Département de génie électrique et de génie informatique
Université de Sherbrooke

ÉVALUATION FORMATIVE - SOLUTION

Exercice 1 Protocole I2C

On souhaite envoyer un octet à un esclave via le protocole I2C. Les paramètres sont les suivants:

- Adresse de l'esclave: 0x6F
- Données à transmettre: 0xFA

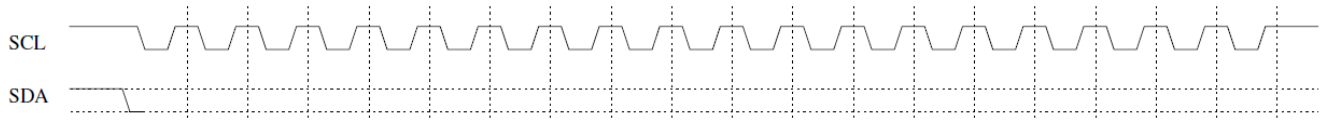
a) Quelle est la séquence précise d'opérations à réaliser pour effectuer ce transfert selon le protocole I2C?

Solution :

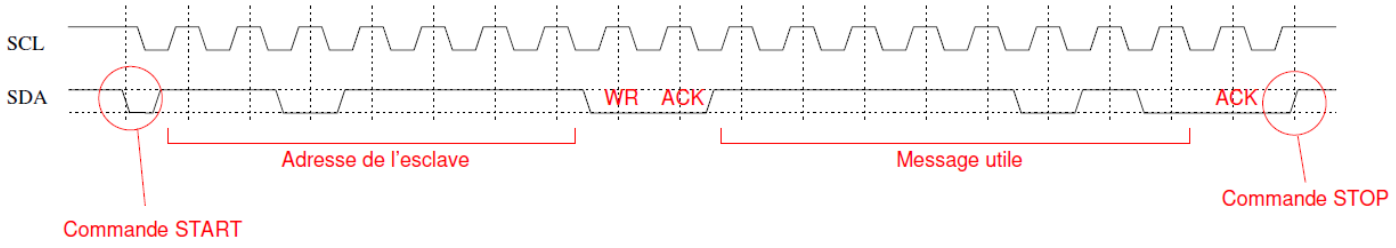
Séquence d'écriture I2C:

1. Commande START;
2. Envoi de l'adresse de l'esclave + bit d'écriture;
3. Réception de l'accusé de réception de l'esclave (bus forcé à zéro par l'esclave via ET-câblé);
4. Envoi du message utile;
5. Réception de l'accusé de réception;
6. Commande STOP.

b) Complétez le chronogramme ci-dessous pour illustrer le transfert complet, en indiquant par annotation les évènements importants.



Solution :



ACK = 0 si la lecture est bonne et 1 si la lecture n'est pas bonne.

C) Dessinez un chronogramme I2C de lecture d'un registre identifié par B7:B0 (MSB:LSB) d'un périphérique esclave à l'adresse A6:A0, et qui contient la valeur D7:D0. Utilisez les symboles S (start), P (stop), A (ack) et NA (no-ack) pour les bits de contrôle. Indiquez clairement (par une couleur ou autre) qui contrôle (maître ou esclave) le signal SDA.

Solution :

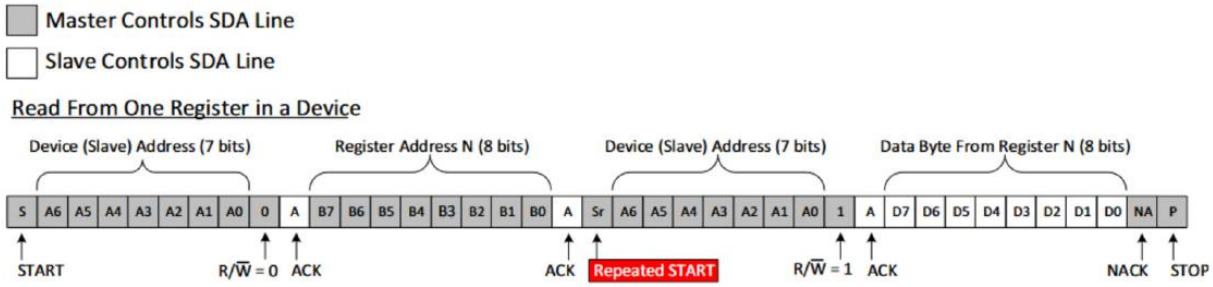


Figure 9. Example I²C Read from Slave Device's Register

Vous pouvez trouver la réponse à cette question à plusieurs endroits, entre autres à la figure 10 de la fiche technique de l'accéléromètre MMA8652 utilisé sur la carte MX3.

