**Exercices en assembleur**

**Exercice 2 : fonction produit scalaire**

. intel\_syntax noprefix

. text

. global prod\_scalaire

. type prod\_scalaire , @function

prod\_scalaire :

#RDI : adresse du premier tableau d’octets

#RSI : adresse du second tableau d’octets

#RDX : longueur des tableaux (int -> 32 bits -> EDX)

PUSH RBP

MOV RBP, RSP

MOV R8, 0

MOV RAX,0

MOV R9,0

MOV R10,0

loop :

CMP R8D, EDX # on compare la valeur de l’indice à la taille , s’il la dépasse , c’est fini

JGE end # on jump donc vers l’étiquette de fin

MOV AL , byte ptr [ RDI + R8] # la multicplication se fait entre une opérande et le contenu de RAX

MOV R9B, byte ptr [ RSI + R8] # on doit donc mettrela val d’ une de nos deux opérandes dans RAX

IMUL R9B #ici j’ai pris la fonction IMUL avec 8 bits , le résultat va dans AX

ADD R10, RAX # ici on ajoute à R10 le résultat de la multiplication précédente

MOV RAX,0 #on remet RAX à 0 pour faire la multiplication des deux prochains octets

INC R8 # on incrémente R8 pour aller à l’indice suivant dans les tableaux

JMP loop

MOV RAX, R10 # à la fin des deux tableaux , R10 contient la somme des produits et on

# place le résultat dans RAX qui est retourné par la fonction

end :

POP RBP

RET

**Exercice 3 : fibonacci**

. intel\_syntax noprefix

. text

. global Fibo

. type Fibo , @function

Fibo :

#RDI : contient n ( indice du nombre)

PUSH RBP

MOV RBP,RSP

MOV R8,0

MOV R8,1

MOV RAX ,0

MOV R9,0

MOV R10,0

MOV R10,1

Loop :

CMP R8, RDI

JGE end\_loop

PUSH R10

ADD R10,R9

POP R9

INC R8

JMP Loop

end\_loop :

MOV EAX, R10D

end :

POP RBP

RET

**Exercice 4 : palindrome**

. intel\_syntax noprefix

. text

. global palindrome

. type palindrome , @function

palindrome :

#RDI : adresse de la chaine de caractères

PUSH RBP

MOV RBP,RSP

MOV R8,0

MOV R9,0

MOV R10,0

MOV R11,0

MOV RAX,0

MOV RAX,1

Loop1 : ( première boucle pour connaitre la taille de la chaine)

MOV R9B , byte ptr [ RDI + R8]

CMP R9B,0

JE end\_loop1

INC R8

end\_loop1 :

Loop2 :

MOV R10B , byte ptr [ RDI + R8 – 1]

MOV R11B, byte ptr [ RDI ]

CMP R11 , R10

JE continue

MOV RAX,0

JMP end

continue :

INC RDI

DEC R8

end :

POP RBP

RET

**Exercice 5 : Division**

. intel\_syntax noprefix

. text

. global division

. type division, @function

division :

#RDI : contient le nombre a => 32 bits => EDI

#RSI : contient le nombre b => 32 bits => ESI

PUSH RBP

MOV RBP,RSP

MOV R8, 0

MOV R8D, EDI

MOV RAX,0

Loop :

CMP R8D,ESI

JLE end

SUB R8D,ESI

INC RAX

JMP Loop

end :

POP RBP

RET

**Exercice 6 : Prefixe\_commun**

. intel\_syntax noprefix

. text

. global Prefixe\_commun

. type Prefixe\_commun, @function

Prefixe\_commun :

#RDI : adresse du tab 1 => t1

#RSI : adresse du tab 2 => t2

#RDX : taille des tableaux => 32 bits => EDX

PUSH RBP

MOV RBP , RSP

MOV RAX,0

MOV RAX, 1

MOV R8,0

Loop :

CMP R8D , EDX

JGE end

MOV R9B , byte ptr [ RDI + R8 ]

MOV R10B , byte ptr [ RSI + R8]

CMP R9B , R10B

JE same

MOV RAX , 0

JMP end

same :

INC R8

JMP Loop

(MOV RAX,R8)

end :

POP RBP

RET

**Exercice 7 : function F\_deux**

. intel\_syntax noprefix

. text

. global F\_deux

. type F\_deux , @function

F\_deux :

#RDI : contient le nombre n à diviser

PUSH RBP

MOV RBP , RSP

MOV EAX , 0

MOV RCX , 1

Loop :

MOV RDX, RDI

AND RDX , RCX

JNZ end

INC EAX

ADD RCX,RCX

JMP Loop

end :

POP RBP

RET

**Exercice 8 : histogramme version 1**

. intel\_syntax noprefix

. text

. global histo

. type histo , @function

histo :

#RDI : adresse du tableau d’octets, ts

#RSI : taille du tableau ts

# RDX : adresse du tableau de 256 entiers de 64 bits, th

PUSH RBP

MOV RBP,RSP

PUSH R13

PUSH R14

MOV R9,256 #taille du tableau

MOV R10,0 #index parcours th

MOV R11,0 #index parcours ts

MOV R13,0 #val de ts

MOV R14,0 #val de th

Loop\_th:

CMP R10,R9

JGE loopts

MOV qword ptr [ RDX + 8\*R10], 0

INC R10

JMP Loop\_th

loopts :

CMP R11,RSI

JGE end

MOV R13B , byte ptr [ RDI + R11]

MOV R14, qword ptr [ RDX + 8\*R13]

INC R14

MOV qword ptr [ RDX+ 8\*R13],R14

INC R11

JMP loopts

end : POP R14 POP R13 POP RBP RET

**Exercice 8 : histogramme version 1**

. intel\_syntax noprefix

. text

. global histo

. type histo , @function

histo :

#RDI : adresse du tableau d’octets, ts

#RSI : taille du tableau ts

# RDX : adresse du tableau de 256 entiers de 64 bits, th

PUSH RBP

MOV RBP,RSP

MOV RAX,0

Loop1:

CMP RAX,0x100

JAE suite1

MOV qword ptr[RDX+8\*RAX],0

INC RAX

JMP Loop1

suite1 :

MOV RAX,0

Loop2:

CMP RAX,RSI

JAE suite2

MOV RCX,0

MOV CL,byte ptr[RDI+RAX]

INC qword ptr[RDX +8\*RCX]

INC RAX

JMP Loop2

suite2 :

POP RBP

RET

**exercice 9 : function count\_ok**

. intel\_syntax noprefix

. text

. global count\_ok

. type count\_ok , @function

count\_ok :

PUSH RBP

MOV RBP,RSP

MOV RAX,0

Loop :

CMP byte ptr [ RDI] , 0

JE end

CMP byte ptr [ RDI ] , ‘o’

JNE continue

CMP byte ptr[ RDI +1] , ‘k’

JNE continue

INC RAX

INC RDI

continue

INC RDI

JMP loop

end :

POP RBP

RET