Qroc A23 Architecture des ordinateurs

TTN / FA - 19 février 2009 - 2 heures

Documents joints à ce sujet seuls autorisés - calculette autorisée

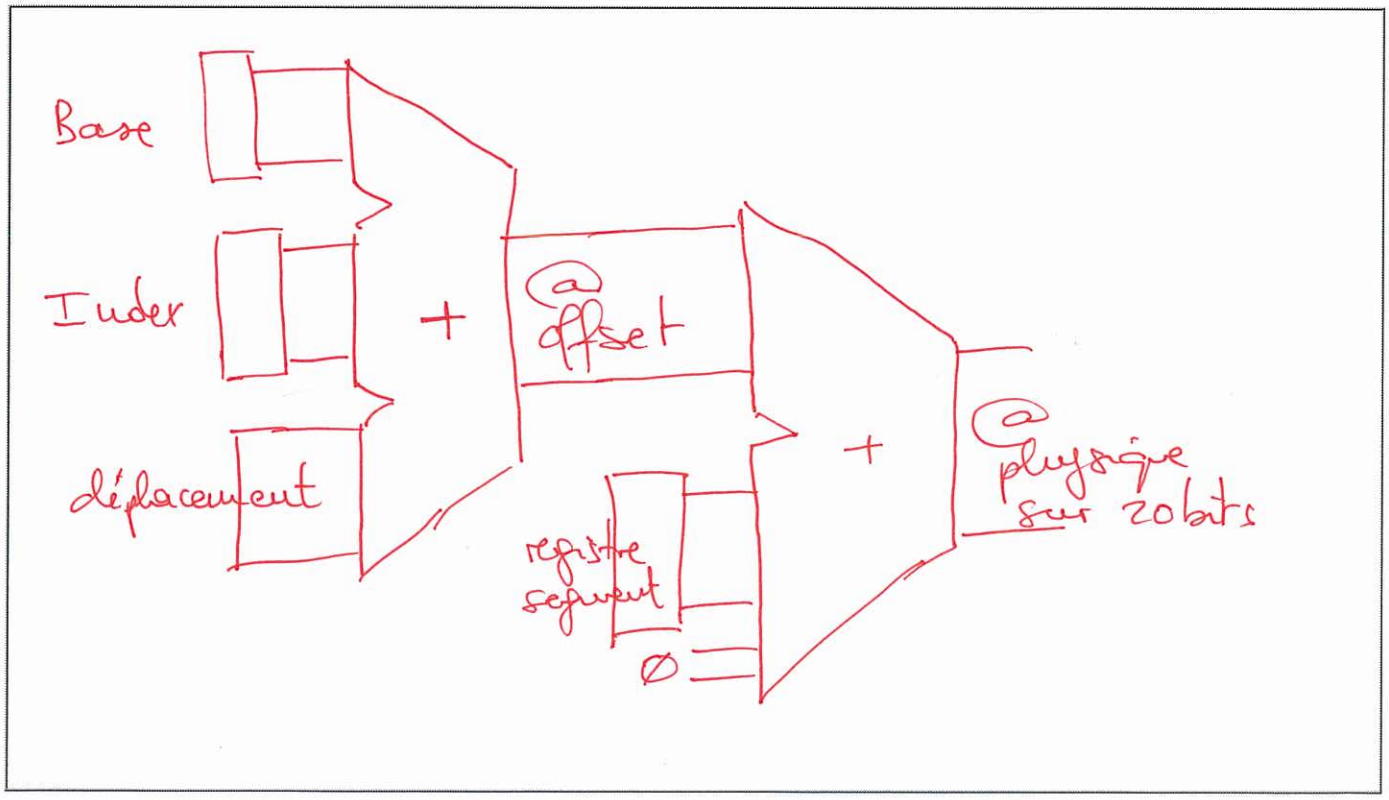
Dans tout le sujet, on considère la plate-forme 16 bits Turbo-C 8086.

Matériel

1. Résumer par un tableau les différents cas de fonctionnement (lecture, écriture, entrée, sortie, repos) du bus de données en fonction des principaux signaux de contrôle du 8086.

$\overline{RD}$	$\overline{WR}$	M/$\overline{IO}$	cas de fonctionnement
0	1	1	lecture
1	0	1	écriture
0	1	0	entrée
1	0	0	sortie
1	1	X	repos

2. Résumer par un schéma à base d'additionneurs le fonctionnement des modes d'adressage direct et indirect du 8086 jusqu'à obtention de l'adresse physique sur 20 bits.



3. Quels sont les types d'information circulant sur le bus de données lors des comportements spontané et programmé d'un processeur ?

spontané : codes machine - programmé : données manipulées par l'instruction

4. Dans quelle entité du microprocesseur sont stockés les octets du code machine au moment de son exécution ?

le registre instruction (R.I.)

