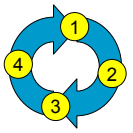
Architecture des Ordinateurs
Partie II : Microprocesseur

2. Instructions machines (suite)

David Simplot
simplot@fil.univ-lille1.fr

Objectifs

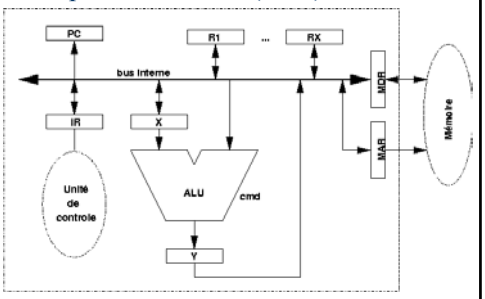
- Voir les instructions élémentaires du microprocesseur
- Comment on les réalise à l'intérieur du µP…
- Quatre phases du µP
- Trois types d'instructions
  - De rangement
  - De calcul
  - De branchement



Au sommaire...

- **Instructions (suite)**
- Sous-routines
- INT/DMA
- Microprogrammation

Instructions (6/14)
Exemple d'exécution (suite)



Instructions (7/14)
Exemple d'exécution (suite)

- Phase 1 – Lire l'instruction
  - PC vaut 1515
  - Mettre PC dans MAR et donner l'ordre de lecture
    - PCout – LDMAR
    - Read – WaitMemory
  - Placer la valeur de MDR dans IR pour que l'instruction soit décodée
    - MDRout – LDIR
  - Pb. Lors de la phase 3, on va avoir besoin de récupérer les paramètres qui sont à PC+1
    - → on anticipe

Instructions (8/14)
Exemple d'exécution (suite)

- Phase 1 (suite)
  - On profite du fait que la lecture en mémoire est lente pour incrémenter PC
  - Dès que l'on est en phase 3, PC pointe vers le premier argument
  - Nb. S'il n'y a pas d'arguments, PC pointe vers l'instruction suivante.
  - « code » réel de la phase 1 :
    - PCout – LDMAR – LDX
    - Read – INCX – LDY
    - Yout – LDPC – WaitMemory
    - MDRout - LDIR

Instructions (9/14)
Exemple d'exécution (suite)

- Phase 2 – décodage de l'instruction
  - Pris en charge par l'UC (unité de contrôle)
- Phase 3 – exécution
  - Deux sous-phases :
    - 3.1 Récupérer les arguments éventuellement
    - 3.2 Exécution
  - 3.1 lecture argument + incrémentation PC
    - PCout – LDMAR – LDX
    - Read – INCX – LDY
    - Yout – LDPC - WaitMemory

Instructions (10/14)
Exemple d'exécution (suite)

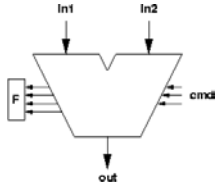
- Phase 3 (suite)
  - 3.1 lecture argument + incrémentation PC
    - PCout – LDMAR – LDX
    - Read – INCX – LDY
    - Yout – LDPC - WaitMemory
  - 3.2 exécuter l'instruction (ici MOV R1,4)
    - R1out – LDX
    - MDRout – ADD – LDY
    - Yout – LDR1
- Phase 4 (préparer l'instruction suivante)
  - Rien pour l'instant ☺

Instructions (11/14)
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 (12/14)
Instructions de branchement (suite)

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



Instructions (13/14)
Instructions de branchement (suite)

- 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 postionné

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, 0
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 1530
1523: MOV R4, [R3]
1524: ADD R4, R1          ; a[i] = a[i] + i
1525: INC R1
1526: INC R3
1527: JMP 1518
1530: …
```


Instructions (14/14)
Modes d'adressage

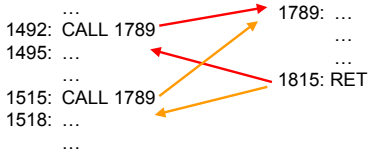
- Valeur immédiate
  - MOV R1, 3E
- Valeur d'un registre
  - MOV R1, R2
- Adressage direct
  - MOV R1,[1515]
- Adressage indirect
  - MOV R1,[R2]
- Adressage indexé
  - MOV R1,[1515+R2]

Au sommaire...

- Instructions
- **Sous-routines**
- INT/DMA
- Microprogrammation

Sous-routines (1/7)

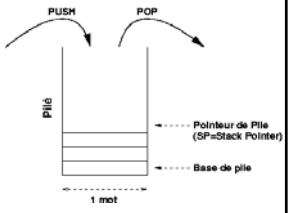
- Instruction CALL ou JSR
  - C'est une instruction de branchement



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

Sous-routines (2/7)
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



Sous-routines (3/7)
Adresse de retour (suite)

- 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…

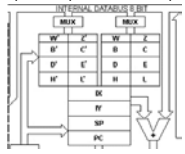
Sous-routine (4/7)
Registres utilisables

- 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
```

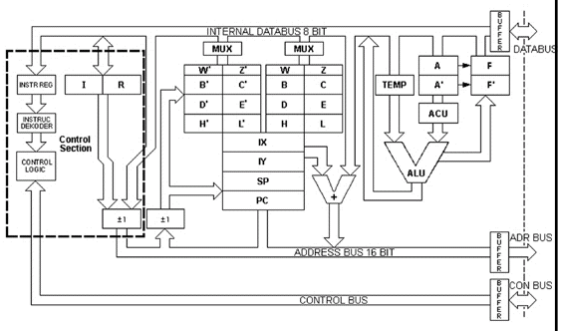

Sous-routine (5/7)
Registres utilisables (suite)

- Autre méthode :
  - Le µP dispose de plusieurs « banques » de registres



Z80
de Zilog

- Pb. de cette méthode :
  - Les paramètres de la routine ne peuvent être passés via les registres
  - Nombre d'appels imbriqués limité par le nombre de banques



Sous-routines (6/7)
Passage de paramètres

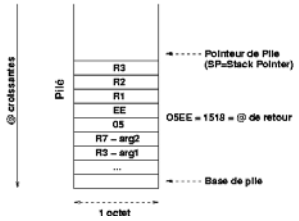
- Deux possibilités
  - Registres
    - Sauf si banques de registres
  - Pile

```
        push R3 ; argument 1
        push R7 ; argument 2
1515:   call 1789
        pop R7
        pop R3

1789:   push R1
        push R2
        push R3
        …
        ret
```

Obligatoire pour rendre la pile « propre »

- Argument 2 en SP+6 - Argument 1 en SP+7



Sous-routines (7/7)
Passage de paramètres (suite)

- Inconvénients des deux techniques
  - Par registres :
    - Nombre de paramètres limité
  - Par pile :
    - Coûteux pour de petites fonctions
    - → technique de compilation *in-lining*