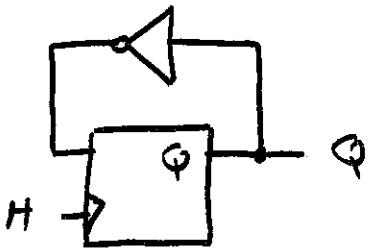
Question 1

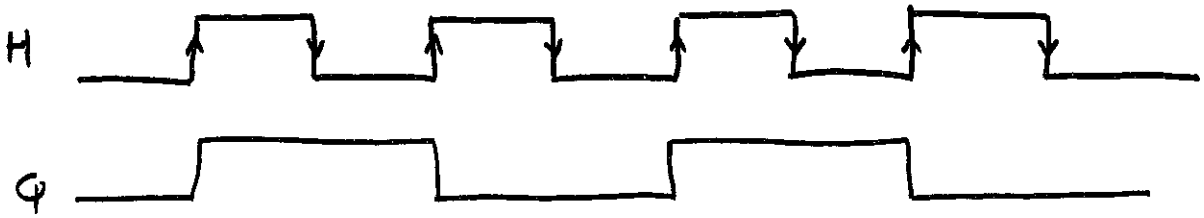
15

1.1)



15

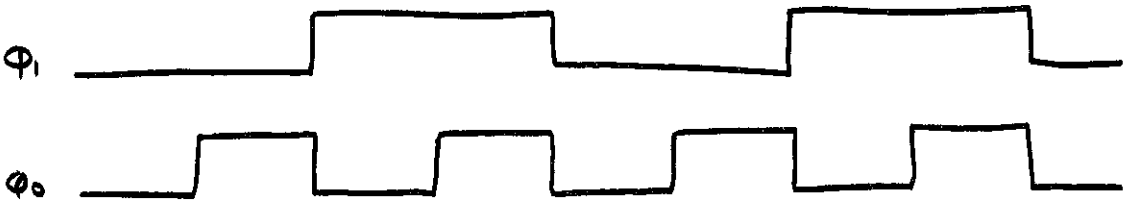
1.2)



La fréquence du signal Q est égale à la moitié de celle de H.

15

1.3)

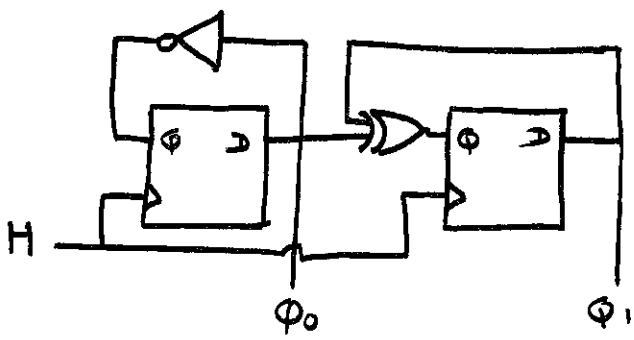


Pour calculer la table de vérité, il faut imaginer que les nouvelles valeurs sont calculées au moyen des anciennes

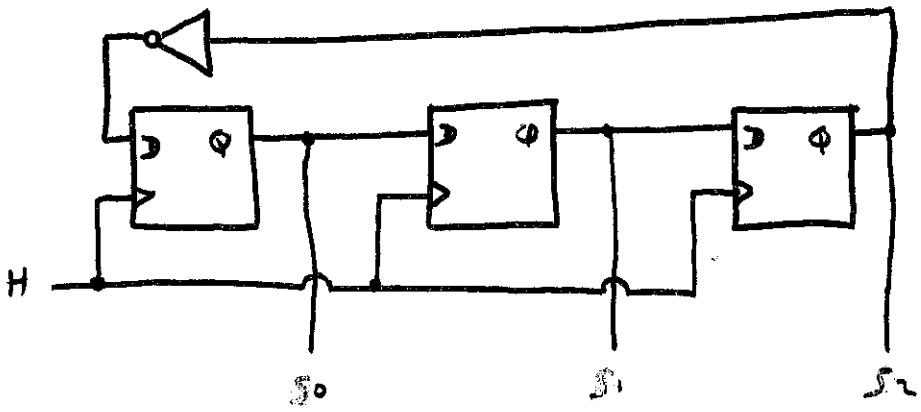
D_0 \ Q_1	0	1
0	1	1
1	0	0

D_1 \ Q_1	0	1
0	0	1
1	1	0

$D_0 = \overline{Q_0}$ $\qquad$ $D_1 = Q_0 \oplus Q_1$



1.4) 15



Question 2

2.1) 20

rrmovl rA,rB	pushl rA	ret
icode: ifun = M1[PC] rA:rB = M1[PC+1] valP = PC+2	icode: ifun = M1[PC] rA:rB = M1[PC+1] valP = PC+2	icode: ifun = M1[PC] valP = PC+1
valA = R[rA]	valA = R[rA] valB = R[%esp]	valA = R[%esp] valB = R[%esp]
valE = 0+valA	valE = valB + (-4)	valE = valB + 4
	M4[valE] = valA	valM = M4[valA]
R[rB] = valE	R[%esp] = valE	R[%esp] = valE
PC = valP	PC = valP	PC = valM

2.2) Voir feuille jointe 20

2.3) 10

Il faut deux accès en lecture, pour lire les anciennes valeurs de rA et rB, et deux accès en écriture, pour stocker les nouvelles valeurs de rA et rB, résultant de l'échange de leurs valeurs.

