**Corrections Livret élève**

**1 Exercices séquence 1 - corrigé**

**Exercice 1.1**

1. La logique de décodage
2. Le bus de données est bidirectionnel 3 états
3. lors d'une lecture : la mémoire, lors d'une écriture : le processeur, lors d'une entrée : un dispositif d'entrée, lors d'une sortie : le processeur
4. l'architecture en bus limite le nombre de connexions entre les composants ; dans une architecture complètement maillée un ajout de composant exigerait une connexion à chaque composant existant ce qui est irréaliste
5. L'instruction d'entrée est IN, celle de sortie est OUT
6. C'est le signal M/IO qui distingue les accès IO (M/IO = 0) des accès mémoire (M/IO = 1)
7. À une adresse mémoire n donnée, on relit ce qu'on a écrit précédemment ; ce qu'on lit à une adresse IO n donnée (e.g. entrée d'une donnée reçue via un réseau) n'a rien à voir (a priori) avec ce qu'on écrit à cette même adresse n (e.g. sortie d'une donnée à émettre sur le réseau)
8. Calcule de la taille de l'espace adressable : un processeur comportant n signaux d'adresses est capable d'adresser 2n emplacements différents. En général, une adresse repère 1 octet (même si le bus de données comporte plus de 8 signaux). Le processeur est donc capable d'adresser 2n octets (taille de l'espace adressable = 2n octets).
9. La capacité de stockage d'une  mémoire comportant 10 signaux d'adresses et 8 signaux de données est de 210 octets = 1 ko = 1024 octets.

**2 Exercices séquence 2 - corrigé**

**Exercices de la séquence en ligne**

Voir corrigé du TD 1

**Exercice 2.1**

1. Le processeur lit la mémoire pour **a)** acquérir un(e partie de) code machine (cycle fetch) **b)** parce qu'une instruction le lui demande (lecture programmée).
2. Les 2 types d'informations circulant sur le bus de données sont les codes machine et les données manipulées par le programme
3. ordres de grandeur liés aux entrées-sorties :
   1. 9600 bps ≈ 10000 bps, or 1 caractère ≈ 10 bits (en ajoutant 1 bit start et 1 bit stop) soit 1000 caractères/s soit 1 caractère/ms soit 20 caractères en 20 ms
   2. 20ms = 1/50e de seconde si un processeur exécuter 50 million d'instructions par seconde, il exécute 1 million d'instructions en 1/50e de seconde
   3. en conclusion pendant l'émission de 20 caractères à 9600 bps un tel processeur est capable d'exécuter environ 1 million d'instructions
4. la portion de puissance du processeur utilisée pour le compte du système d'exploitation s'appelle l'overhead

**3 Exercices séquence 3 - corrigé**

**Exercice 3.1**

1. Chaque adresse repère 1 octet donc la distance entre 2 mots (mot = mot de 16 bits dans la terminologie Intel du 8086) est donc de 2
2. L'ordre de stockage utilisé par les processeurs Intel est l'ordre dit "little endian" où l'octet de poids faible est stocké avant (adresse n) l'octet de poids fort (adresse n+1)
3. Les registres n'ont pas d'adresses car ils sont situés dans le processeur ; seuls les emplacements mémoire et IO ont une adresse
4. Les registres du 8086 ont une taille de 16 bits (2 octets)
5. Les registre AH et AL ont une taille de 8 bits : ce sont les parties de poids fort (AH) et faible (AL) du registre AX.
6. Un indicateur ou flag (drapeau) ont une taille de 1 bit.
7. Les registres sécables du 8086 sont AX, BX, CX et DX
8. Le processeur 8086 comporte 20 signaux d'adresses (A0 à A19) ; son espace adressable a une taille de 220 octets (1 Mo)
9. Une adresse offset est relative à l'adresse de début de segment. Elle a donc un sens à l'intérieur d'un segment.
10. Les adresses physiques du 8086 ont une taille de 20 bits ; les adresses offsets : 16 bits ; les adresses segments : 16 bits ; les adresses segmentées (segment : offset) : 32 bits.
11. Si l'adresse segmentée est 4A2B:027C alors l'adresse segment est 4A2B et l'adresse offset est 027C. On calcule l'adresse physique comme ceci : 4A2B0 + 027C = 4A52C.
12. Dans la terminologie Intel, un mot est un mot de 2 octets, un double mot est un mot de 4 octets, un paragraphe est un bloc de 16 octets.
13. Une adresse offset est codée sur 16 bits. La taille d'un segment est donc 216 octets = 64 ko.
14. Les adresses de début de segment se terminent (en hexadécimal) par un 0 (e.g. 4A2B0) elles sont donc multiples de 10 hexa = 16 décimal.
15. Sous Turbo Debugger, la notation CS:0100 est équivalente à 427C:0100 si le registre CS contient 427C.
16. Associations entre registres segment et offset :
    1. BX, SI et DI sont des registres offsets pour accéder au segment de données
    2. BP et SP sont des registres offsets pour accéder au segment de pile
    3. IP est le registre offset pour accéder au segment de code
17. voir adressage CHS des disques dur sur wikipedia

