Nom :
Prénom :
Promo :

Architecture des ordinateurs

IMT Lille Douai - épreuve du 14 février 2019 (Christophe TOMBELLE)
2 heures - calculatrice non programmable autorisée - sans documents (voir annexe).
Dans toute l'épreuve, on considère un processeur compatible Intel 8086 et le compilateur Turbo-C utilisant la syntaxe Intel pour l'assembleur. Toutes les adresses, registres d'adresses et pointeurs sont intra-segment (NEAR).

Matériel

Questions générales

1. Quel est le sens de circulation de l'information sur le bus de données lors d'un cycle de lecture ?

Mémoire vers processeur.

2. Quelles sont les deux phases de prise en charge d'une instruction machine par un microprocesseur ?

1ère phase : lecture code machine d'une instruction

2ème phase : exécution de l'instruction

3. Quels sont les 3 types d'opérandes possibles pour l'opérande de droite (source) d'une instruction `MOV` ?

.immédiat

.mémoire

.registre

4. Que signifient les crochets dans la syntaxe d'un opérande 8086 ?

qu'il s'agit d'un opérande mémoire

Questions appliquées

Mise en situation pour les questions 5 à 14. La valeur initiale des registres est :

```
AX=0403 BX=0202 CX=0201 DX=0100 SP=FFF0 BP=2000 SI=1000 DI=4000
DS=4000 ES=1000 SS=3000 CS=2000 IP=0100
```

Le contenu actuel du segment de données est :

```
DS:1200 03 05 07 09 01 02 04 06 ......
```

Le processeur est sur le point de prendre en charge l'instruction suivante :

```
CS:0100>8B4004          MOV AX,[BX+SI+4]
```

5. Quelle est l'adresse offset des octets suivants ?

adresse du `8B` : 0100

adresse du `40` : 0101

adresse du `04` : 0102

6. Quelle est l'adresse physique de début du segment de code ?

20000

7. Quelle est l'adresse physique (du 1er octet) de cette instruction ?

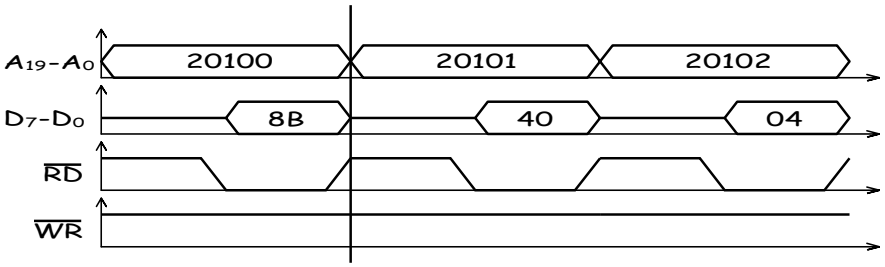
20100

8. Dans le chronogramme de la question suivante, quelle est la taille des bus ?

Bus d'adresses : 20 bits

Bus de données : 8 bits

9. Dessiner le chronogramme montrant uniquement l'acquisition du code machine de cette instruction. Annoter les bus avec les valeurs qu'ils véhiculent.



10. Quelle est l'adresse offset de l'octet de poids faible l'opérande (source) ?

1206 = BX + SI + 4 avec (BX=0202, SI=1000)

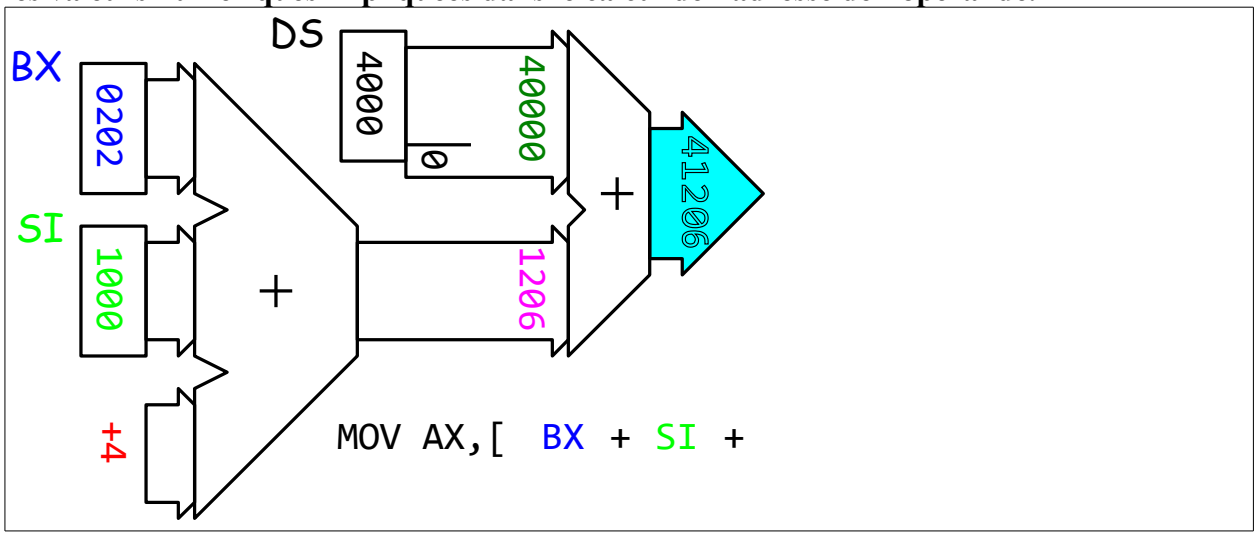
11. Quelle est l'adresse physique de début du segment de données ?

