NOM : Khelaufi

Prénom : Samy

Année 2013/2014

18

A remplir obligatoirement par l'enseignant responsable du contrôle

Date : 10 janvier 2014
Contrôle de : Architecture des systèmes à microprocesseur
Durée : 2 heures
Professeur responsable : N.ABOUCHI
Documents : ☐ autorisés ☑ non autorisés
Si oui : type(s) de documents autorisés :

Calculatrices alphanumériques : ☐ autorisées ☑ non autorisées

REPONDRE SUR LE SUJET : ☑ OUI NON

LES TELEPHONES PORTABLES ET AUTRES APPAREILS DE STOCKAGE DE DONNEES NUMERIQUES NE SONT PAS AUTORISES

A l'attention des élèves : rappels importants sur la discipline des examens

La présence à tous les examens est strictement obligatoire ; tout élève présent à une épreuve doit rendre une copie, même blanche, portant son nom, son prénom et la nature de l'épreuve.
Toute absence non justifiée est sanctionnée par un zéro.
Toute fraude ou tentative de fraude avérée est sanctionnée par un zéro à l'épreuve et portée à la connaissance de la direction des études qui pourra réunir le Conseil de Discipline. Les sanctions prises peuvent aller jusqu'à l'exclusion définitive du (des) élève(s) mis en cause.
Toute suspicion sur la régularité et le caractère équitable d'une épreuve est signalée à la direction des études qui pourra décider l'annulation de l'épreuve; tous les élèves concernés par l'épreuve sont alors convoqués à une épreuve de remplacement à une date fixée par le responsable d'année.

Partie 1 : Généralités sur les microprocesseurs : une machine fictive (5 pts)

La machine sur laquelle porte l'exercice comporte les éléments suivants :

- ✓ une mémoire principale de 32 octets,
- ✓ un registre de travail de 8 bits (nommé **ACCU**) **constituant une des deux entrées de l'UAL avant exécution de l'opération et le résultat de l'opération ensuite,**
- ✓ l'Unité Arithmétique et Logique (**UAL**) est capable d'exécuter seulement les opérations d'addition et de soustraction,
- ✓ un registre de 2 bits (nommé **FLAGS**) qui sont positionnés en fonction du résultat de l'UAL (Bit0=1 si le résultat est nul, Bit1=1 si le résultat est négatif),
- ✓ un registre compteur d'instruction (**IP**) qui contient l'adresse de la prochaine instruction à exécuter,
- ✓ un registre instruction (**INST**) qui contient le code de l'instruction courante.

1. **Définition du langage binaire (1 pt) :**

- Combien de bits faut-il pour représenter une adresse ?

Mémoire principale : 32 octets donc il faut au moins 5 bits pour représenter une adresse

- En supposant que le nombre d'instructions disponibles est de 8 et que chaque instruction est représentée sous la forme **[Codop Adresse]**, combien de bits faut-il pour coder une instruction ? **(Codop = Code opératoire)**

8 instructions = 2^3 ⇒ codé sur 3 bits

[Codop 3 bits + Adresse 5 bits] ⇒ il faut 8 bits pour coder une instruction

2. **Définition du langage symbolique (2 pts) :**

- La liste des instructions est la suivante : addition, soustraction, chargement du registre ACCU (depuis la mémoire), rangement du registre ACCU (dans la mémoire), branchement, branchement si nul, branchement si négatif, fin. Donner un code binaire (Codop) et un nom symbolique (Mnémonique) à chacune d'entre elles.

0h ADD
1h SUB
2h LOAD
3h SAVE
4h JMP
5h JZ
6h JN
7h END

- Ecrire un programme (en langage symbolique) permettant d'ajouter deux nombres stockés aux adresses 01H et 02H de la mémoire et rangeant le résultat à l'adresse 03H de la mémoire.

```
LOAD   01 h
ADD    02 h
SAVE   03 h
```

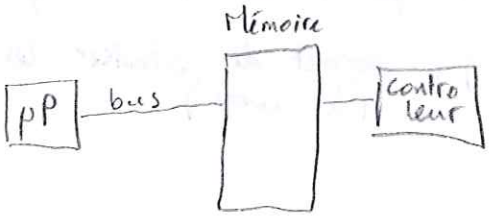
3. **Communication (2 pts) :**

On souhaite faire communiquer notre machine à travers une liaison série asynchrone. Pour cela, on utilise un contrôleur de périphérique fictif « nommé **SERIE** ». Ce dernier, vu comme une position mémoire (**exemple : la position 06H**), ne nécessite pas de programmation, dispose d'un registre d'émission (nommé **EMI**) et d'un registre d'état (nommé **ETAT**). Tous les bits du registre d'état sont à zéro sauf le Bit 0, ce dernier est positionné en fonction de l'état du périphérique (Bit0=1 si le contrôleur de périphérique et prêt à émettre, Bit0=0 si le contrôleur de périphérique n'est pas prêt à émettre).