Notez qu'en lecture, on doit faire un Restart et répéter l'adresse du slave, contrairement à une écriture comme au procédural.

Exercice 2 Écriture de code assembleur

Considérez le code C suivant :

```
1 int valeur = 64;
2
3 int main() {
4       int n = 0;
5       int s = 0;
6
7       while ( s <= valeur) {
8       s = s + 2 * n + 1;          /* mise à jour de la somme */
9       n = n + 1;                  /* incrémentation de l'index */
10       }
11 }
```

a) Comment feriez-vous en assembleur pour calculer $2 \cdot n + 1$ sans avoir recours à une multiplication ni une addition?

Solution :

Décalage à gauche de 1 position (pour la multiplication par 2) et OU avec '1'. Cette dernière étape correspond à additionner '1' puisque le décalage à gauche laisse le bit de poids faible nécessairement à '0'.

b) Écrivez un programme en langage assembleur MIPS qui réalise la même fonction, de manière à ce que la valeur finale de n se trouve dans $t0 en fin d'exécution.

Solution :

```
1 .data
2 valeur: .byte 64              # autre format possible
3
4 .text
5
6 main:
7       la $a0, valeur          # Les 2 premières lignes chargent dans $t3 l'octet à l'adresse valeur
8       lbu $t3, 0($a0)         # On pourrait aussi les remplacer par lbu $t3, valeur($0) (mode
                                d'adressage non supporté)
9       li $t1, 0               # initialisation de s à 0
10      li $t0, 0               # initialisation de n à 0
11 boucle: sll $t2, $t0, 1      # décalage à gauche de 1 position, résultat placé dans $t2
12      ori $t2, $t2, 1         # on ajoute '1'
13      add $t1, $t1, $t2       # addition de 2n+1 à s
14      addi $t0, $t0, 1        # n=n+1
15      blt $t1, $t3, boucle    # Si s < valeur, on boucle
16      nop
17      li $v0, 10              # Fin du programme
18       syscall
```

c) Écrivez un programme qui change l'état des bits 15, 16 et 17 du mot de 32 bits à l'adresse 0x10010000.

Solution :

```
1 .data 0x10010000
2 valeur: .word 0xFA857E40      # Donnée fournie à titre d'exemple
3                               # alternative: valeur: .space 4
4
5 .text
6
7 main:
8       lw $t0, valeur          # chargement du mot de 32 bits à l'adresse valeur
9       li $t1, 0x00038000      # chargement du masque
10      xor $t0, $t0, $t1       # ou exclusif pour faire basculer les bits 17, 16 et 15
11      sw $t0, valeur          # Résultat retourné à l'adresse valeur
12
13      li $v0, 10              # Fin du programme
14      syscall
```

Exercice 3 Analyse de code assembleur

Soit le code assembleur suivant :

```
1 .data 0x10010000
2 var:   .word 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
3
4 .text
5
6 main:
7       la $t1, var
8       lw $t3, 4($t1)
9       lw $t4, 16($t1)
10      addu $t5, $t4, $t3
11
12      .end main
```

a) Donnez la valeur du registre de destination après les instructions aux lignes 7, 8 et 9.

Solution :

ligne 7: $t1, ligne 8: $t3, ligne 9: $t4

b) Commenter le code suivant.

Solution :

```
1 .data 0x10010000
2 var:   .word 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16     # Créer l'étiquette 'var' à la case mémoire
3        # 0x10010000 et les 16 premières cases mémoires convient les nombres 1 à 16.
4 .text
5
6 main:
7       la $t1, var             # Écrit l'adresse 0x10010000 dans $t1
8       lw $t3, 4($t1)          # Écrit le contenu (2) de la 5ème (var+4) case mémoire dans $t3
9       lw $t4, 16($t1)         # Écrit le contenu (5) de la 16ème (var+16) case mémoire dans $t3
10      addu $t5, $t4, $t3      # additionne $t3 et $t4 , puis écrit la réponse dans $t5
11
12      .end main
```

c) Que contient $t5 en fin d'exécution?