**4 Exercices séquence 4 - corrigé**

**TP2 4 premières lignes**

**// AX = 1; (adressage immédiat)**

mov ax,1

**// AX = constante; (adressage immédiat)**

mov ax,constante

**// AX = variable1; (adressage direct)**

mov ax,[variable1]

**// AX = variable2; (adressage basé sur BX)**

mov bx,offset variable2 // bx = &variable2;

mov ax,[bx] // adressage basé sur BX

**Exercice 4.1 Modes d'adressage**

1. En adressage direct, l'adresse apparaît dans le code machine
2. L'adressage immédiat ne peut pas être utilisé comme opérande gauche
3. En adressage indirect, le code machine code le nom des registres dont la combinaison fournit l'adresse de l'opérande
4. En adressage immédiat, l'opérande apparaît dans le code machine
5. L'adressage immédiat ne fait appel à aucune adresse
6. Le mode d'adressage [BX+SI+dépl] accède au segment de données
7. Il s'agit d'une forme d'adressage indirect : l'adressage basé, indexé avec déplacement
8. BX est un registre de base (dans BX, B comme Base)
9. SI est un registre d'indes (dans SI, I comme Index)
10. [BX] est un adressage basé
11. [dépl] est un adressage direct
12. [SI+dépl] est un adressage indexé avec déplacement
13. [BP+SI+dépl] accède au segment de pile (BP comme Base de Pile)

**Exercice 4.2 Instructions**

1. LDS BX,[unPointeurFar] charge le couple de registres DS:BX avec l'adresse fournie par le pointeur far ; LES BX,[unPointeurFar] charge ES:BX
2. LEA BX,[BX+SI+dépl] charge dans BX l'adresse résultant de la somme BX+SI+dépl appelée adresse effective
3. L'adresse effective est une adresse offset
4. Dans l'instruction ADD operandeGauche,operandeDroit le résultat est écrit dans operandeGauche
5. Si AL contient 5A, après exécution de AND AL,0F AL contient 0A. Après exécution de 4 SHR AL,1 AL contient 05

**Exercice 4.2 Nombres signés**

1. Dans un nombre signé, c'est le bit de poids le plus fort qui a, par convention, un poids négatif
2. Le nombre -1 codé sur un octet s'écrit 11111111, sur 2 octets, il s'écrit 1111111111111111
3. Le nombre 32767 (215-1) s'écrit 0111111111111111 si on lui ajoute 1 on trouve 1000000000000000 (32768 si le nombre est non signé, mais aberrant si le nombre est signé : 32767 + 1 = -32768 !!!)

