3. La suite de Fibonacci F0, F1, F2, . . . est définie ainsi :

F0 = 0

F1 = 1

F2 = F0 + F1

F3 = F1 + F2

F4 = F2 + F3

.

.

Fi+2 = Fi + Fi+1

Les premiers termes de cette suite sont donc 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, . . .

Implémenter en assembleur x86-64 une fonction fibonacci(n) calculant Fn à partir de

n. Par exemple, fibonacci(10) doit retourner 55.

Le résultat sera représenté sur 32 bits. En cas de dépassement arithmétique, la fonction

doit retourner ce résultat modulo 232

FIBONACCI

EN PSEUDO-CODE

Start : %% Rnb : le nombre n donné

Rinc <- 1

Rprec<- 0

Rcur <- 1

Rn <- 0 %% Rn contient le nombre cherché

Rid <- 2 %% On traite les cas de base 0 et 1 séparément

JMP Cas\_0 if Rnb == 0

JMP Cas\_1 if Rnb == 1

JMP Loop

Cas\_0: Rn <- 0

JMP end

Cas\_1 : Rn <- 1

JMP end

Loop : JMP valeur if Rid > Rnb

Rval <- Rcur

Rcur <- Rcur + Rprec

Rprec <- Rval

Rid <- Rid + Rinc

JMP Loop

Valeur: Rn <- Rcur

End:

EN ASSEMBLEUR

FIBONACCI

.intel\_syntax noprefix

.text

.global fibonacci

.type fibonacci, at function

Fibonacci: %% RDI : le nombre n donné (sur 32 bits -> EDI)

PUSH RBP

PUSH RBX

MOV RBP, RSP

MOV RBX, 0 %% correspond à l’élément 0 (cas de base)

MOV RAX, 1 %% correspond à l’élément 1 (cas de base)

MOV R8, 2 %% indice courrant ( on traite 0 et 1 séparément)

CMP EDI, 0

JE Cas\_0

CMP EDI, 1

JE Cas\_1

JMP Loop

Cas\_0 : MOV EAX, 0

JMP end

Cas\_1 : JMP end

Loop: JMP end if R8D > EDI

MOV R9D, EAX

ADD EAX, EBX

MOV EBX, R9D

INC R8D

JMP Loop

End: POP RBX

POP RBP

RET