IUT Paris Descartes

Département STID INFO

Nom : CAPOSTOLET Prénom : Anaïs

Groupe : 412 Date : 20/03/18

Matière : ARCHITECTURE

Note : 20

Appréciation :

PARTIE I : COURS

A] La chaine de développement logiciel

Q1) Le langage de haut niveau n'est pas dépendant du microprocesseur. Ce dernier est multiplateforme car il n'est pas encore compilé. En revanche le langage d'assemblage est propre à un microprocesseur car son étendue dépend du jeu d'instruction qui varie d'un CPU à l'autre. De même, le langage machine est propre à microprocesseur car il dépend des instructions. 0,75

1/12

(Q2) Un module objet est un programme en langage machine mais non exécutable, il est généré par l'assembleur. La différence avec un fichier exécutable est l'absence des bibliothèque statiques. 1,25

(Q3) Une bibliothèque statique est un module de code qui contient des fonctionnalités et qui est utilisé comme ressource par un programme. Elles sont copiées dans le programme par l'éditeur de lien. 0,5

(Q4) L'éditeur de liens statiques transforme un module objet non exécutable fichier exécutable .EXE en y copiant les bibliothèques statiques nécessaires. 0,5

2/2

3] Le Fichier listing

1) Les lignes du cadre 2 :

IDEAL → style de programmation
DOSSEG → Programme pour le DOS
MODEL TINY → Stipule qu'il s'agit d'un petit programme
P8086 → Stipule qu'on programme pour le CPU Intel 8086

0,5

(Q2) L'adresse de rangement de cette variable est 0000h. Elle est initialisée à la valeur 65535. Elle appartient au segment de donnée (DATASEG). Son encombrement en mémoire est de 2 octets car elle est déclarée en tant que DW, mot donc 16 bits car on est sur un CPU 16 bits. L'adresse de la donnée suivante est 0006

1,25

(Q3) L'adresse de rangement de cette instruction est 001B. Cette instruction appartient au segment de code (CODESEG). Son encombrement mémoire est 2 octets L'adresse de l'instruction suivante est 001D.

1

3/12

Q4) Les adresses redémarrent à 0 à chaque segment car ce sont des adresses logiques, qui sont donc par rapport au début du segment en mémoire. 0,5

C) La segmentation

Q1) Un segment est une portion contigüe de mémoire, de taille fixe, et dédiée à une usage spécifique. 0,5

Q2) Le CPU repère les segments en mémoire principale grâce à des registres qui contiennent les adresses de début des segments en mémoire. 0,5
Ce sont les registres de segmentation: ds, es, ss, cs

Q3) L'intérêt de la segmentation est l'utilisation du même code pour plusieurs utilisateurs différents. 0,25

Q4) Dans l'annexe 1, 2 segments sont déclarés : DATASEG, CODESEG, les segment contenant respectivement, les données (variables) et le code (instructions). 0,25

PARTIE II : ECRITURE D'UN PROGRAMME

Version 1

Q-1)

Variable	Valeur Min	Valeur Max	Format Min	Format Retenu
x	1	6	~~8~~	DB
y	0	6	~~8~~	DB
Resultat	1	46656	16	DW

1,75

5/12

Q2) Partie 1 : Variables puis_DST.asm

```
X   DB   3
Y   DB   2
R   DW   0
```

0,75

Q3) Partie 2 : Code puis_DST.asm

```
    mov al, [X]
    xor ah, ah
    mov cl, [Y]
    xor ch, ch

boucle:
    mul ax
    loop boucle
finb:
    mov [R], ax
```

1

2,5

6/12

Version 2 :

(Q-1) Nouvelle partie 2 :

```
    mov al, [X]
    xor ah, ah
    mov cl, [Y]
    xor ch, ch
    cmp CX, 0
    jne boucle
    mov [R], 1
    jmp fin

boucle:
    mul ax
    loop boucle

finb :
    mov [R], ax

fin :
```

1

7/12

Version 3

(Q4)

Partie 1 : Variables : puis_DST.asm

```
X       DB      3
Y       DB      2
R       DW      0
msgErr  DB "Valeur Max depassee",10,13,"$"
```

Partie 2 : Code : puis_DST.asm

```
mov al, [X]
xor ah, ah
mov cl, [Y]
xor ch, ch
cmp ax, 6
ja erreur
cmp cx, 6
ja erreur
cmp cx, 0
jne boucle
mov [R], 1
jmp fin
```