Représentation des données

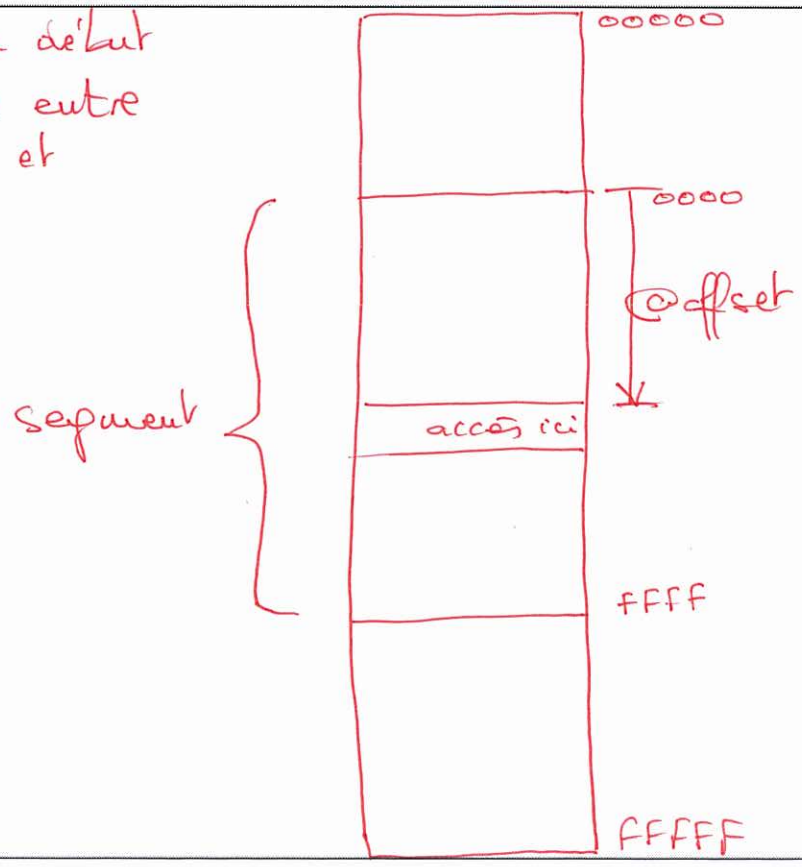
5. En little-endian, qu'est-ce qui change selon qu'un debugger représente le contenu de la mémoire par mots de 8 bits ou par mots de 16 bits ? Illustrer avec un exemple.

Adresse Données
0000 : 00 01 02 03 04 05 06 07
(par mots de 8 bits)

0000 : 0100 0302 0504 0706
(par mots de 16 bits)

6. Dans une architecture segmentée comme celle du 8086, qu'est-ce qu'une adresse offset ? Fournir une définition et illustrer par une interprétation graphique.

@ offset = @ relative au début du segment = distance entre le début du segment et l'adresse d'accès



Accès aux données

7. Dans les notations assembleur relatives aux modes d'adressage, que spécifie ce qui apparaît entre les crochets ?

une adresse (soit directement, soit par une combinaison de registres)

On considère les déclarations globales suivantes en langage C :

```
#define NULL 0
typedef struct s1 *ptr;
struct s1 {
        ptr ch1;
        int *ch2;
  int ch3;
};
struct s1 e;
int i, *p, tab[10];
```

8. Classer les différents identifiants (noms) en : constantes, types et variables. Pour les variables indiquer la taille mémoire qu'elles occupent globalement dans le segment de données.

constantes : NULL

types : ptr, struct s1

variables : e, i, p, tab

taille de l'ensemble des variables : 30 octets (6+2+2+20)

9. Coder en assembleur les instructions suivantes en respectant la méthodologie vue en cours :

```
p = &i;
```

```
MOV   [p], offset i
```

```
e.ch1 = (ptr) &p;
```

```
MOV   [e], offset p
```

```
e.ch2 = tab;
```

