CORRIGÉ

CY TECH	**Examen Session Normale** **02/05/2024**
E. Ansermin, R. Grignon, T. Guidini Gonçalves, M. Masmoudi	Architecture des ordinateurs
ING1 GI	Année 2023–2024

Modalités :

- Durée : 2h
- Aucun document n'est autorisé, calculatrice autorisée
- Annexe incluse
- Pages 7 et 8 à compléter et à rendre

Exercice 1 : QCM (3 pts) 0,5 chaque réponse correcte.

1. **Un circuit séquentiel est …**

(a.) un circuit logique où les sorties à l'instant t dépendent des entrées à l'instant t et de leur état interne.
b. un circuit logique où les sorties ne sont fonctions que des entrées.
c. un circuit logique où les sorties à l'instant t dépendent que des entrées à l'instant t.
(d.) un circuit logique construit au moyen des circuits combinatoires et de cellules élémentaires de mémorisation.

2. **Le complément de l'expression $A + \bar{B}.C$ est :** Loi de Morgan

a. $A.\bar{B} + \bar{A}.\bar{C}$
(b.) $\bar{A}.B + \bar{A}.\bar{C}$
c. $\bar{A}.\bar{B} + \bar{B}.\bar{C}$
d. $A.\bar{B} + \bar{A}.C$

$\overline{A + \bar{B}.C} = \bar{A} \cdot \overline{\bar{B}.C} = \bar{A} \cdot (B + \bar{C}) = \bar{A}B + \bar{A}.\bar{C}$

Le ET(·) est prioritaire

3. **Quels modes d'adressage sont corrects ?**

(a.) MOV AL, [BL]
b. INC 64, AX
(c.) MOV BX, 54
d. INC 01

4. **Le registre FLAG (drapeau ou registre d'État) est un ensemble de 16 bits. Quels flags nous rencontrons dans ce registre ?**

a. ZF (zero flag), SF (sign flag), OF (overflow flag), PF (pointer flag)
b. SF (sign flag), CF (carry flag), IF (index flag), OF (overflow flag)
(c.) CF (carry flag), OF (overflow flag), ZF (zero flag), SF (sign flag)
d. PF (pointer flag), CF (carry flag), IF (index flag), ZF (zero flag)

Exercice 2 : Codage (2 pts)

1. Soit le nombre codé suivant la norme IEEE 754 en simple précision et représenté en hexadécimal BFF33333. Coder ce nombre en décimal.

2. Effectuer l'opération suivante en base hexadécimale (marquez les retenues sur la feuille d'examen).

					Eventuelle retenue
	5	E	3	2	**Opérande 1**
+	F	1	A	C	**Opérande 2**
=	1	4	F	D	E **Résultat**

Exercice 3 : Circuit Logique : Cheminement de cartons (6 pts)

Partie A : Circuit combinatoire

Les cartons, en sortie de l'usine de fabrication sont acheminés sur 4 tapis roulants différents (A, B, C, D) jusqu'au chariot de manutention. Chaque tapis transporte des cartons de poids spécifique : A pour 100 Kg, B pour 150 Kg, C pour 250 Kg, et D pour 175 Kg. Chaque tapis ne peut acheminer qu'un seul carton à la fois. Il est équipé d'un capteur qui indique la présence d'un carton en émettant un signal 1. Pour gagner du temps par rapport à l'attente des cartons, un chariot est considéré comme rempli et prêt à être acheminé au poste suivant, lorsqu'il accueille au moins 50% de sa capacité.

On souhaite construire un circuit logique à 4 entrées (**A, B, C, D**) et une sortie **S** qui est égale à 1 lorsque le chariot est rempli à 50% ou plus, et égale à 0 dans le cas contraire.