Solution :

Chaque mot occupe 4 octets, donc $t5 contient $2 + 5 = 7$.

Exercice 4 Architecture MIPS

a) Donner les modes d'adressage des instructions MIPS32.

Solution :

1. Adressage par registre

2. Adressage par valeur immédiate

3. Adressage par adresse mémoire et adresse mémoire avec décalage

4. Adressage par saut du PC (adresse programme)

b) Expliquez la différence entre une instruction, une pseudo-instruction et un mode d'adressage non-supporté.

Solution :

1. Instruction : correspond directement à une instruction sur 32 bits supportée par le processeur MIPS.

2. Pseudo-instruction : instruction non supportée par le processeur MIPS que l'assembleur décompose traduit en une ou plusieurs instructions supportées.

3. Mode d'adressage non supporté : l'instruction en assembleur utilise un mode d'adressage non supporté par le processeur MIPS; l'assembleur traduit en plusieurs instructions pour réaliser l'adressage en plus d'une étape.

c) Donnez des exemples de mode d'adressage non compatible nativement avec le MIPS32.

Solution :

lw $t1, tableau($t1)

ble $t1, $t2, boucle

d) Comment fait-on en langage assembleur MIPS pour transférer un mot de 32 bits de l'adresse 0x10010000 à l'adresse 0x100100F8?

Solution :

Le transfert direct mémoire-mémoire ne peut se faire par une seule instruction. Il faut d'abord

charger le contenu de l'adresse source dans un registre, puis le transférer à l'adresse destination.

Ceci peut se faire ainsi:

```
1 la $t0, 0x10010000
2 la $t1, 0x100100F8
3 lw $t2, 0($t0)
4 sw $t2, 0($t1)
```

Exercice 5 Microcontrôleurs

a) En quoi consiste la protection de contexte en programmation mixte C/assembleur?

Solution :

Ce sont les conventions qui assurent que la routine assembleur n'effectuera pas de changements dans l'état du processeur qui pourrait affecter le comportement du code C au retour. Il s'agit bien de conventions, car il y a plusieurs manières de fonctionner. Le compilateur XC32 se charge de sauvegarder certains registres ($s0-$s7) avant un appel vers une routine en assembleur et de les restaurer après le retour. Donc, une partie du contexte est automatiquement préservée. Si la routine en assembleur utilise d'autres registres, elle doit se charger explicitement de sauvegarder et restaurer leurs valeurs originales. D'autres compilateurs utilisent des conventions semblables, mais pas identiques.

b) Décrivez la convention d'appel d'une fonction assembleur en ce qui concerne les paramètres d'entrée et de retour.

Solution :

Avec le compilateur XC32, $a0-$a3 sont utilisés pour passer des paramètres à la routine assembleur, alors que $v0-$v1 sont utilisés pour les valeurs de retour.