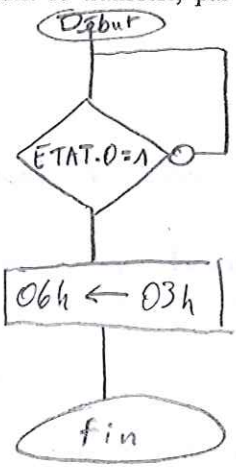
- A quoi sert le contrôleur de périphériques ?

Le controleur de périphériques permet de connaitre l'état du périphérique (ici, savoir si le périphérique est prêt à emmettre ou pas)

- Donner un schéma simple situant le contrôleur de périphérique (et son périphérique) par rapport à la mémoire et au processeur.



- Donner l'algorithme réalisant le transfert, par le lien série, du caractère ASCII stockées dans la position mémoire 03H.



- Ecrire le programme correspondant.

```
TEST :   JZ    ETAT , TEST
         LOAD  03 h
         SAVE  06 h
         END
```

Partie 2 : Mise en œuvre et programmation du 8051F020 : (10 pts)

[P1 : 5,5 / P1.. 4] 9,5

- Citez les différents espaces mémoire du microcontrôleur 8051F020. Expliquez leur rôle. (1pt)

RAM interne : permet de stocker des données + contient les banques de registre et les registres SFR

RAM externe (XDATA) : permet de stocker plus de données

Mémoire de code (CODE) : permet de stocker les instructions (le code)

1

- Le 8051F020 est un processeur de type Harvard, quels sont les deux registres utilisés pour accéder aux données et au code ? (0,5pt)

Pointeur de code

Pointeur de données

1

- Expliquer en donnant des exemples quand est-ce que ces registres sont utilisés ? (0,5pt)

Le pointeur de code pointe vers l'instruction suivante

Le pointeur de données permet de se positionner à une adresse en RAM pour en lire ou écrire du contenu

0,5

- Quel mode d'adressage est utilisé pour l'opérande en gras, et que fait chacune des instructions suivantes : (2pts)

2

MOV A, **#42H** immédiat.
On stocke la valeur 42h dans l'accumulateur

MOVX **@DPTR**, A indirect
On stocke le contenu de l'accumulateur dans la case mémoire de la XDATA qui possède l'adresse que DPTR contient

MOV **00**, 23H direct
On stocke le contenu de la case 23h dans la case d'adresse 00

MOV C,**22H** direct
On stocke le bit 22h dans le registre C (bit de carry)

MOVC A,**@A+DPTR** indirect
On stocke le contenu de la case mémoire située en mémoire de code à l'adresse formée par la somme du contenu de A et de DPTR dans l'accumulateur

Analyse de code Assembleur (2 pts)

- Que fait le programme suivant ? (2 pts)

```
1000H                MOV A, #0D5H
1002H                ADD A, #0AAH
1004H                PUSH ACC
                     ...
1010H                LCALL S_PROG
1013H                DEC A
1014H                JMP FIN
                     ...
2000H S_PROG:        POP ACC
2001H                INC A
2002H                PUSH ACC
2003H                RET
                     ...
2004H        FIN:    JMP FIN
```

- On stocke la valeur D5h dans l'acc
- On fait Acc = Acc + AAh (l'Acc contient alors 5Fh et déborde, ce qui place le bit C à 1)
- On sauvegarde Acc dans la pile
- Appel de "S_PROG" (sous programme)
- On vide la pile dans l'acc (Acc contiendra alors 5Fh)
- Incrémente Acc (Acc contient 60h)
- Sauvegarde de l'Acc dans la pile
- Fin sous prog et retour au "Main"
- Décrémente Acc (Acc contient 5Fh)
- On boucle indéfiniment (pour occuper le µP car fin du programme)

0,1

- Quelles sont ses incidences sur le registre d'état et sur la pile ?

Registre état : le bit C passe à 1

Pile : la pile est chargée 2 fois mais vidée une fois donc elle contiendra 60h

0,1

- Le programme présente-t-il un problème quelconque ?

En soit le programme tourne, le seul "problème" que je vois est le débordement de l'Accumulateur lors de l'addition de D5h et AAh

- Quelle est la valeur de l'accumulateur (registre A) à la fin de l'exécution du programme ?

5Fh

Codage en C et configuration de périphérique (4 pts).

On souhaite mettre en œuvre le timer3 du 8051F020 pour générer une interruption toutes les 20 ms.

Coder les 2 fonctions suivantes :

- **Init_Timer3** : cette fonction permet de configurer le timer3 pour générer une interruption toutes les 20 ms avec Sysclock = 22,1184 Mhz.
- **ISR_Timer3** : c'est la fonction d'interruption du Timer 3. Le code « application » dans cette routine se limite à une seule action : faire allumer une LED pendant 1s et la faire éteindre durant 10s, et ce de manière infinie.

Il peut vous manquer des informations, à vous de préciser quelles sont ces informations manquantes. Les diverses informations utiles à cet exercice vous sont fournies en annexe.
N'oubliez pas de commenter votre code.