1. Compléter la table de vérité pour un chariot de 700 kg (***Tableau 1***).
2. En utilisant le tableau de Karnaugh, donner l'équation simplifiée de **S**.
3. Considérant que les données des capteurs de tapis A, B, C et D doivent être envoyées à une seule destination Z pour simplifier le câblage et la gestion des signaux, proposez une solution en utilisant un multiplexeur à 2 entrées d'adresse.

Partie B : Circuit séquentiel

L'usine souhaite ajouter un mécanisme qui permet de transporter au maximum 3 cartons sur le chariot. Pour cela, elle souhaite concevoir un circuit séquentiel à base de bascule T pour incrémenter le nombre de cartons sur le chariot (voir ***Schéma 1*** incomplet). La sortie du circuit, notée S, sera une valeur comprise entre 0 et 3 et codée sur deux bits, notés Q1 et Q0. Lorsque la valeur de S atteint 3, cela indique que le chariot est plein et doit être acheminé au poste de vérification. On considère qu'un front montant d'horloge correspond à l'ajout d'un carton au chariot.

1) Compléter la table de vérité (***Tableau 2***). Une bascule T possède une entrée T (*Toggle*) qui contrôle le changement d'état de la bascule. Si T=0, la sortie garde sa valeur. Si T=1, la bascule change d'état d'une période d'horloge à la période suivante.

2) Déduire les équations logiques de T0 et T1.

3) Compléter le schéma de câblage correspondant (***Schéma 1***). 1.0

Exercice 4 : Mémoires (4 pts)

On dispose d'une mémoire vive (RAM) de 8 Mbit, d'une mémoire morte (ROM) de 8 Mbit et de deux périphériques (P1 et P2) adressables respectivement sur 8 Ko et 4 Ko. On désire les rendre accessibles à un microprocesseur via les bus d'adresse (24 bits), de données (8 bits) et de commande. La RAM sera située dans les adresses les plus faibles, viendront ensuite la ROM et les deux périphériques.

1. Déterminer combien de fils (bits) seront utilisés pour adresser chaque mémoire et chaque périphérique. 0,25 chaque
2. Les quatre bits les plus significatifs du bus d'adresse du microprocesseur (A23 à A20) seront utilisés pour sélectionner parmi les quatre composants à connecter au microprocesseur, comme indiqué dans le tableau ci-dessous. 0,25 chaque

A23	A22	A21	A20	Composant
0	0	0	1	RAM
0	0	1	0	ROM
0	1	0	0	P1
1	0	0	0	P2

1 Mbit = 1/8 Mo

RAM : 8 Mbit = 8 · 1/8 Mo = 1 Mo = 2^{20} octets

ROM : Pareil, 20 fils (bits)

P1 = 8 Ko = $2^3 \cdot 2^{10} = 2^{13}$ → 13 fils

P2 = 4 Ko = $2^2 \cdot 2^{10} = 2^{12}$ → 12 fils

Donnez la représentation de l'espace mémoire (les adresses minimale et maximale de chaque composant en hexadécimal).

3. L'adresse mémoire (1F2)16 est utilisée dans les 4 composants. Donnez l'adresse correspondante en hexadécimal pour chaque composant. 0,25 chaque

Exercice 5 : Jeux d'instruction (5 pts)

Dans cet exercice, vous devez indiquer quel est l'état des registres IP (instructions), AX, BX et CX (registres), SP (pile), le flag Z (zéro), ainsi que le contenu de la pile, **après** chaque instruction du code.

Votre source de travail est le code assembleur donné ci-après. Pour chaque ligne d'instruction se trouve les octets de code associés pour que vous puissiez créer le code en mémoire.
La première instruction visible du **code principal** se situe à l'adresse **0x100**.
La première instruction de la fonction **add2IntEven** se trouve à l'adresse **0x200**.

Nous vous rappelons que les registres IP, SP (pointeur de pile), AX, BX et CX sont des registres 16 bits, que chaque adresse de code est sur 16 bits également, et chaque mot mémoire contient 1 octet.
Le code exécuté est l'appel d'une fonction qui prend en entrée 2 valeurs entières, effectue la somme et retourne un booléen (au sens langage C du terme) pour indiquer si cette somme est paire.

