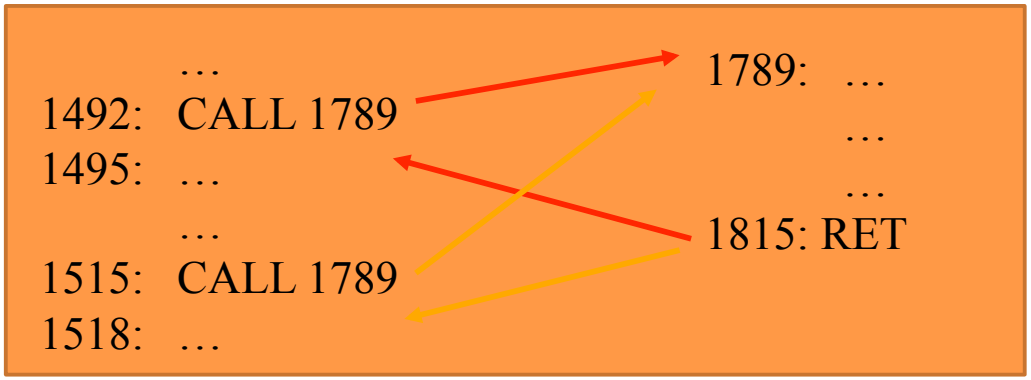
```
for ( i=0 ; i<15 ; i++ )
    a[i] = a[i] + i;
```

- i est dans le registre R1, a est un tableau de bytes dont l'adresse de début est dans R2

```
1515: MOV R1, 00          ; initialisation boucle
1517: MOV R3, R2
1518: CMP R1, 0F          ; début boucle
1520: JGE 1531        ; codé sur 3 octets
1523: MOV R4, (R3)
1524: ADD R4, R1          ; a[i] = a[i] + i
1525: MOV (R3), R4
1526: INC R1
1527: INC R3
1528: JMP 1518
1531: …
```

SOUS-ROUTINES

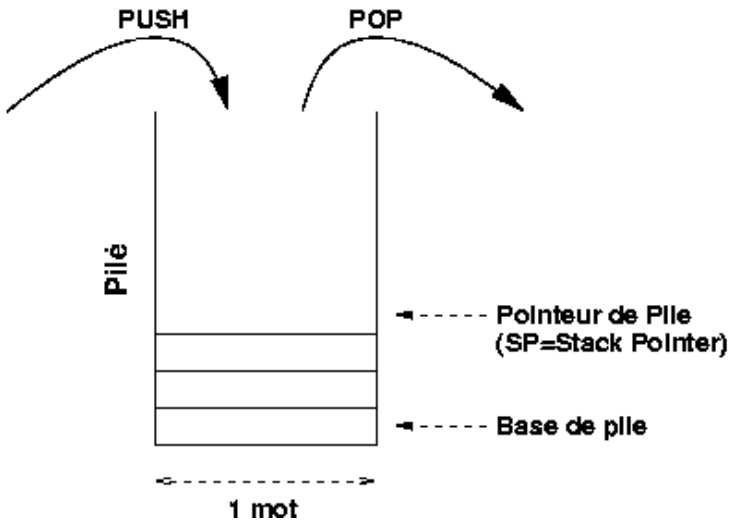
- **Instruction CALL ou JSR**
  - **Ce sont des instructions de branchement**



  - **Pb. Comment savoir si l'on revient en 1495 ou 1518 ?**
  - **Quels sont les registres que l'on peut utiliser dans la sous-routines ?**

ADRESSE DE RETOUR

- On utilise une pile
  - = LIFO
  - Deux instructions :
    - push
    - pop
  - On utilise un registre pointeur de pile SP
    - push → on décrémente SP
    - pop → on incrémente SP



ADRESSE DE RETOUR

- **Après la lecture de l'instruction CALL, PC contient l'adresse de retour**
- **Lors du CALL :**
  - On « pushe » PC
  - On met l'adresse de la sous-routine dans PC
- **Lors du RET**
  - On « pope » l'adresse de retour et on la met dans PC

La sous-routine doit rendre la pile « propre », *i.e.* autant de push que de pop…

Registres utilisables

- **Pour les registres…**
  - **La sous-routine sauvegarde dans la pile les valeurs des registres qu'elle va utiliser**

```
1789:       push R1
            push R2
            push R5
            mov R1, 5
            …
            pop R5
            pop R2
            pop R1
1815:       ret
```

Passage de paramètres

- **Deux possibilités**
  - **Par registres**
    - **Fenêtrage de registres (cours de M1)**
  - **Par la pile**

```
        push R3      ; argument 1
        push R7      ; argument 2
515:    call 1789
        pop R7
        pop R3
```

Obligatoire pour rendre la pile « propre »

Argument 1 et 2 accessibles depuis SP