8/10

IUT Paris Descartes

Département STID INFO

Nom : CADOSTOCET Prénom : Arsène

Groupe : 1T2 Date : 20/03/18

Matière : ARCHITECTURE

Note :

Appréciation :

```
boucle :
        mul ax
        loop boucle

fin :
        mov [R], ax
        jmp fin

erreur :
        mov ah, 9
        mov dx, OFFSET msgErr
        int 21h

fin :
```

3

9/12

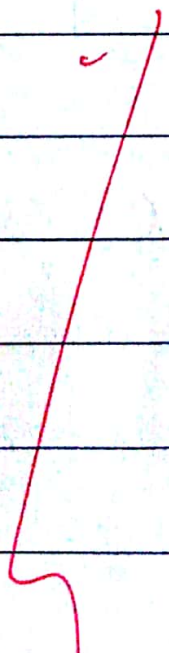
Version 4

(Q1)

Partie 3 : sous programme puis_sp

```
cmp cx, 0
jne boucle
mov ax, 1
ret

boucle:
mul ax
loop boucle
```



Partie 2 : code appelant

```
mov al, [X]
xor ah, ah
mov cl, [Y]
xor ch, ch
call puis_sp
mov [R], ax
```

2pts

Q2) Bonus

Partie 3 : Sous programme puis_sp

```
mov bp, sp
mov ax, [bp+2]
mov cx, [bp+4]
cmp cx, 0
jne boucle
mov bx, 1
ret

boucle:
mul ax
loop boucle
```

11/12

```
fib:
    mov bx, ax
    ret
```

Partie 2 : Code appelant

```
push [x]
push [x]
call puis_sp
pop bx
pop ax
mov [R], bx
```

3

DST UE21 - M2101
Architecture et programmation des mécanismes de base d'un système informatique

IUT de Paris Descartes, Département Informatique
Ph. Darche

Mars 2018
Durée : 2h

Aucun document autorisé. Calculatrice non autorisée. Sujet de trois pages, quatre annexes et deux transparents de cours. Les parties sont indépendantes. Répondez dans l'ordre de numérotation des problèmes. Barème indicatif. **Justifiez clairement vos réponses.** Aucun programme rendu sur feuille de brouillon.

Partie I : Questions de cours (9 points)

A. La chaîne de développement logiciel

Q1. Est-ce qu'un langage de haut niveau est dépendant du microprocesseur ? Mêmes questions pour les langages d'assemblage et machine. Justifiez vos réponses.

Q2. Donnez la définition d'un module objet (extension de fichier .obj ou .o selon le système d'exploitation). Quel est l'outil logiciel qui le génère ? Quelle(s) différence(s) y a-t-il entre un fichier objet et un fichier exécutable ?

Q3. Qu'est-ce qu'une bibliothèque (statique) ? Que contient-elle ?

Q4. Quel est l'intérêt d'un éditeur de liens (statiques) ?

B. Le fichier listing

Soit le fichier listing en annexe 1.

Q1. A quoi servent les lignes du cadre 2 ?

Q2. Soit le cadre 4. Quelle est l'adresse de rangement de la variable `C` à la ligne 15 ? Quelle est la valeur d'initialisation de cette variable ? A quel segment appartient cette variable ? Quel est son encombrement mémoire (en octet) ? Quelle est l'adresse de la donnée suivante ?

Q3. Soit le cadre 5. Quelle est l'adresse de rangement de l'instruction `mov ah, 4Ch` à la ligne 32 ? A quel segment appartient cette instruction ? Quel est son encombrement mémoire (en octet) ? Quelle est l'adresse de l'instruction suivante ?

Q4. Pourquoi les valeurs des adresses démarrent à zéro à chaque changement de segment ?

C. La segmentation

Q1. Donnez la définition d'un segment.

Q2. Comment le CPU repère-t-il les segments en mémoire principale ?

Q3. Quel est l'intérêt de la segmentation ?

Q4. Soit l'annexe 1. Combien y a-t-il de segments déclarés dans le programme ? Nommez-les.

Partie II : Ecriture d'un programme en langage d'assemblage (12,75 points + bonus)

On décide d'écrire un programme en langage d'assemblage 8086 qui calcule la fonction x^y (x et y, entiers naturels) et qui range le résultat dans une variable **résultat**.