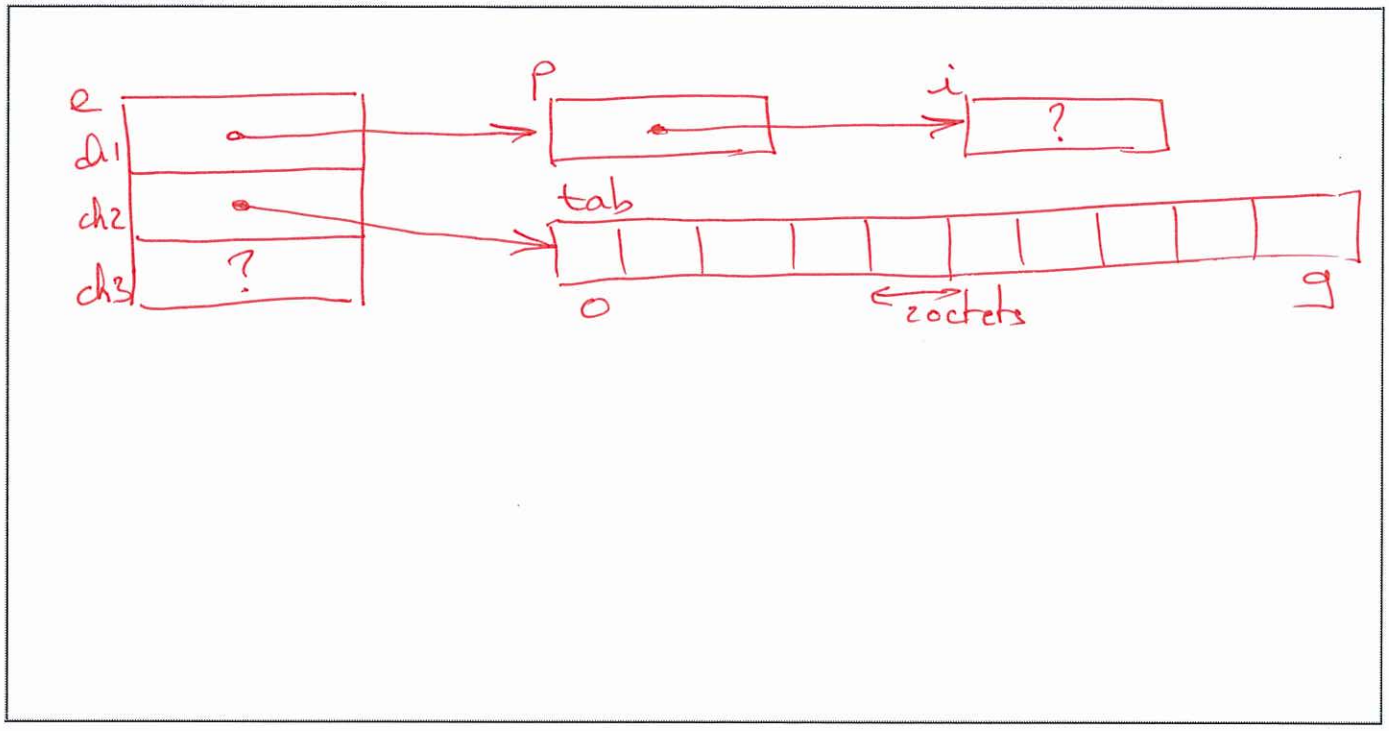
```
MOV   [e+2], offset tab
```

```
e.ch3 = i;
```

```
MOV   AX, [i]
MOV   [e+4], AX
```

10. A l'aide de schémas de mémoire, représenter ces variables et les relations entre elles après exécution des

instructions précédentes.



Structuration du code

On considère la portion de code en langage C suivante :

```
struct elt {
        char *nom;
        int age;
};
struct elt tab[10];
char *nom;
int i, age;
...
// on suppose que tab est initialisé puis :
nom=NULL;
i=0;
while (i<10 && nom==NULL) {
        if (tab[i].nom != NULL) {
                age = tab[i].age;
                nom = tab[i].nom;
        }
        i++;
}
```

11. Coder en assembleur la partie algorithmique de cette portion de code en respectant la méthodologie vue en cours :

```
nom=NULL;
```

```
MOV     [nom], NULL
```

```
i=0;
```

```
MOV     [i], 0
```

```
                MOV   BX, offset tab
while (i<10 && nom==NULL) {
```

```
whø:  CMP   [i],10
      JGE   endwhø
      CMP   [nom], NULL
      JNE   endwhø
```

```
        if (tab[i].nom != NULL) {
```

```
      MOV   SI,[i]
      ADD   SI,SI
      ADD   SI,SI
      CMP   [BX+SI],NULL
      JE    fsiø
```

```
            age = tab[i].age;
```

```
      MOV   AX,[BX+SI+2]
      MOV   [age], AX
```

```
            nom = tab[i].nom;
```

```
      MOV   AX,[BX+SI]
      MOV   [nom], AX
```

```
        }
```

```
fsiø:
```

```
        i++;
```

```
      INC   [i]
```

```
}
```

```
      JMP   whø
endwhø:
```

