NOM:

PRENOM: CORRIGÉ

PROMO:

Qroc A23

Epreuve du 25 octobre 2010
1 heure - sans documents (voir annexe)
Telecom Lille 1 - C. Tombelle

On considère la plate-forme 16 bits Turbo-C 8086. Dans l'ensemble de l'épreuve, sauf indication contraire, les adresses et pointeurs sont supposés intra-segment (« near »).

1. Quel est le registre du microprocesseur accueillant le code machine ?

le registre instruction

2. Qu'est-ce qu'un cycle fetch ?

un cycle de lecture du code machine

3. Quelle est la différence entre un code machine et un code assembleur ?

code machine est numérique (destiné à la machine)
code assembleur est symbolique (" à l'être humain)

4. Quels sont les signaux de contrôle par lequel le processeur indique le sens de fonctionnement du bus de données ?

$\overline{RD}$ et $\overline{WR}$

5. Quel est le signal de contrôle par lequel le processeur indique qu'il s'adresse à une mémoire ou un dispositif d'entrées-sorties ?

$M/\overline{IO}$

6. Entre quels instants est compté le temps d'accès d'une mémoire ?

entre le temps d'apparition de l'adresse et celui d'apparition de la donnée

7. Quelle partie du processeur est chargé de reconnaître l'instruction et d'organiser son exécution ?

le séquenceur

8. Quelle est la fonction arithmétique mise en œuvre lors du calcul de l'adresse offset dans le mode d'adressage basé, indexé avec déplacement.

l'addition

9. Calculer l'adresse physique correspondant à l'adresse segmentée 4A2B:0200

4A2B0 + 0200 = 4A4B0

10. Citez 2 modes de gestion des entrées-sorties.

les interruptions et le DMA

Modes d'adressage

On suppose que la variable `i` a pour adresse `2000` et contient la donnée `1000`. Donnez le contenu

des registres après exécution des instructions suivantes :

`mov ax,offset i`	ax = 2000
`mov ax,i`	ax = 1000
`mov ax,[i]`	ax = 1000
`mov bx,[2000]`	bx = 1000
`mov bx,2000`	bx = 2000
`mov ax,bx`	ax = 2000
`mov ax,[bx]`	ax = 1000
`lea bx,[bx]`	bx = 2000

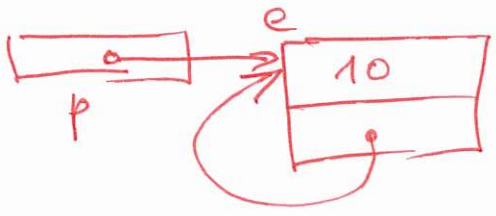
Accès aux données

On considère les déclarations suivantes :

```
typedef struct maillon *PMaillon;
struct maillon {
    int data;
    PMaillon suivant;
};
struct maillon e;
struct maillon *p;
```

11. Coder en assembleur et dessinez en partie droite l'état des variables après exécution de ces 3 instructions.

```
//  p = &e;
  MOV  [p], offset e
//  e.data = 10;
  MOV  [e], 10
//  p->suivant = p;
  MOV  BX, [p]
  MOV  [BX+2], BX
```



On ajoute les déclarations suivantes aux déclarations précédentes :

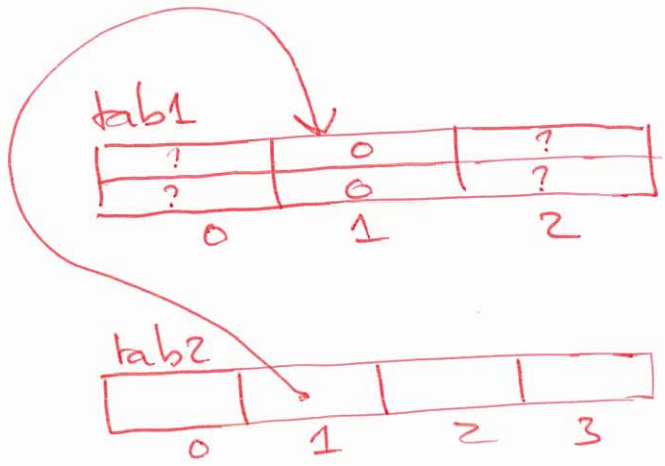
```
int i = ...; // valeur inconnue lors de la compilation
struct maillon tab1[3];
struct maillon *tab2[4];
```

