NOM:

PRENOM:

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 ?

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

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

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

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 ?

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

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

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.

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

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

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

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;


//  e.data = 10;


//  p->suivant = p;
```

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;




//  tab1[i].suivant = 0;



//  tab2[i] = &tab1[i];
```

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;

//  while (i<9) {


//       j=i;


//       while (j<10) {



//         if (tab[i] < tab[j]) {









            ...
//            }

//            j++;

//       }


//       i++;

//  }
```

Sous-programmes

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

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

Le code assembleur ...

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

variables globales : segment de ...

paramètres : segment de ...

variables locales : segment de ...

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);
```