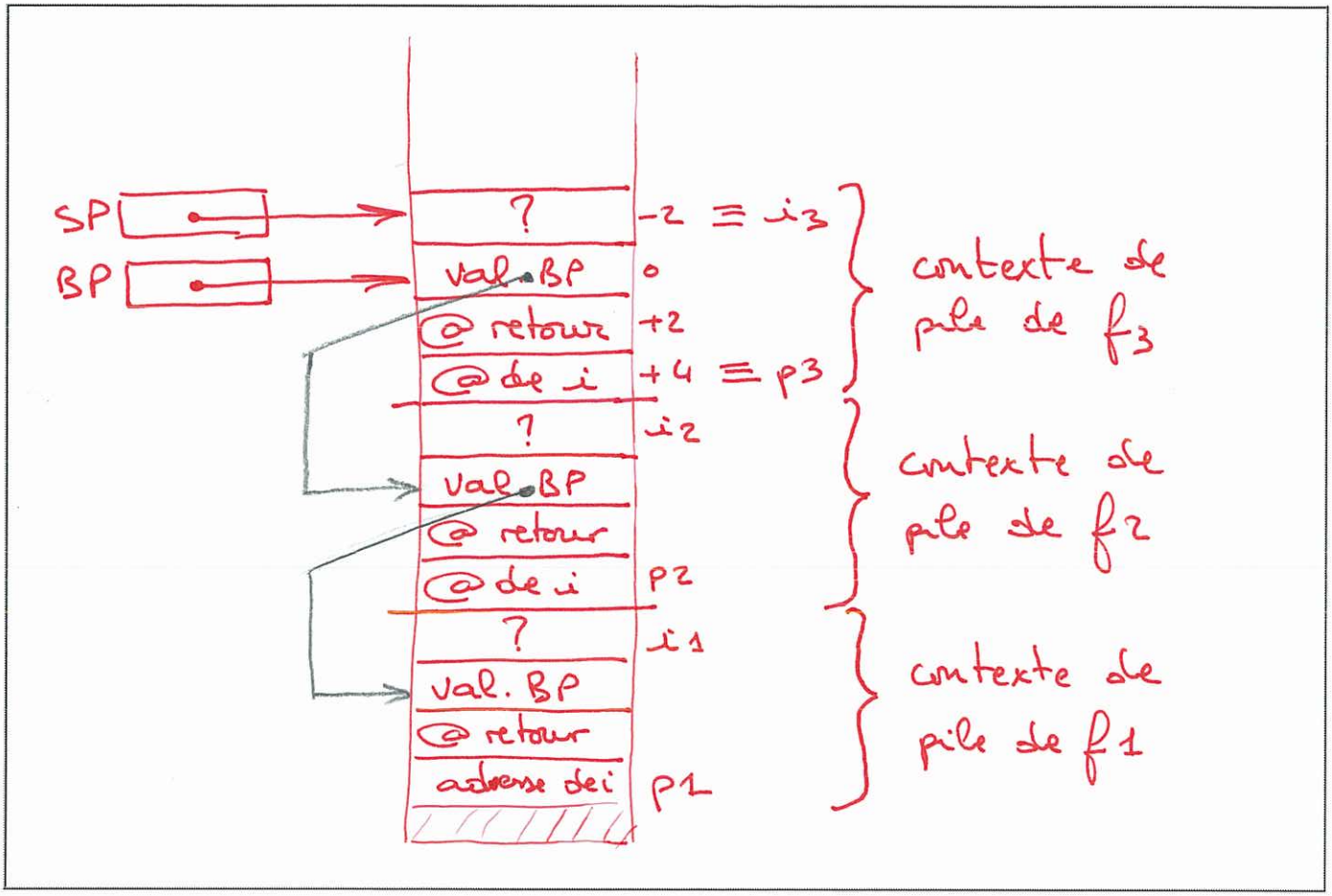
Sous-programmes

On considère la portion de code suivante :

```
int i;
void f3(int *p3) {
        int i3;
        // point B
        ...
}
void f2(int *p2) {
        int i2;
        f3(p2);
}
```

```
void f1(int *p1) {
        int i1;
        f2(p1);
}
int main() {
        // point A
        f1(&i);
}
```

12. Dessiner le schéma de pile résultant de l'exécution entre le point A et le point B. Ne représenter que le résultat final. Mettre en évidence les contextes des procédures f1, f2 et f3. Représenter SP et BP, les déplacements relatifs à BP pour la procédure f3 seulement. Repérer le nom de tous les paramètres et variables mis en jeu.



13. A l'aide de ce schéma, calculer l'espace occupé sur la pile.

24 octets

Coder en assembleur, l'appel initial :

```
f1( &i );
```

```
MOV   AX, offset i
PUSH  AX
CALL  -f1
POP   CX
```