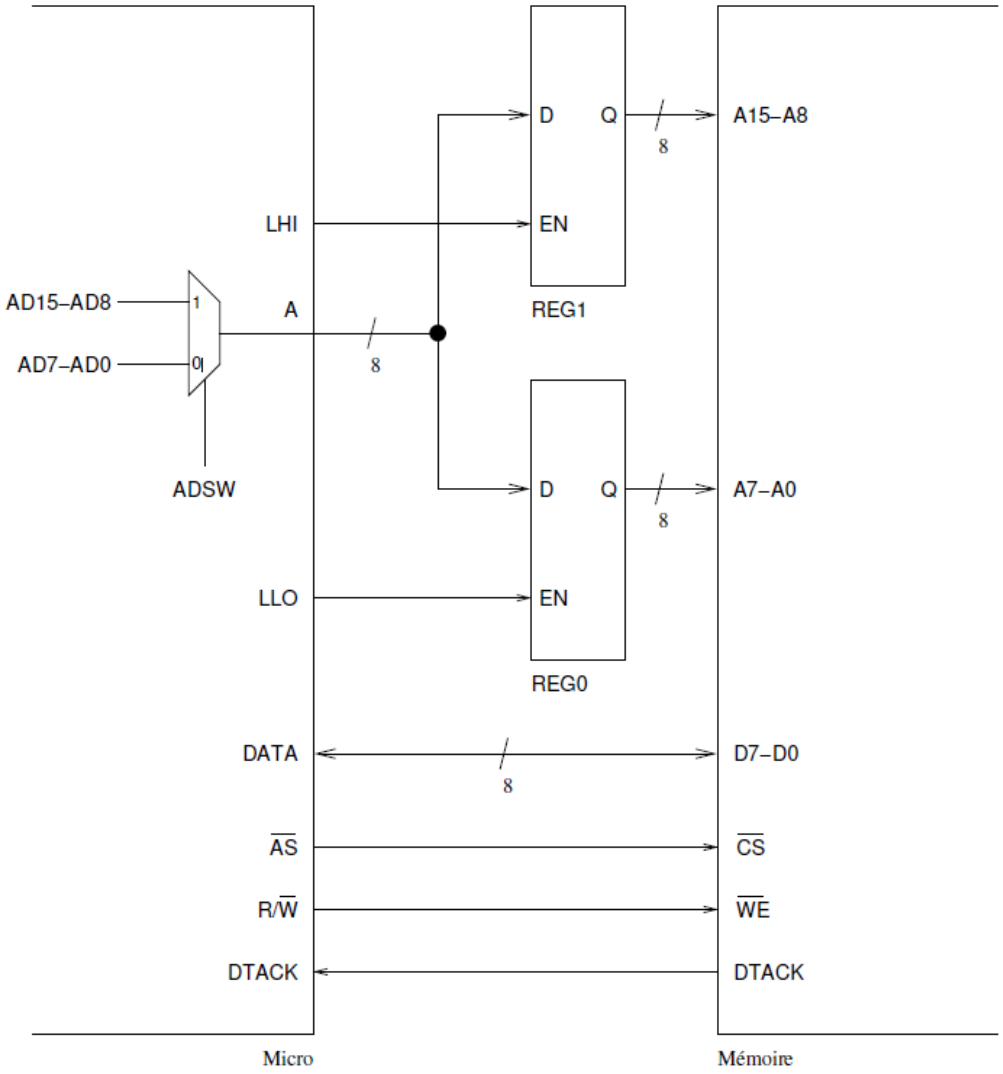
c) Qu'est-ce qui justifie l'utilisation du mécanisme d'interruption ou d'interrogation lorsque le microcontrôleur doit recevoir des données par un port externe? Lequel choisir et pourquoi?

Solution :

Dans l'interrogation, le microcontrôleur décide de son propre chef d'interroger un port externe pour connaître son état. Ceci peut être une approche naturelle si le moment où l'information externe est nécessaire coïncide avec l'état du programme ou si l'interrogation n'a pas besoin d'être fréquente. L'inconvénient principal est la perte de temps par le microcontrôleur passée à interroger à répétition un périphérique. Le mécanisme d'interruption permet d'interrompre le flot du programme du microcontrôleur pour répondre immédiatement à un évènement externe. Ceci est pratiquement la seule alternative si le temps de réponse à un évènement doit être court. On peut aussi déclencher des acquisitions à intervalles réguliers en déclenchant des interruptions à partir d'une horloge ou d'un compteur. Ceci est beaucoup moins lourd que de tenter de créer des délais en logiciel pour faire de l'interrogation.