2.4) Dans le schéma actuel, l'une des deux valeurs pouvant être écrite dans la banque de registres est valM, valeur issue d'une lecture mémoire. Or, XCHGL n'utilise pas la mémoire. Il faut donc un nouveau bus allant de l'une des deux valeurs (par exemple valB) à la banque de registres, multiplexée avec valM. [10]

2.5) Voir feuille jointe. [10]

On met la valeur de rA dans rB de la même façon que pour RRMOVL. On suppose que, quand la mémoire n'est pas utilisée, valM est égale à valB, grâce à un nouveau multiplexeur.

Question 3

[20]

3.1) Le passage des paramètres par la pile permet de gérer les fonctions récentrantes. On empile les paramètres dans l'ordre inverse afin de pouvoir gérer les fonctions à nombre d'arguments variables : ainsi, le premier paramètre a toujours la même place par rapport au sommet de pile, ce qui permet à la fonction appelée de toujours savoir où il se trouve.

Les conventions "caller/callee save" fixent qui, de la fonction appelante ou de la fonction appelée, est responsable de la sauvegarde de certains registres.

Fonction appelante : EAX, ECX, EDX

Fonction appelée : EBX, ESI, EDI et EBP

3.2) [20]

```
    .pos 0
    irmovl  200, %esp   #Initialisation de la pile
    irmovl  5, %eax
    pushl   %eax        # Empilage de la constante 5
    call    fact
    iaddl   4, %esp     # Fonctionnellement inutile ici, mais plus propre
    rmmovl  %eax, a
    halt
    .align  4           # Alignement sur un mot de la zone de données
a:  .long   0           # "a" est l'étiquette de la zone mémoire
```

3.3) [30]

```
fact2: mrmovl  4(%esp), %edx    # %edx = n (paramètre d'appel)
    irmovl  1, %eax             # %eax = 1 (valeur de retour)
    rrmovl  %edx, %ecx
    subl    %eax, %ecx          # %ecx = (n-1)
    jle     fin2                # Si %ecx <= 0, donc si n <= 1
    pushl   %edx                # Empilage 1er paramètre de mul : n
    pushl   %ecx                # Empilage paramètre de fact2(n-1)
    call    fact2               # Récursion
    iaddl   4, %esp             # Dépilage paramètre de fact2(n-1)
    pushl   %eax                # Empilage 2ème paramètre de mul : fact2(n-1)
    call    mul
    iaddl   8, %esp             # Dépilage des deux paramètres de mul
fin2: ret
```

```
fact: mrmovl  4(%esp), %edx   # %edx = n (paramètre d'appel)
      irmovl  -1, %eax        # %eax = -1 (valeur de retour)
      andl    %edx, %edx      # Test de n
(*)   jl      fin             # si: n >= 0
      pushl   %edx            # Empilage paramètre de fact2
      call    fact2
      iaddl   4, %esp         # Dépilage paramètre de fact2
fin:  ret                     # Valeur de retour de la récursion
```

Note: les 4 instructions du bloc (*) peuvent être remplacées par l'unique instruction : " jge fact2 ", du fait que les contextes de pile de fact2 et fact sont identiques.

— Feuille à détacher et à rendre avec la copie —

20

10

Numéro d'anonymat : CORRIGÉ

```
################ Fetch Stage     ###################################

# Does fetched instruction require a regid byte?
bool need_regids =
icode in { OPL, IOPL, POPL, IRMOVL, RMMOVL, MRMOVL };

# Does fetched instruction require a constant word?
bool need_valC =
icode in { IRMOVL, RMMOVL, MRMOVL, JXX, CALL, IOPL };

################ Decode Stage    ###################################

## What register should be used as the A source?
int srcA = [
icode in { RMMOVL, OPL } : rA;
icode in { POPL } : RESP;
1 : RNONE; # Don't need register
];

## What register should be used as the B source?
int srcB = [
icode in { OPL, IOPL, RMMOVL, MRMOVL } : rB;
icode in { POPL, CALL } : RESP;
1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
int dstE = [
icode in { IRMOVL, OPL, IOPL } : rB;
icode in { POPL, CALL } : RESP;
1 : RNONE;  # Don't need register
];

## What register should be used as the M destination?
int dstM = [
icode in { MRMOVL, POPL } : rA;
1 : RNONE;  # Don't need register
];
```

RRMOVL, PUSHL, XCHGL

RRMOVL, PUSHL, XCHGL

RET

XCHGL

PUSHL, RET

RRMOVL, XCHGL

PUSHL, RET

XCHGL (avec valM contenant valB si la mémoire n'est pas utilisée)

```
################ Execute Stage   ###################################

## Select input A to ALU
int aluA = [
icode in { OPL } : valA;
icode in { IRMOVL, RMMOVL, MRMOVL, IOPL } : valC;
icode in { CALL } : -4;
icode in { POPL } : 4;
# Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
icode in { RMMOVL, MRMOVL, OPL, IOPL, CALL, POPL } : valB;
icode in { IRMOVL } : 0;
# Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
icode in { OPL, IOPL } : ifun;
1 : ALUADD;
];

## Should the condition codes be updated?
bool set_cc = icode in { OPL, IOPL };

################ Memory Stage    ###################################

## Set read control signal
bool mem_read = icode in { MRMOVL, POPL };

## Set write control signal
bool mem_write = icode in { RMMOVL, CALL };

## Select memory address
int mem_addr = [
icode in { RMMOVL, CALL, MRMOVL } : valE;
icode in { POPL } : valA;
# Other instructions don't need address
];

## Select memory input data
int mem_data = [
icode in { RMMOVL } : valA;
icode == CALL : valP;
# Default: Don't write anything
];

################ Program Counter Update ############################

## What address should instruction be fetched at

int new_pc = [
icode == CALL : valC;
icode == JXX && Bch : valC;
1 : valP;
];
```