40000 car DS=4000

12. Quelle est l'adresse physique de l'opérande ?

41206 = 40000 + 1206

13. Dessiner un schéma de l'unité d'adressage complète interne au 8086 en l'annotant avec les valeurs numériques impliquées dans le calcul de l'adresse de l'opérande.



14. Quelle est la valeur de AX après exécution de cette instruction ?

Le contenu actuel du segment de données est :
`DS:1200 03 05 07 09 01 02 04 06 ......` réponse : `0604`

Modes d'adressage

On considère la situation initiale suivante :
Registres :

```
AX=0600 BX=0004 CX=0400 DX=0100 SP=FFEE BP=FFF0 SI=0302 DI=0002
DS=0CDA ES=1CDA SS=2CDA CS=3CDA IP=006B
```

Déclarations en C :

```
#define VALUE 0x304
int i = 0x302;   /* adresse de i : 0x300 */
int j = VALUE;   /* adresse de j : 0x302 */
int k = 0x306;   /* adresse de k : 0x304 */
int *m = &i;     /* adresse de m : 0x306 */
```

Les instructions ci-dessous sont indépendantes les unes des autres et s'exécutent donc en repartant à chaque fois de cette situation initiale. Dessiner au brouillon `i`, `j`, `k`, `m` et `BX` puis :

15. Compléter ci-dessous le tableau en précisant la valeur hexadécimale obtenue dans le registre `BX` après exécution de chaque instruction. Préciser le(s) mode(s) d'adressage mis en œuvre (répondre « aucun » si l'instruction n'en utilise pas).

Instruction	Valeur de `BX`	Mode(s) d'adressage
`MOV BX,offset j`	`0302` MOV BX,offset j <=> MOV BX,0x302	`immédiat`
`MOV BX,VALUE`	`0304` MOV BX,VALUE <=> MOV BX,0x304	`immédiat`
`MOV BX,[VALUE]`	`0306` valeur de k (MOV BX, [VALUE] <=> MOV BX,[304])	`direct`
`LEA BX,[VALUE]`	`0304` adresse entre crochets (LEA BX,[VALUE] <=> LEA BX,[304])	`direct`
`MOV BX,k`	`0306` valeur de k (avec ou sans crochets)	`direct`
`MOV BX,[m]`	`0300` valeur de m = adresse de i	`direct`
`MOV BX,[BX+DI+i]`	`0300` [BX+DI+i] [0004+0002+300] valeur de m = adresse de i	`indirect`
`LEA BX,[BX+SI-4]`	`0302` [BX+SI-4]=[0004+0302-4] LEA BX,[302]	`indirect`