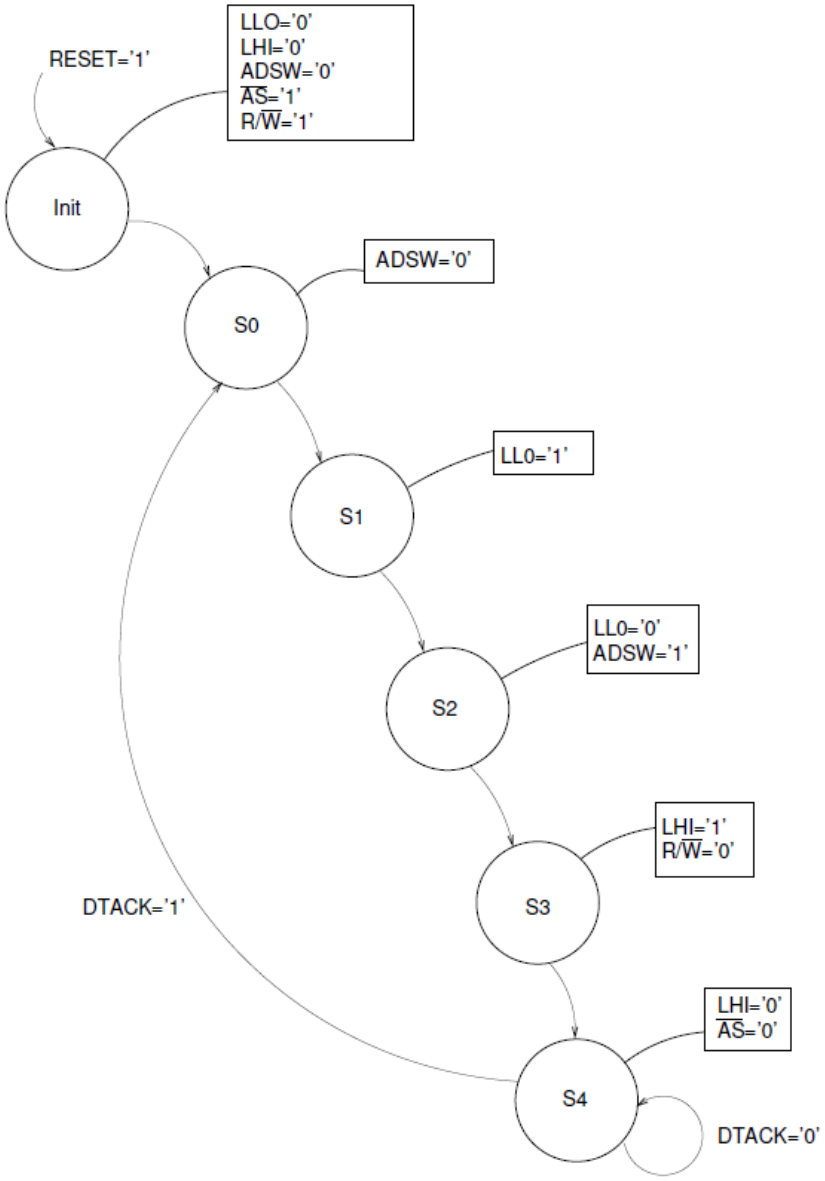
Exercice 6 Séquence d'évènements dans les interfaces

Vous disposez d'un microcontrôleur minimaliste n'ayant que 8 lignes d'adresse que vous souhaitez interfacer avec une mémoire de 64K. Un circuit d'interface (illustré ci-dessous) a été conçu à cette fin incluant 2 registres de 8 bits (REG0 et REG1), ce qui permet de capturer l'adresse complète (16 bits) en deux temps et de déclencher l'opération de lecture ou d'écriture par la suite. On a une adresse complète AD15-AD0 à l'intérieur du microcontrôleur et un multiplexeur interne permet de sélectionner la moitié haute ou basse de celle-ci vers les lignes d'adresses externes A. Les lignes LH1 et LH0 sont raccordées aux entrées "enable" des registres REG1 et REG0, respectivement, et permettent d'activer la capture des données présente à leur entrée D. Le contenu d'un registre est toujours présent à sa sortie Q. Une fois l'adresse complète présente sur les lignes d'adresse de la puce mémoire, la ligne AS (Adress Strobe) passe à '0' et active ainsi l'entrée CS (Chip Select) de la puce mémoire. La puce mémoire va alors utiliser l'adresse présentée et les données (s'il s'agit d'une écriture) pour effectuer l'opération souhaitée (selon la valeur de la ligne R=W qui vaut '1' pour une lecture et '0' pour une écriture). Le microcontrôleur doit ensuite attendre que la ligne DTACK monte à '1' pour signifier que l'accès mémoire est complété.



a) Complétez le graphe ci-dessous de la machine à états qui génère les signaux de commande et qui effectue en continu des écritures en mémoire. Vous n'avez pas à gérer le bus de données.