### 5 Exercices séquence 5 - corrigé

### TP2 suite

**// p1 = &variable3; (via SI)**

mov si,offset variable3 // ou bien : lea si,[variable3] // si = &variable3

mov [p1],si // p1 = si

**// AX = \*p1; (via BX)**

mov bx,[p1] // bx = p1

mov ax,[bx] // ax = objet pointé par bx

**// AH = enreg.champ2; (adressage basé sur BX avec déplacement)**

mov bx,offset enreg // bx = &enreg;

mov ah,[bx+2] // champ2 : déplacement de 2 octets

**// AX = i; (adressage indexé par SI)**

mov si,offset i // si = &i

mov ax,[si] // adressage basé sur si

**// AL = enreg.champ3; (adressage indexé par SI avec déplacement)**

mov si,offset enreg // si = &enreg

mov al,[si+3] // champ3 : déplacement de 3 octets

**// AH = tab\_byte[ i ]; (adressage basé sur BX et indexé par SI)**

mov bx,offset tab\_byte // bx = &tab\_byte[0]

mov si,[i] // si = i

mov ah,[bx+si]

**// AL = tab\_enr[ 4 ].champ3; (adressage basé, indexé avec déplacement)**

mov bx,offset tab\_enr // bx = &tab\_enr[0]

mov si,16 // si = 4\*4

mov al,[bx+si+3]

**// AH = tab\_enr[ i ].champ3; (adressage basé, indexé avec déplacement)**

mov si,[i] // si = i

add si,si // si = 2\*si

add si,si // si = 2\*si

mov ah,[bx+si+3

**6 Exercices séquence 6 - corrigé**

**Exercice 6.1**

1. Une étiquette est une adresse exprimée de manière symbolique
2. Un processeur prend des décisions à l'aide d'instructions de sauts conditionnels
3. Les instructions JMP CALL et RET existent en version intra et inter-segment
4. Du code mort est du code qui ne sera jamais exécuté car il n'est pas accessible (code non repéré par une étiquette et précédé d'un saut inconditionnel)
5. Le code qui suit un saut inconditionnel doit donc être repéré par une étiquette
6. Une boucle se reconnaît à la présence d'un saut "en arrière" (étiquette située avant)

**Exercice 6.2**

**// -- affichage poids forts**

mov dl,[car]

shr dl,1 // cadrer le quartet de poids fort en poids faible

shr dl,1

shr dl,1

shr dl,1

**// si dl <= 9**

cmp dl,9

ja sin0

**// ajouter code ascii de '0'**

add dl,'0' // '0' = 0x30 = 48

// sinon -- si dl > 9

jmp fsi0

sin0:

**// retirer 10 et ajouter le code ascii de 'A'**

add dl,'A' - 10 // 'A' = 0x41 = 65

// 65-10=55 est calculé lors de la compilation

**// fsi**

fsi0:

**// afficher**

mov ah,2

int 21h

**// -- afficher poids faibles**

mov dl,[car]

and dl,0Fh // isoler le quartet de poids faible

**// si dl <= 9**

cmp dl,9

ja sin1

**// ajouter code ascii de '0'**

add dl,'0'

**// sinon -- si dl > 9**

jmp fsi1

sin1:

**// retirer 10 et ajouter le code ascii de 'A'**

add dl,'A' - 10

**// fsi**

fsi1:

**// afficher**

mov ah,2

int 21h

**Exercice 6.3**

**/\* i = 0; \*/**  
 mov [i],0  
**/\* while (chaine[i] != '\0') { \*/**  
 mov bx,offset chaine  
whil0: mov si,[i]  
 cmp byte ptr[bx+si],'\0'  
 je endwh0  
**/\* if (chaine[i] == ',') \*/**

cmp byte ptr[bx+si],','  
 jne endif0  
**/\* chaine[i] = ';' \*/**  
 mov byte ptr[bx+si],';'  
endif0:  
**/\* i++; \*/**  
 inc [i];  
**/\* } \*/**  
endwh0:

NB :

* byte ptr indique que les opérandes ont une taille de 8 bits
* byte ptr est nécessaire quand aucun opérande ne précise sa taille

**Exercice 6.4 Réflexion**

Dans un switch (expression) où expression ne peut prendre que des valeurs consécutives comme 0, 1, 2, 3... on peut faire pointer bx sur un tableau d'adresses de sauts. Si le tableau s'appelle tabJmp et que bx contient la valeur de l'expression, on écrira :

add bx,bx  
 jmp [bx+tabJmp]

Si expression vaut 0, le saut aura lieu vers l'adresse de saut contenue dans tabJmp[0],  
si expression vaut 1, le saut aura lieu vers l'adresse de saut contenue dans tabJmp[1], etc...

