Durée 2 semaines Documents autorisés

Sujet d'examen

Architecture des ordinateurs

Décembre 2006

1 Circuit combinatoire

Exercice 1 — Table de vérité, formes normales et circuit logique.

On se donne la table de vérité suivante :

a	b	c	res
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

1. Donnez une forme normale associée à cette table.
2. Dessinez un circuit logique associé à cette table.

Correction.

1. On sélectionne les lignes de sortie égale à 1. À chaque ligne, on fait la conjonction des lignes d'entrées égales à 1 et de la négation des lignes d'entrées nulles. Reste à faire la disjonction du tout. Dans notre cas, on a :

$$\bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot b \cdot \bar{c}$$

   On peut soit utiliser des règles de simplifications soit juste regarder le tableau pour constater que $res = \bar{c}$
2. (ToDo)

2 Circuit séquentiel

À l'intersection d'une route de directions Nord/Sud et d'une route de direction Est/Ouest, des feux passent alternativement du rouge (signal de valeur 0) au vert (signal de valeur 1) sous l'action d'un interrupteur. Nous allons modéliser ce dispositif par des circuits.

On suppose disposer d'une ligne d'horloge t ayant pendant un demi-cycle de temps la valeur 0 puis la valeur 1 pendant le demi-cycle suivant, etc. Donc un cycle d'horloge est commence quand la ligne t passe de 1 à 0, se prolonge quand la ligne passe de 0 à 1 est se termine au début du cycle suivant. On suppose que le temps de passage des portes logiques est négligeable devant la durée de ce cycle. Cette ligne est reliée à tous les circuits construit dans cet exercice.

Les lignes NS et EO representent la couleur des feux Nord/Sud et Est/Ouest.

Un interrupteur est à la disposition des piétons désirant traverser. En actionnant cet interrupteur, on fait passer *tous* les feux au rouge puis une seule direction est mise au vert alors que l'autre reste rouge.

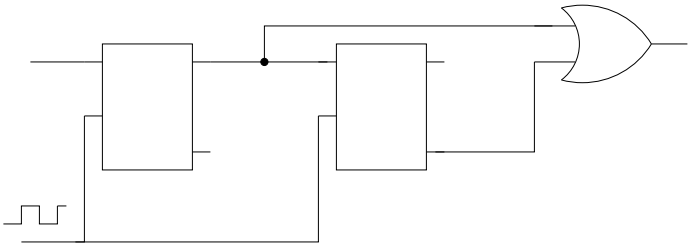
2.1 Modélisation de l'interrupteur

Lorsque le piéton actionne l'interrupteur, cela fait passer la ligne d'entrée x de la valeur 1 à la valeur 0 pendant un temps supérieur à un cycle d'horloge. On suppose que la ligne x revient ensuite automatiquement à la valeur 1.

Question. Donnez un circuit de lignes d'entrée x et t et de ligne de sortie y tel qu'à l'activation de l'interrupteur, la sortie y produise un signal 0 pendant un cycle d'horloge puis revient à sa valeur normale 1.

Indications : utiliser 2 bascules D.

Correction. Au repos $x = 1$ et $\overline{Q} = 0$, donc y est à 1. Lorsque x passe à 0, alors $y = 0$. Mais $\overline{Q}$ passe à 1 au cycle suivant, et donc y repasse à 1. Lorsque x repasse à 1, la ligne y reste à 1.



2.2 Modélisation du circuit des feux

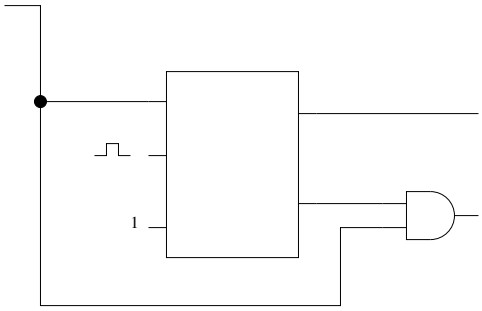
À l'aide d'une bascule JK, on veut modéliser des circuits mettant :
- alternativement les feux au rouge puis au vert quand la ligne de sortie y reste à 1 ;
- tous les feux au rouge pendant un cycle à réception du signal $y = 0$.

Construisez 3 circuits différents codant les comportements différents après l'actionnement de l'interrupteur :

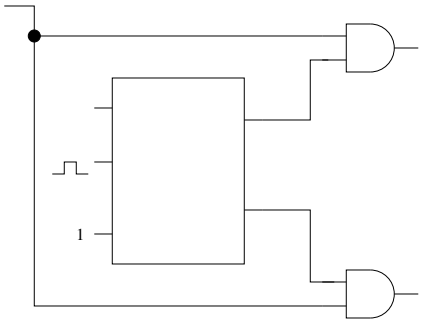
1. le feu de direction Nord/Sud est remis au vert.
2. le feu au vert avant l'actionnement de l'interrupteur est remis au vert.
3. le feu au rouge avant l'actionnement de l'interrupteur est remis au vert.