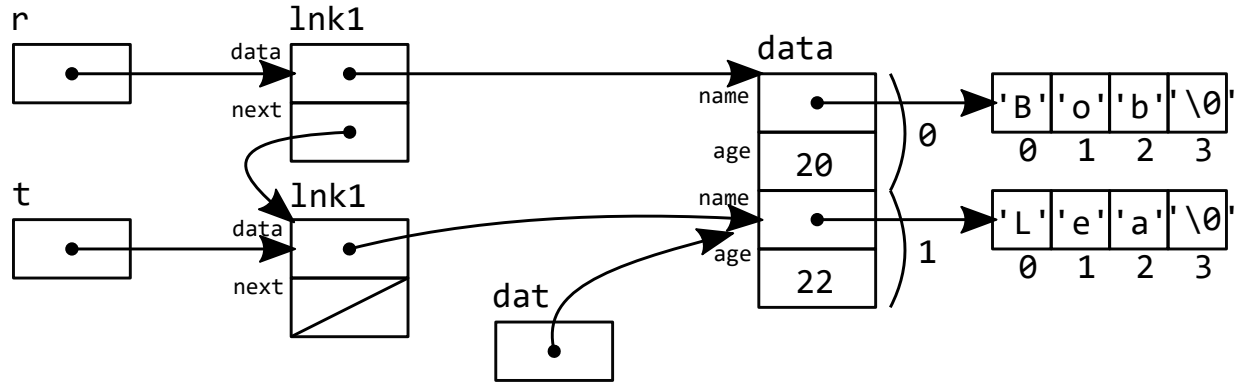
Structures de données

On considère les déclarations suivantes en langage C.

```
typedef char *String;
typedef struct Data *IData;
struct Data {
       String name;
       int age;
} data[2];
IData dat;
typedef struct Link *ILink;
struct Link {
       IData data;
       ILink next;
} lnk1, lnk2;
ILink r, t;
```

```
/*  initialisations */
data[0].name = "Bob";
(* data).age = 20;
dat = &data[1];
dat->name = "Lea";
dat->age = 22;
r = &lnk1;
t = &lnk2;
r->data = data;
r->next = t;
t->data = dat;
t->next = NULL;
```

16. À l'aide de schémas de mémoire, représenter les variables après exécution des instructions (ne pas montrer les étapes). Respecter la taille relative des cadres représentant les variables (1 octet = 1 carré). Bien préciser les nom, forme, flèche, valeur et indice des variables. Indiquer clairement où les pointeurs pointent.



17. Coder en assembleur les instructions suivantes.

```
dat = &data[1];
```

```
MOV [dat],offset data + 4
```

```
dat->age = 22;
```

```
MOV BX,[dat]
MOV [BX+2],22
```

```
r = &lnk1;
```

```
MOV [r],offset lnk1
```

```
r->data = data;
```

```
MOV BX,[r]
MOV [BX],offset data
```

```
r->next = t;
```

```
MOV AX,[t]
MOV [BX+2],AX
```

Structures de contrôle

On considère les déclarations globales suivantes :

```
#define CNT_ROWS 4
#define CNT_COLS 3

char *stCols, *stRows, st;
int col, row ;
```

On considère l'algorithme suivant :

```
col = 0;
while (col < CNT_COLS) {
       stCols[col] = LOW;
       row = 0;
       while (row < CNT_ROWS) {
              st = stRows[row];
              ...
              row++;
       }
       stCols[col] = HIGH;
       col++;
}
```

+-On suppose que les initialisations nécessaires ont eut lieu.

18. Coder l'algorithme ci-dessous en assembleur (chaque ligne dans le cadre qui la suit).

```
col = 0;
```
```
        MOV [col],0
```
```
while (col < CNT_COLS) {
```
```
whil0 : CMP [col],CNT_COLS
        JGE endwh0
```
```
      stCols[col] = LOW; // attention, byte ptr qq part
```
```
        MOV BX,[stCols]
        MOV SI,[col]
        MOV byte ptr[BX+SI],LOW
```
```
      row = 0;
```
```
        MOV [row],0
```
```
      while (row < CNT_ROWS) {
```
```
whil1 : CMP [row],CNT_ROWS
        JGE endwh1
```
```
            st = stRows[row];
```
```
        MOV BX,[stRows]
        MOV SI,[row]
        MOV AL,[BX+SI]
        MOV [st],AL
```
```
            ...
            row++;
```
```
        INC [row]
```
```
      }
```
```
        JMP whil1
endwh1:
```
```
      ...
      col++;
```
```
        INC [col]
```
```
}
```
```
        JMP whil0
endwh0:
```

Passage des arguments

L'algorithme précédent est maintenant écrit sous forme de fonction :

```
char kbdScan(char *stCols, char *stRows) {
      int col, row;
      char scan, st;
      // point B
      ...
      return scan;
}
```

On s'intéresse à l'appel de **kbdScan** depuis la fonction **main**.

On a les variables globales suivantes :

```
char rows[CNT_ROWS] = { 7, 5, 3, 2 };
char cols[CNT_COLS] = { 4, 9, 8 };
char key;

int main(void) {
      // point A
      key = kbdScan(cols, rows);
      return 0;
}
```

19. Coder en assembleur la ligne suivante :

`key = kbdScan(cols, rows);`

```
        MOV  AX,offset rows
        PUSH AX
        MOV  AX,offset cols
        PUSH AX
        CALL _kbdScan
        ADD  SP,4
        MOV  [key],AL
```

20. Dessiner le résultat de l'évolution de la pile lors de l'exécution du point A au point B lors de l'appel `key = kbdScan(cols, rows)`**. Faire apparaître :**

- le contexte de pile (frame) de cette fonction,
- les registres associés,
- les variables globales `cols et rows`**,**
- les relations (<u>flèches</u>) éventuelles avec ces variables globales,
- les noms,
- les déplacements relatifs à BP,
- les valeurs quand l'exécution a atteint le `point B`**.**