Sysclock = 22,1184 MHz → 22,1184 µs
Sysclock/12 → ~ 2 µs

```
Init_Timer3 () {
    TMR3RL = -10 000;   // on précharge le timer à la valeur -10 000 pour avoir 20ms environ entre chaque IT
                        On aura auparavant déclaré TMR3RL en sfr16 dans les initialisation du programme
    TMR3 = -10 000;     // on précharge le timer pour le premier coup, ensuite il se rechargera tout seul.
                        Idem, ne pas oublier de déclarer sfr16 TMR3
    EIE2 |= 0x01;       // <=> ET3 = 1, on autorise l'IT ET3 du Timer3
    Il faudrai faire EA = 1; (autoriser les IT, il me manque le registre nécessaire dans la datasheet)
    TMR3CN &= 0x83h;    // T3XCLK = 0 pour utiliser SYSCLOCK
    TMR3CN |= 0x04h;    // T3M = 0 pour utiliser SYSCLOCK/12
                        // TR3 = 1 pour démarrer Timer3
                        // TF3 = 0 pour lui faire savoir qu'il n'a pas déborder
}
```

```
ISR_Timer3 () interrupt 14 {
    TMR3CN &= 0x7F;     // TF3 = 0, remise à zéro du flag de débordement
    i++;
    if (i == 50)
        LED = 0;
    if (i == 550)
    {
        LED = 1;
        i = 0;
    }
}
```

Comme le timer est sur 16 bits, on ne peut pas compter sur 1s ou 10s
On utilise un indice i tel que :
- i est incrémenté à chaque IT
- si i = 50 ⇒ 50 × 20 ms = 1 seconde alors il se sera écoulé 1 seconde depuis que la LED est allumée, donc on l'éteint
- si i = 550, il se sera écoulé (550-50) ⇒ 500 × 20 ms = 10 s depuis que la LED est éteinte, donc on l'allume

Ne pas oublier de déclarer LED
Ne pas oublier, dans le main, d'allumer la LED pour le "premier" tour (il n'est pas indispensable, mais ça permet de commencer par 1s allumé puis 10 sec éteint au lieu d'attendre que tout se mette en place)
Ne pas oublier de faire boucler le main dans un while(1) après

Partie 3 : notions avancées d'architectures : (5 pts) 4

Question : carte mère (1/2 pt) : Qu'est-ce qu'un chipset et quel est son rôle dans la carte mère d'un PC ?

Question 2 : mémoire vive (1/2 pt) : quels types de mémoires vives sont utilisés dans un PC et pourquoi ?

RAM DDR 2 / DDR3 car c'est une mémoire avec un petit temps d'accès => rapide

0,5

Question 3 : carte graphique (1/2 pt) : quel est le rôle de la carte graphique dans le PC, est-elle toujours indispensable ?

La carte graphique permet de faire des calculs et des affichages, et non elle n'est pas indispensable (certaines cartes mères possèdent un chipset graphique) 0,5

Question 4 : disque dur (1/2 pt) : quel est le temps d'accès d'un disque dur interne (ordre de grandeur), comment le situer par rapport au temps d'accès de la mémoire cache et de la mémoire principale ?

temps accès Disque dur > temps accès Mémoire Vive > temps accès Mémoire Cache

0,5

Question 5 : port SATA (1/2 pt) : rappeler les principaux avantages du port SATA. Comment est utilisé le port SATA dans le PC ?

SATA : plus rapide, prend peu de place => favorise la circulation de l'air (refroidissement)

Utilisé pour connecter le disque dur, lecteur CD, DVD... à la carte mère

0,5

Question 6 : port USB (1/2 pt) : les ports USB supportent la connexion à chaud et la reconnaissance automatique des dispositifs (Plug and Play), expliquer le principe de fonctionnement de ce dispositif.

Question 7 : clé USB (1/2 pt) : rappeler le principe de fonctionnement des clés USB (type de mémoire utilisées, mode d'accès, etc.) ? Mémoire Flash, mode d'accès aléatoire donc accéder à n'importe quel secteur prend autant de temps

OK

Question 8 : clavier (1/2 pt) : décrivez le fonctionnement d'un clavier matriciel ?

- Scrutation des colonnes pour vérifier si une touche est pressée
- Scrutation des lignes pour identifier quelle touche est pressé

OK

Question 9 : cours architectures évoluées (1 pt) : à quoi sert le cache pour un processeur ? Pourquoi l'utilisation du cache n'est pas adaptée au temps réel sévère comme le traitement du signal ?

Cache : permet d'avoir les données proches du µP = rapidité d'accès

Un cache n'est pas adapté au temps réel car ce sont des algorithmes qui supposent (par probabilité) quelles sont les données à charger à l'avance dans le cache pour faire gagner du temps au µP. Mais les algo ne peuvent pas prédire le futur et si une donnée qui était nécessaire n'est pas chargée, il faudra aller la chercher en mémoire
⇒ on ne maitrise pas le temps (car on ne sait pas quand auront lieu des défauts de cache)