Correction.

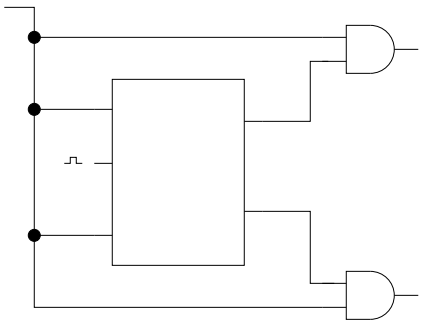
1. – Fonctionnement normal : $y = 1 \Rightarrow J = K = 1$: á chaque cycle, la valeur de Q (et de $\overline{Q}$) est inversée. Les feux passent donc alternativement au vert puis au rouge.
   – Lorsque le bouton est pressé passe à la valeur 0, ainsi qu'une entrée de la porte *and* A. À ce cycle d'horloge, Q prend la valeur 0. La porte *and* A place la valeur de sortie E/W à 0.
   – Au cycle suivant, on a à nouveau $J = K = 1$: Q prend la valeur 1 $\Rightarrow$ l'axe N/S passe au vert.



2. Ici, les deux portes *and* mettent la valeur des sorties N/S et E/W à 0 pendant un cycle. Toutefois, les sorties de la bascule JK continuent de changer de valeur une fois pendant ce cycle. Au cycle suivant, la valeur des sorties N/S et E/W est la même qu'avant le passage du piéton.



3. Ici, les entrées J et K sont reliées à l'entrée y. Pendant que l'on annule les sorties N/S et E/W, les entrées J et K sont elles aussi mises á la valeur 0. Les valeurs de sortie Q et $\overline{Q}$ ne sont donc pas modifiées pendant ce temps, mais seulement au cycle suivant (lorsque J et K sont remises à 1). Les valeurs prises par les sorties N/S et E/W au cycle suivant sont donc l'inverse de ce qu'elles étaient avant le passage du piéton.



3 Assembleur

Exercice 2 — Codage assembleur.

Un pangramme est une chaîne de caractères qui contient toutes les lettres de l'alphabet. Le pangramme suivant comporte 42 lettres :

```
Portez ce vieux whisky au petit juge blond qui fume
```

On suppose qu'une chaîne de caractères est stockée en mémoire à l'adresse `MMNN` et que cette suite d'octets se termine bien par un 0.

Écrivez un programme assembleur qui vérifie si une chaîne de caractères est un pangramme : ce programme se termine avec un 1 dans le registre `R4` si c'est le cas et 0 sinon. Notez que seules l'espace — code ascii 32 — et les lettres de l'alphabet (minuscules — code ascii compris entre 97 et 122 — ou/et majuscules — code ascii compris entre 65 et 90) peuvent composer un pangramme ; une chaîne de caractères contenant un autre type de caractères ne peut être un pangramme (i.e. 12321 n'en est pas un).

N'hésitez pas à indiquez les espaces mémoire dont vous supposez l'existence pour faire votre programme (par exemple, indiquez qu'à l'adresse NNMM se trouve des octets codant une information). On suppose qu'aucun registre n'est utilisé avant l'appel de votre programme et vous n'avez pas à gérer d'éventuelles erreurs comme la présence de caractères non alphabétiques dans votre chaîne.