La démarche de développement sera d'écrire une première version sans se préoccuper de la valeur particulière y = 0. La version 2 prendra en compte ce cas particulier. La version 3 consistera à afficher un message d'erreur pour les valeurs maximales de x et de y. La version 4 transforme le programme de la version 2 en sous-programme.

Vous éviterez au maximum d'utiliser dans le code les variables mémoires déclarées pour une question de rapidité d'exécution en privilégiant les registres du microprocesseur. Par contre, en fin de chaque programme, vous n'oublierez pas de ranger le résultat du calcul dans la variable concernée. La dernière annexe donne les instructions de sauts conditionnels qui peuvent vous être utile.

Version 1. Ecriture d'un programme principal sans traitement des cas particuliers

Ecriture d'un programme en langage d'assemblage 8086 qui calcule x^y ($x \in \mathbb{N}^*$ et $y \in \mathbb{N}$, x et y < 7) **de manière itérative** (*i.e.* pas de récursivité). **Un squelette de programme est fourni en annexe 2. Les parties 1 et 2 seront écrites sur votre copie.**

Q1. Trois variables, `x`, `y` et `résultat`, seront définies. Pour vous aider à définir les formats de ces variables, veuillez remplir le tableau QII-1. Rappelons que $6^6 = 46656$ et $2^{16} = 65536$.

Variables	Valeur minimale	Valeur maximale	Format minimum (bits)	Format retenu pour le programme
`x`				
`y`				
`résultat`				

Tableau QII-1 : Etude du format des variables

Q2. Donnez la définition des trois variables correspondantes en langage d'assemblage avec une initialisation. Précisez le numéro de la partie concernée dans le programme squelette. **Pour la suite, aucune autre variable ne devra être déclarée.**

Q3. Ecrivez le programme correspondant appelé *puis_DST.asm* **avec interdiction d'utiliser un sous-programme.** Vous devrez par ailleurs utiliser l'instruction `mul`. Vous ne traiterez pas le cas particulier x^0.

Version 2. Ecriture d'un programme principal avec traitement du cas particulier

Q1. Modifiez votre programme pour qu'il traite maintenant le cas particulier x^0 (nouvelle partie 2).

Version 3. Ecriture d'un programme principal avec traitement des cas particuliers et affichage d'un message.

Q1. Modifiez le programme précédent (déclaration des variables et code) pour qu'il affiche un message si l'un des deux nombres x ou y dépasse la valeur maximale autorisée et pour que le calcul ne se fasse pas. Vous vous inspirerez du programme *affiche.asm* de l'annexe 3 pour l'appel système pour l'affichage.

Version 4. Transformation du programme de la version 2 en sous-programme

Q1. Transformez votre programme de la version 2 (*i.e.* calcul d'une puissance avec traitement du cas particulier y = 0 mais pas de test de valeur maximale et pas d'affichage du message d'erreur) en sous-programme que vous appellerez *puis_sp*.

Ecrivez ensuite le programme l'appelant (nouvelle partie 2 à écrire). **Le passage des paramètres se fera obligatoirement par valeur et par registre. Un squelette de sous-programme est fourni en annexe 4 (partie 3 à écrire).**

Q2. Bonus. Transformez votre programme pour que le passage se fasse par valeur et par la pile.

Annexe 2 : Squelette de programme en langage d'assemblage 8086

```
; Nom du programme : puis_DST.asm
; Fonction : calcul de x^y

            IDEAL
            DOSSEG
            MODEL TINY
            P8086

            STACK 100h

            DATASEG
```

Partie 1

```
            CODESEG
; programme principal

debut:      mov ax,@data
            mov ds,ax
            mov es,ax

; corps du programme
```

Partie 2

```
            mov ah,4ch
            int 21h
END debut
```

Annexe 3 : Squelette du programme affiche.asm en langage d'assemblage 8086

```
; Nom du programme : affiche.asm
; Fonction : affichage d'une chaîne de caractères

        IDEAL
        DOSSEG
        MODEL TINY
        P8086

        STACK 100h

        DATASEG

msg1    DB "ceci est un message ! ",10,13, "$"

        CODESEG
; Programme principal

debut:  mov ax,@data
        mov ds,ax
        mov es,ax

; Corps du programme

        mov ah,9
        mov dx,OFFSET msg1
        int 21h

; Terminaison du programme par appel système
        mov ah,4ch
        int 21h
END debut
```

Annexe 4 : Squelette de sous-programme

```
; Déclaration d'un sous-programme
PROC puis_sp NEAR

        Partie 3

fin_sp:
ENDP puis_sp
```