NB :

* Il faut préalablement avoir initialisé tabJmp avec les adresses de saut, i.e. adresse du code correspondant à case 0, case 1, ...
* Cette méthode ne convient pas si les valeurs possibles de l'expression sont séparées par des "trous" de taille irrégulière comme 0, 2130, 3084, 3085 etc...

**7 Exercices séquence 9 - corrigé**

Un pilote de périphérique (device driver) est une partie modulaire du système d'exploitation destinée à gérer une dispositif d'entrées-sorties de l'ordinateur. Il fournit un accès logiciel au périphérique via des primitives standards open, read, write, close et ioctl. Si le matériel génère une demande d'interruption, le pilote intègre le sous-programme de traitement de cette demande.

**Exercice 9.1 Polling**

1. Pour savoir si un événement s'est produit, il faut lire un registre d'état du périphérique qui est à son origine.
2. Le polling est acceptable si la durée entre 2 consultations permettant de ne pas "manquer" un événement fait que le processeur n'est pas trop mobilisé (cas d'un ordinateur) ou que cette mobilisation soit acceptable car le processeur n'a rien d'autre à faire (cas d'un système dédié à un seul programme).

**Exercice 9.2 Interruptions**

1. Un vecteur contient l'adresse "far" d'un sous-programme de traitement d'une demande d'interruption. La table des vecteurs d'interruptions (TVI) regroupe ces adressses en un tableau. Les numéros de vecteurs sont les indices de ce tableau.
2. La taille d'un vecteur (pointeur "far") est de 4 octets. Le vecteur de numéro 0x21 est situé à l'adresse 0x00084 puisque la TVI est située à l'adresse 0x00000.
3. Un numéro de vecteur est codé sur 1 octet. La TVI a 256 éléments.
4. La TVI a une taille de 1024 octets (256 pointeurs "far").
5. Dans les interruptions d'origine matérielle, le numéro de vecteur est fourni par un contrôleur d'interruptions programmable (PIC).
6. Dans les interruptions d'origine logicielle, le numéro de vecteur est fourni par le code machine de l'interruption INT <nvect> ou <nvect> désigne le numéro de vecteur.
7. Les interruptions logicielles servent à accéder aux services du système d'exploitation.
8. Le numéro de vecteur des exceptions est implicite et dépend de la nature de l'exception (division par 0, code machine invalide...).
9. Un sous-programme d'interruption doit sauvegarder (PUSH) les registres qu'il va modifier et les restituer (POP) ensuite (avant IRET).

**Exercice 9.3 DMA**

1. Une interface réseau, un contrôleur de disque dur sont des dispositifs d'entrées-sorties intensives faisant appel au DMA.
2. C'est le processeur qui programme le contrôleur de DMA (DMAC) puis lance la sous-traitance DMA.
3. Le DMAC exécute de manière matérielle l'équivalent d'un algorithme de transfert par exemple (dispositif d'entrée vers mémoire)
4. a) tanque donnée non disponible dans dispositif d'entrée  
    ;  
   b) lire donnée du dispositif d'entrée;  
   d) écrire en mémoire la onnée lue;  
   e) recommencer en a) jusqu'à la fin du transfert;
5. C'est le DMAC qui synchronise le transfert d'un mot à l'aide des signaux suivants :
   * DRQx le dispositif IO demande au DMAC à transférer un mot
   * HOLD le DMAC demande au µP de libérer les bus
   * HLDA le µP confirme au DMAC la libération des bus
   * DACKx lz DMAC confirme au dispositif IO le transfert d'un mot
   * IOR lecture dispositif d'entrée + MWR écriture mémoire => transfert dispositif d'entrée vers mémoire
   * MRD lecture mémoire + IOW écriture dispositif de sortie => transfert mémoire vers dispositif de sortie
6. Les mots transférés passent par le bus de données
7. Une demande d'interruption signale la fin de la sous-traitance DMA
8. Pendant le transfert d'un mot, le µP libére l'usage des bus