Correction. On suppose qu'à l'adresse AABB commence un espace mémoire constitué de 26 octets consécutifs initialisés à 0.

```
      MV NN,R2
      MV MM,R3        ; on fait pointer RX1 sur le d\'ebut de la cha\^\i{}ne

boucle_0:
      LD RX1,R0
      MV R0,R1        ; mettre le code ascii courant dans R1

      MV  0,R7
      SUB R1,R7       ;  a-t-on atteind la fin de la cha\^\i{}ne
      JZ analyse      ; si oui on fait l'aralyse de la situation

      MV 122,R7       ; s'agit il bien d'un caract\'ere de code sup\'erieur \'a 122
      SUB R1,R7
      JC suivant

      MV 65,R7        ; s'agit il bien d'un caract\'ere de code inf\'erieur \'a 65
      SUB R7,R1
      JC suivant

      MV R0,R1        ; mettre le code ascii courant dans R1
      MV 65,R7
      SUB R7,R1       ; on d\'ecale tous les codes ascii de 65

      MV 26,R7
      SUB R1,R7
      JC majuscule    ; si le code ascii est sup\'erieur \'a 26
      JMP lettre

majuscule:
      MV R0,R1    ; mettre le code ascii courant dans R1
      MV 97,R7
      SUB R7,R1   ; on d\'ecale tous les codes ascii de 97
      JC suivant  ; s'agit il bien d'une lettre

lettre:           ;  si oui
      MV BB,R4
      MV AA,R5    ; on doit bien pointer sur la bonne cellule

lettre_0:
      DEC R1
      JZ lettre_2
      INC R4
      JC lettre_1
      JMP lettre_0
lettre_1:
      INC R5
      JMP lettre_0

lettre_2:
      LD RX2,R0
      MV 1,R7
```

```
      AND R7,R0
      ST R0,RX2

suivant:
      INC R2
      JC incRX1_0    ; dans ce cas, il faut incr\'ementer R3
      JMP boucle_0
incRX1_0:
      INC R3
      JMP boucle_0

analyse:
                     ; on fait la somme des~$26$ octets stock\'es \`a partir
                     ; de l'adresse AABB
      MV 26,R1       ; R1 va servir de  compteur pour parcourir le tableau

      MV 0,R5        ; R5 va contenir cette somme
      MV BB,R2
      MV AA,R3       ; on fait pointer RX1 sur le d\'ebut de la cha\^\i{}ne
analyse_0:
      LD RX1,R0
      ADD R0,R5      ; la somme se fait ici
      DEC R1         ; passer \`a la cellule suivante
      JZ fin         ; on a pris en compte l'ensemble des cellules
      INC R2
      JC analyse_1   ; dans ce cas, il faut incr\'ementer R3
      JMP analyse_0
analyse_1:
      INC R3
      JMP analyse_0
fin:                 ; derni\`ere chose, mettre le r\'esultat dans R4
      SUB R5,26
      JZ OK          ; si 26 lettres ont \'et\'e detect\'ees
      JMP PASOK      ; sinon
OK:
      MV 1,R4
PASOK:
      MV 0,R4
```

Jeu d'instructions. On dispose de registres 8 bits généraux nommés de $R0$ à $R7$. On dénomme `RX0` le registre 16 bits obtenu par la concaténation de `R1` et `R0` (`R1` octet de poids fort, `RX1` par `R2` et `R3`, `RX2` par `R4` et `R5` et `RX3` par `R6` et `R7`.

Notre microprocesseur dispose des instructions suivantes :

- `ST R0, RXn` — (*pour STore*) qui stocke la valeur du registre `R0` en mémoire à l'adresse (sur 16 bits) contenue dans le registre `RXn` ;
- `LD RXn, R0` — (*pour LoaD*) qui charge dans le registre `R0` la valeur stockée en mémoire à l'adresse (sur 16 bits) contenue dans le registre `RXn` ;
- `MV` ***arg***`,R`***n*** — (*pour MoVe*) qui charge dans le registre `R`***n*** la valeur de l'argument "***arg***". L'argument est soit une valeur immédiate, soit un registre ;
- `JMP label` — (*pour JuMP*) qui effectue un branchement à l'adresse (sur 16 bits) spécifiée par l'étiquette `label` ;
- `ADD R`***m***`, R`***n*** — (*pour ADD :-)*) qui additionne la valeur du registre `R`***n*** avec la valeur du registre `R`***m*** et stocke le résultat dans le registre `R`***n*** ;

- `SUB Rm, Rn` — (*pour SUBstract*) qui soustrait la valeur du registre `Rm` à la valeur du registre `Rn` et stocke le résultat dans le registre `Rn` ;
- `AND Rm, Rn` — qui fait la conjonction bit à bit des deux registres et range le résultat dans `Rn` ;
- `DEC Rn` — (*pour DECrement*) qui soustrait 1 à la valeur de `Rn` et stocke le résultat dans `Rn` ;
- `INC Rn` — (*pour INCrement*) qui ajoute 1 à la valeur de `Rn` et stocke le résultat dans `Rn` ;
- `NOT Rn` — qui inverse bit à bit le registre `Rn` et qui y stocke le résultat ;
- `JZ label` — (*pour Jump if Zero*) qui effectue un saut à l'adresse 16 bits spécifiée par l'étiquette `label` si l'opération précédente a donné un résultat nul ;
- `JC label` — (*pour Jump if Carry*) qui effectue un saut à l'adresse 16 bits spécifiée par l'étiquette `label` si l'opération précédente a engendrée une retenue (ou si le résultat est négatif).

Exercice 3 — Lecture et correction de code assembleur.