12. Codez en assembleur. Représentez en partie droite l'état des variables après exécution de ces 3 instructions avec i valant 1.

```
//  tab1[i].data = 0;
    MOV  SI,[i]
    ADD  SI,SI
    ADD  SI,SI
    MOV  [SI+tab1],0

//  tab1[i].suivant = 0;
    MOV  [SI+tab1+2],0

//  tab2[i] = &tab1[i];
    ADD  SI,offset tab1
    MOV  DI,[i]
    ADD  DI,DI
    MOV  [DI+tab2],SI
```



Structuration du code

Onconsidère les déclarations suivantes :

```
int i, j;
int tab[10];
```

13. On suppose que `tab` est initialisé. Codez en assembleur l'algorithme suivant :

```
i = 0;
while (i < 9) {
    j = i;
    while (j < 10) {
        if (tab[i] < tab[j]) {
            ...
        }
        j++;
    }
    i++;
}
```

Dans la case ci-dessous, codez en assembleur l'algorithme sous chaque ligne de commentaire comme le ferait un compilateur C. Procédez avec méthode : positionnez d'abord les étiquettes et les instructions qui y renvoient selon les schémas type de génération de code utilisés par les compilateurs pour les structures de contrôle ; codez ensuite les autres instructions.

```
//  i=0;
    MOV  [i], 0
// while (i<9) {
wh0: CMP [i], 9
     JGE fwh0
//      j=i;
     MOV  SI, [i]
     MOV  [j], SI
//      while (j<10) {
wh1: CMP  [j], 10
     JGE  fwh1

//        if (tab[i] < tab[j]) {
     MOV  SI, [i]
     ADD  SI, SI
     MOV  DI, [j]
     ADD  DI, DI
     MOV  AX, [SI+tab]
     CMP  AX, [DI+tab]
     JGE  fsi0


          ...
//          }
fsi0
//          j++;
     INC  [j]
//      }
     JMP  wh1
fwh1:
//      i++;
     INC  [i]
//  }
     JMP  wh0
fwh0:
```

Sous-programmes

14. Dans un langage évolué, quelle est la différence entre le concept de procédure et celui de fonction.

une procédure modifie l'état du système
une fonction calcule et retourne un résultat

15. Que fait le code assembleur généré par un compilateur C pour fournir le résultat d'une fonction ? (ex : `return resultat;`)

Le code assembleur ... met le résultat dans un registre (Ax) ou un couple de registre (DX : Ax)

16. Dans quels segments respectifs sont stockés les variables globales, les paramètres, les variables locales.

variables globales : segment de ... données

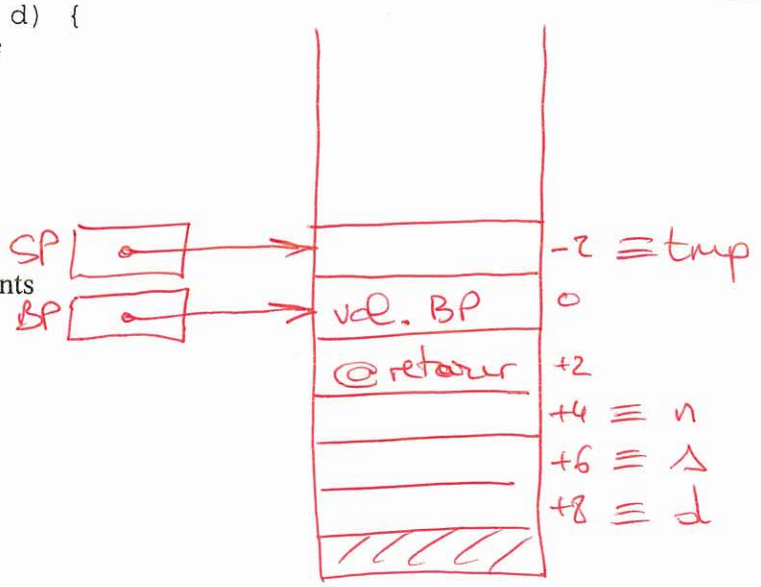
paramètres : segment de ... pile

variables locales : segment de ... pile

17. Dessinez le contexte de pile (frame) correspondant à la fonction suivante :

```
int h(int n, int s, int d) {
    int tmp = 3 - s - d;
    ...
    return ...;
}
```

Dessinez aussi les registres BP et SP, les noms des variables et paramètres avec leurs déplacements respectifs par rapport à BP.



18. Codez en assembleur l'appel de fonction suivant :

```
// int r; // r : variable globale
// r = h(3, 0, 1);
```

```
MOV  AX, 1
PUSH AX
MOV  AX, 0
PUSH AX
MOV  AX, 3
PUSH AX
CALL _h
ADD  SP, 6
MOV  r, AX
```