1F2 = *0001 1111 0010

RAM : 100 * 1F2

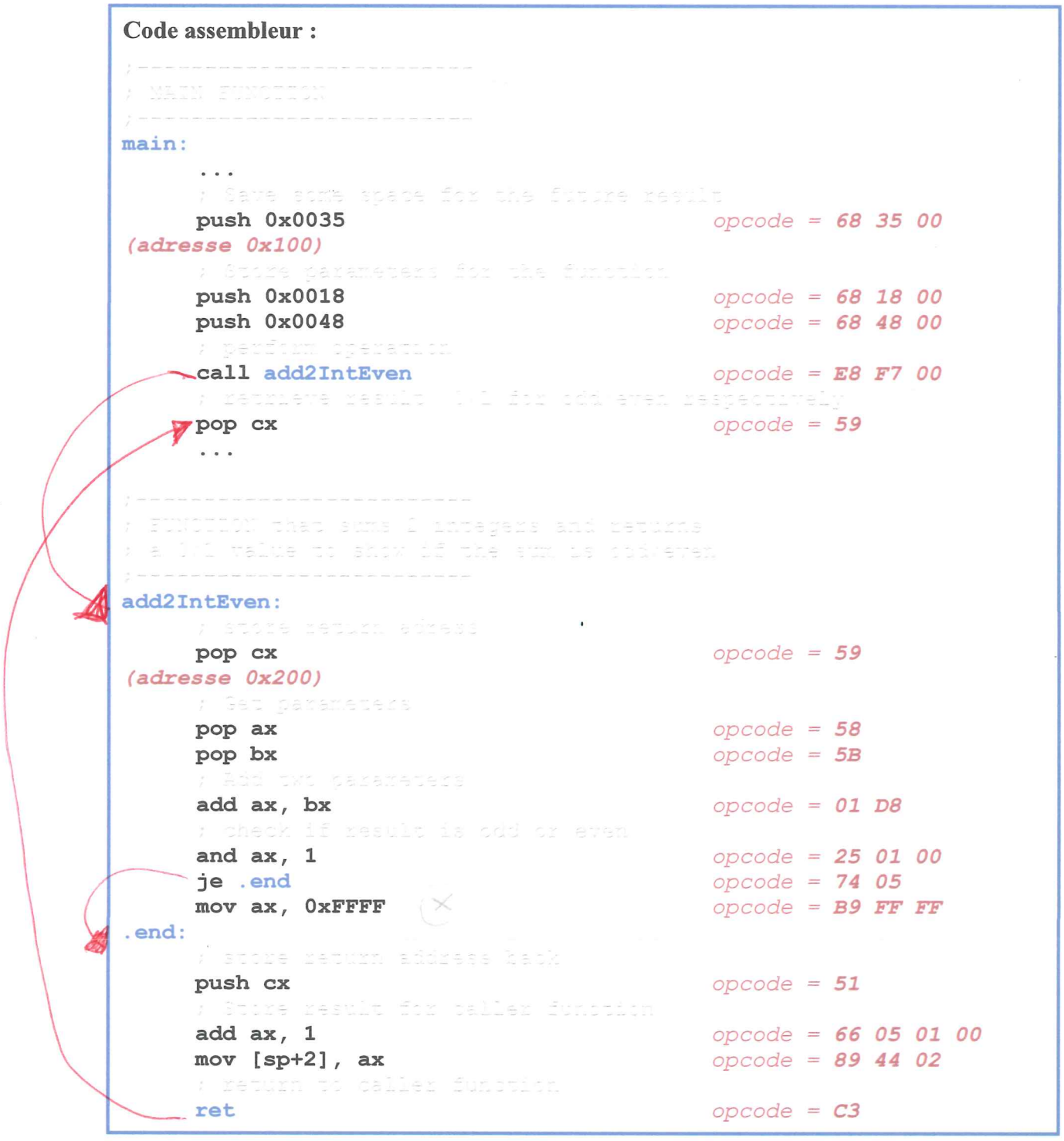
ROM : 200 1F2

P1 : 4000 1F2

P2 : 8000 1F2

Information : l'instruction *'call'* permet de faire un saut à une adresse d'une fonction, mais pour cela elle se charge silencieusement d'empiler l'adresse de l'instruction qui suit l'instruction *'call'*. Il y a donc une instruction *'push'* implicite dans ce cas.
Information : l'instruction ***'ret'*** permet de revenir à la fonction appelante, justement en dépliant l'adresse de retour. C'est l'adresse qui a été empilée lors de l'instruction 'call'. Il y a donc un *'pop'* implicite qui est exécuté ici.

Code assembleur :



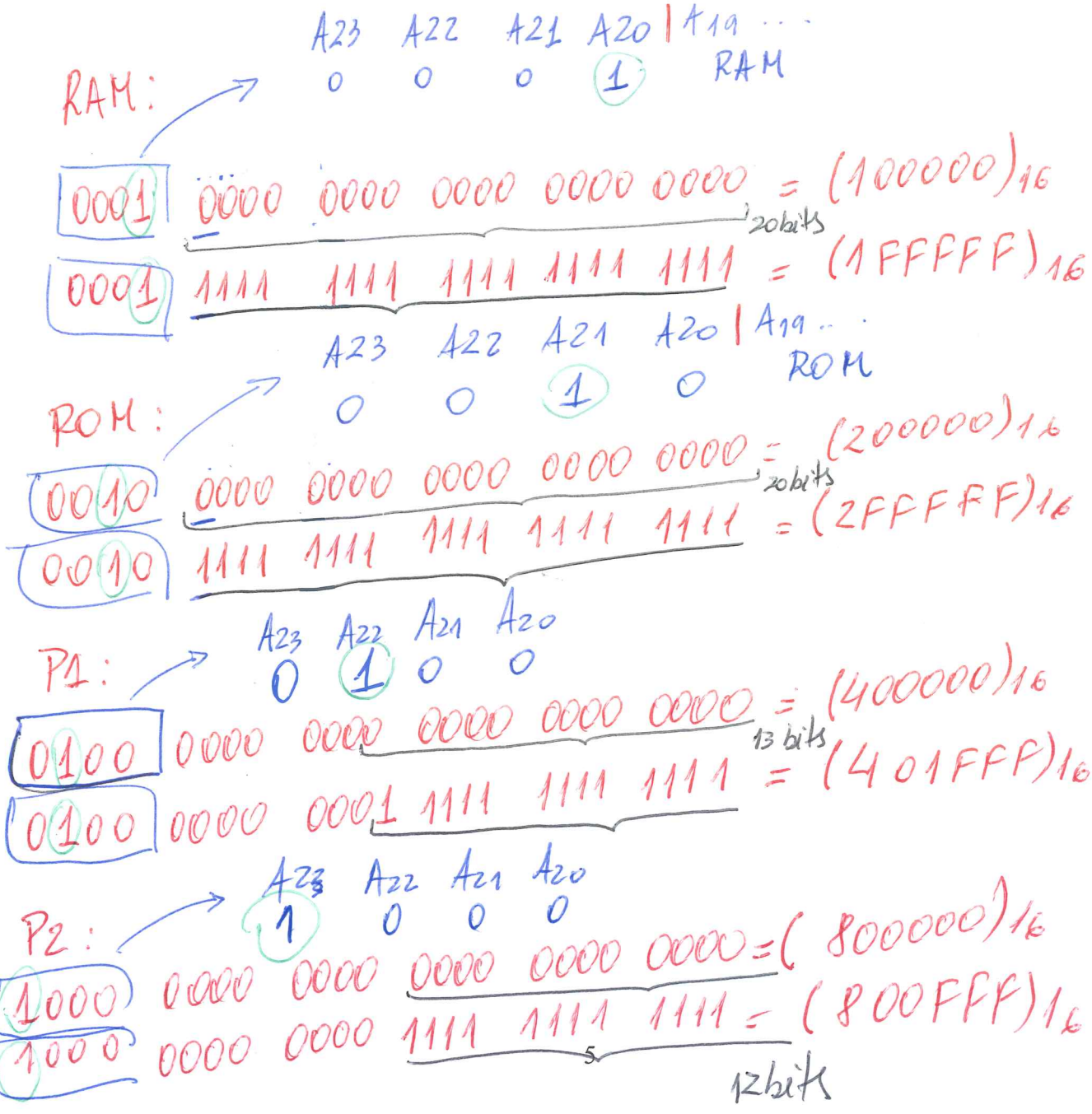
```
main:
        ...
        push 0x0035                          opcode = 68 35 00
(adresse 0x100)
        push 0x0018                          opcode = 68 18 00
        push 0x0048                          opcode = 68 48 00
        call add2IntEven                     opcode = E8 F7 00
        pop cx                               opcode = 59
        ...

add2IntEven:
        pop cx                               opcode = 59
(adresse 0x200)
        pop ax                               opcode = 58
        pop bx                               opcode = 5B
        add ax, bx                           opcode = 01 D8
        and ax, 1                            opcode = 25 01 00
        je .end                              opcode = 74 05
        mov ax, 0xFFFF                       opcode = B9 FF FF
.end:
        push cx                              opcode = 51
        add ax, 1                            opcode = 66 05 01 00
        mov [sp+2], ax                       opcode = 89 44 02
        ret                                  opcode = C3
```