Solution :



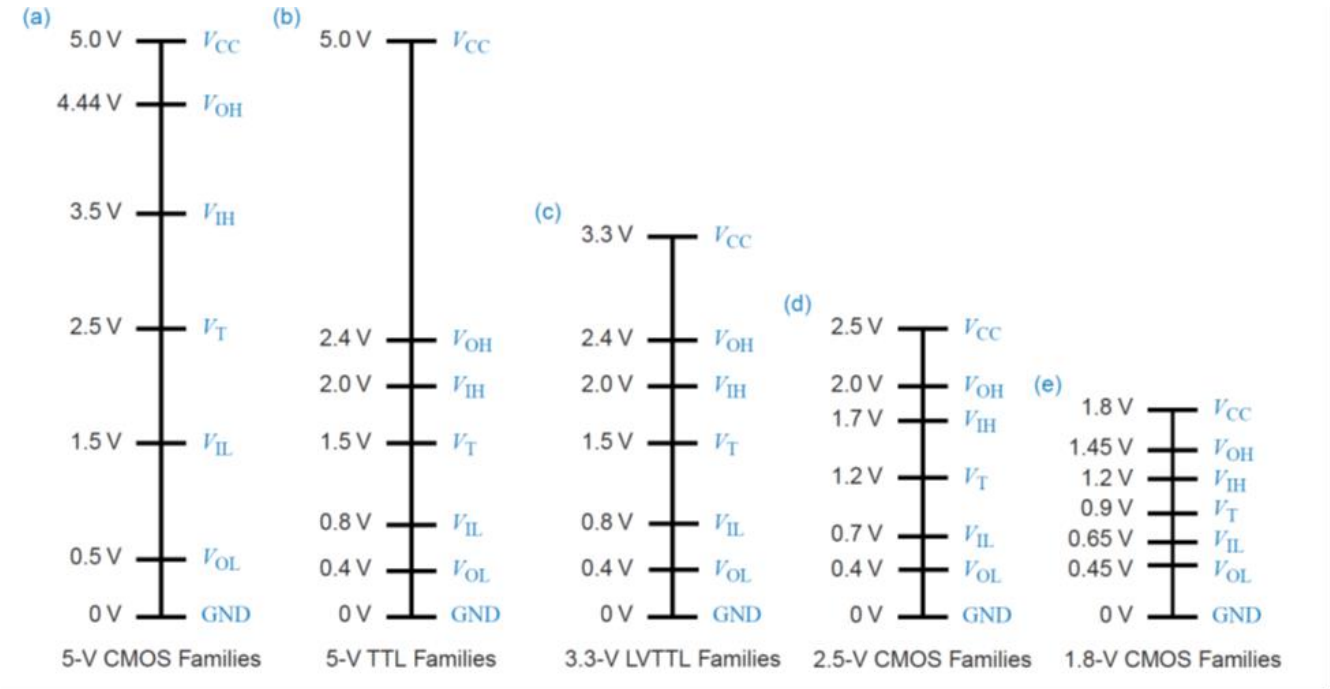
b) Est-il vraiment nécessaire d'avoir deux registres pour la gestion de l'adresse? Expliquez.

Solution :

Non. On pourrait utiliser un registre pour capturer les lignes hautes de l'adresse et garder les lignes basses sur le bus d'adresses du microcontrôleur, lesquelles seraient directement raccordées aux lignes basses de la puce mémoire. Ceci permettrait non seulement d'éliminer un registre, mais également de simplifier la machine à états et de réduire le nombre de coups d'horloge dans le cycle d'écriture.

Exercice 7 Compatibilité électrique

Si un micro-processeur doit lire les données d'un capteur de température utilisant un protocole de communication I2C dont les niveaux de tensions sont les suivants 0,2 V et 3,4 V. Selon la figure ci-dessous, quelles familles sont compatibles avec ce capteurs?



Solution :

5-V TTL seulement.

Exercice 8 Code en C

Expliquer ce que font le lignes suivantes en C, vous avez doit à la librairie fournie pour l'APP.

Solution :

a) unsigned char val1 = PMODS_GetValue(0,1);

Lit la broche JA1 du PMODA et écrit sa valeur dans la variable 'val1'.

b) unsigned int val2 = ADC_AnalogRead(2);

Lit la valeur de l'ADC numéro 2 et écrit sa valeur dans la variable 'val2'.

c) LCD_WriteStringAtPos(msg, 0, 0);

Écrit la string sur le LCD à la position ligne 0, case 0.

Exercice 9 Configuration des ports

Donner les registres pour configurer et utiliser les ports.

Solution :

TRIS : Configure la broche en entrée (1) ou sortie (0).

ANSEL : Configure la broche en analogique (1) ou numérique (0).

LAT : Écrire le port.

PORT : Lire le port.