On se donne un code assembleur (voir page suivante).

1. Que fait ce code ?
2. Une erreur algorithmique a été faite par le programmeur, corrigez la.

```
.data
str:    .zero   11
i:      .long   463960
d:      .long   1
j:      .zero   4
.text .globl _start
_start: movl    i,%ebx
L1:     movl    i, %eax
        movl    $0,%edx
        movl    d,%ecx
        divl    %ecx
        push    %edx
        movl    $10,%eax
        mull    %ecx
        pop     %edx
        movl    %eax,d
        cmpl    %edx,%ebx
        jne     L1
        movl    $str, %ebx
        movl    d, %eax
        movl    $0,%edx
        movl    $10,%ecx
        divl    %ecx
        movl    %eax,d
L2:     movl    d, %eax
        movl    $0,%edx
        movl    $10,%ecx
        divl    %ecx
        movl    %eax,d
        movl    i, %eax
        movl    $0,%edx
        movl    d,%ecx
        divl    %ecx
        movl    %edx,i
        movl    j, %ecx
        movl    $0, %edx
        addl    $'0',%eax
        movb    %al, %ds:(%ebx,%ecx,1)
        inc     %ecx
        movl    %ecx,j
        movl    i, %eax
        movl    $0,%edx
        cmpl    %eax, %edx
        jne     L2
        movl    j, %eax
        addl    $str, %eax
        movl    $0, %edx
        movb    $0, %ds:(%edx,%eax,1)
        movl    $1, %eax
        movl    $0, %ebx
        int     $0x80
```

Correction. Le programme convertit un entier stocké au label i en une chaîne de raractères stockée au label str. En gros ça code salement ce qui suit :

```
#include<stdio.h>

char str[11] ;
int i = 463960 ;
int d=1 ;
int j=0 ;

int main(void){

  while(i%d!=i)
    d*=10 ;

  while(i!=0){
    d/=10 ;
    *(str+j++) = i/d +'0' ;
    i %= d ; /* la correction est i -= d*(i/d) ; */
  }

  *(str+j) = 0 ;
  return 0;
}
```

Avec quelques commentaires, ça donne :

```
.data
str:    .zero   11      /* r\'eserve de la place pour la cha\^\i{}ne finale */
i:      .long   463960  /* l'entier \`a coder en cha\^\i{}ne*/
d:      .long   1       /* pour stocker des puissances de 10 */
j:      .zero   4       /* un compteur */
.text
.globl _start
_start:

        movl    i,%ebx
L1:
        movl    i, %eax         /* on divise i par d */
        movl    $0,%edx         /* le quotient est dans %eax */
        movl    d,%ecx          /* le reste est dans %edx */
        divl    %ecx

        push    %edx            /* la valeur du reste est empil\'ee */
        movl    $10,%eax
        mull    %ecx            /* pour \'eviter d'\^etre perdu ici */
         movl    %eax,d          /* ici d = 10*d */
        pop     %edx
        cmpl    %edx,%ebx       /* on verifie si i%d = i */
        jne     L1
        movl    $str, %ebx      /* on place l'adresse de str pour s'en servir plus tard */
        movl    d, %eax
        movl    $0,%edx
        movl    $10,%ecx
        divl    %ecx            /* on divise d par 10 pour tomber juste*/
        movl    %eax,d          /* d = d/10 */
L2:
        movl    d, %eax         /* rebolotte */
```

```
movl    $0,%edx
movl    $10,%ecx
divl    %ecx            /* on divise d par 10 */
                        /* le quotient est dans %eax */
movl    %eax,d          /* d = d/10 */

movl    i, %eax         /* on divise i par d */
movl    $0,%edx
movl    d,%ecx
divl    %ecx            /* le reste est dans %edx */
movl    %edx,i          /* i = i mod d */
movl    j, %ecx         /* on place '0' \`a la fin de la cha\^\i{}ne */

movl    $0, %edx
addl    $'0',%eax   /* convertion du chiffre en caract\`ere ascii */
movb    %al, %ds:(%ebx,%ecx,1) /* troncation hardi et stockage dans str */

inc     %ecx
movl    %ecx,j   /* incr\`ementation de j */

movl    i, %eax
movl    $0,%edx
cmpl    %eax, %edx  /* si i est diff\'erent de 0, on recommence */
jne     L2

movl    j, %eax         /* on place '0' \`a la fin de la cha\^\i{}ne */
addl    $str, %eax
movl    $0, %edx
movb    $0, %ds:(%edx,%eax,1)

movl    $1, %eax        /* l'appel syst\`eme de sortie */
movl    $0, %ebx
int     $0x80
```