L'état initial est indiqué dans le tableau ci-dessous (**Tableau 3**). La première instruction du code est exécutée et l'état après exécution est indiqué également. Compléter le **Tableau 3** correspondant aux contenus des différents registres, sachant que chaque ligne représente une étape d'exécution du code assembleur précédent.

Annexe : Assembleur

Instruction	Description
mov destination, source	Copie le contenu de **source** dans **destination**
push source	- copie le contenu de **source** au sommet de la pile - commence par décrémenter **sp** de 2 puis effectue la copie
pop destination	- copie les 2 octets qui se trouvent au sommet de la pile dans **destination** - commence par effectuer la copie puis incrémente **sp** de 2
add destination, source	Ajoute le contenu de **source** à **destination**
and destination, source	Effectue un ET logique bit à bit entre **source** et **destination**. Le résultat est stocké dans **destination**.
Je *label*	Je (Jump if Equal) : fait un saut au label spécifié si et seulement si ZF =1.



100 150 250 175 **Pages à compléter et à rendre**

$350 \leq S \leq 700$

A	B	C	D	S	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	425 Kg
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	1	400 Kg
0	1	1	1	1	575 Kg
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	350 Kg
1	0	1	1	1	525 Kg
1	1	0	0	0	
1	1	0	1	1	425 Kg
1	1	1	0	1	500 Kg
1	1	1	1	1	675 Kg

Tableau 1

T1	T0	Q1	Q0	
0	1	0	0	→ 0
1	1	0	1	→ 1
0	1	1	0	→ 2
1	1	1	1	→ 3
0	1	0	0	→ 0

Tableau 2

CD \ AB	00	01	11	10
00			1	
01			1	1
11		1	1	1
10			1	1

$S = BC + CD + AB + ABD$

T1

Q1 \ Q0	0	1
0	0	1
1	0	1

T0

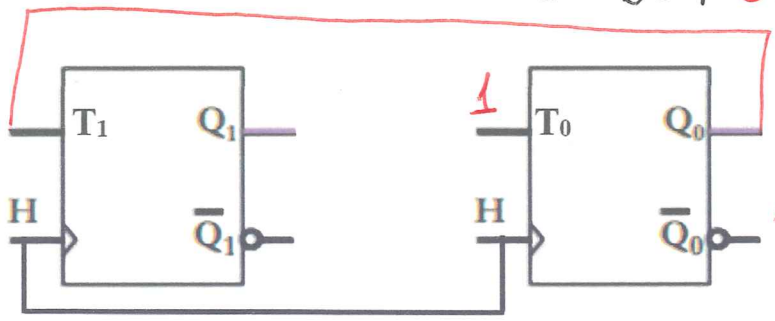
Q1 \ Q0	0	1
0	1	1
1	1	1

$T_0 = 1$

$T_1 = \bar{Q}_1 \cdot Q_0 + Q_1 \cdot Q_0$

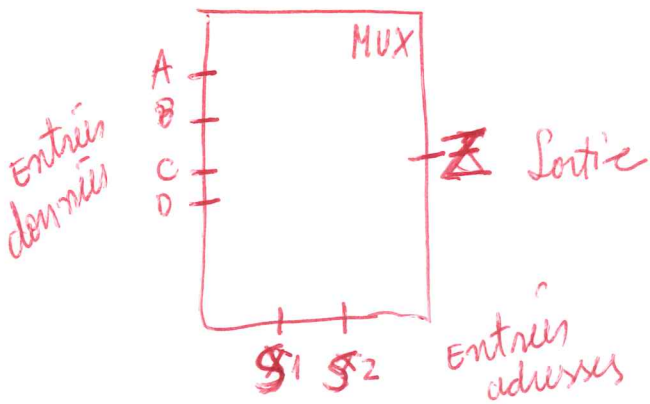
$T_1 = Q_0 (\bar{Q}_1 + Q_1)$

$T_1 = Q_0$



1 pt

Schéma 1



14 4 1 2 9 2 FFFA à FFFF

	Instruction exécutée	IP	AX	BX	CX	SP	Z	Stack (octet de droite avec l'adresse la plus haute = 0xFFFF)
1	*Etat initial*	01 00	00 00	00 00	00 00	FF FE	0	00 00
2	push 0x0035 68 35 00	01 03	00 00	00 00	00 00	FF FC	0	0035 00 00
3	push 0x0018 68 18 00	01 06	00 00	00 00	00 00	~~FF FA~~	0	00 18 00 35 00 00
4	push 0x0048 68 48 00	01 09	00 00	00 00	00 00	~~FF F8~~	0	00 48 00 18 00 35 00 00
5	call add2IntEven E8 F7 00	02 00	00 00	00 00	00 00	~~FF F6~~	0	01 0C 00 48 00 18 00 35 00 00
6	pop CX 59	02 01	00 00	00 00	~~01 0C~~	~~FF F8~~	0	00 48 00 18 00 35 00 00
7	pop AX 58	02 02	~~00 48~~	00 00	01 0C	~~FF FA~~	0	00 18 00 35 00 00
8	pop BX 5B	02 03	00 48	~~00 18~~	01 0C	~~FF FC~~	0	00 35 00 00
9	add AX, BX 01 D8	02 05	~~00 60~~	00 18	01 0C	"	0	"
10	and AX, 1 25 01 00	02 08	~~00 00~~	00 18	01 0C	"	~~0~~ 1	"
11	je .end 74 05	02 0D	"	"	"	"	1	"
12	push CX 51	02 0E	"	"	"	~~FF FA~~	1	01 0C 00 35 00 00
13	add AX, 1 66 05 01 00	02 12	~~00 01~~	"	"	"	~~0~~	"
14	mov [SP+2], AX 89 44 02	02 15	"	"	"	"	0	01 0C 00 01 00 00
15	ret C3	01 0C	"	"	"	~~FF FC~~	0	00 01 00 00
16	pop CX 59	01 0D	"	"	~~00 01~~	~~FF FE~~	0	00 00

Tableau 3

14 lignes

1.0 pour chaque 3 lignes