Microprocesseurs 1 & 2: Travail écrit no 3. 3.5

Nom : ZOPPI **Prénom :** DAMIANO **Classe : T/2** **Date : 30.04.2012**

Problème n° 1 (interfaçage C - assembleur)

a) Codez en assembleur la fonction « f » ci-dessous.

```
typedef int (*oper_t) (int a, int b, int c, int d, int e, int f) ;
                       r0    r1    r2    r3    r4    r5

int f (oper_t oper) { return oper (10,20,30,40,50,60) + 100; }
                                   reg   stack
```

```
sub  sp , #4*2
ldr  r0, = e
ldr  r0, [r0]
str  r0, [sp, #0]
ldr  r0, = f
ldr  r0, [r0]
str  r0, [sp, #4]
bl   f
add  sp, #4*2
f:  mov  r0, #10
    mov  r1, #20
    mov  r2, #30
    mov  r3, #40
    add  r0, r1
    add  r0, r2
    add  r0, r3
```

```
stmfd sp!, [r4, lr]
mov   r4, r0
sub   sp, #4*2
mov   r0, #60
str   r0, [sp, #4]
mov   r0, #50
str   r0, [sp, #0]
mov   r0, #10
mov   r1, #20
mov   r2, #30
mov   r3, #40
blx   r4
add   sp, #4*2
add   r0, #100
ldmfd sp!, [r4, pc]
```

```
ldr  r1, [sp, #0]
add  r0, r1
ldr  r1, [sp, #4]
add  r0, r1
add  r0, #100
mov  pc, lr
```

ldmfd sp!, {r4-r5, lr}

b) Citez les techniques et méthodes utilisées pour le passage d'arguments/paramètres à des fonctions.

- par valeur
- par référence + registre / pile

c) Citez 2 conditions pour qu'une fonction (sous-routine) soit réentrante

- ne doit pas avoir des variables statiques non-constantes
- ne doit pas faire des appels de fonctions non réentrantes
- ne doit pas modifier son propre code

Microprocesseurs 1 & 2: Travail écrit no 3.

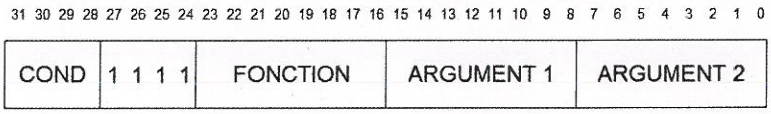
6

Problème n° 2 (Interruptions)

Le jeu d'instructions du processeur i.MX27 codé sur 32 bits contient l'instruction SWI. L'exécution de cette instruction dans le mode utilisateur génère l'exception « swi » et fait basculer le programme dans le mode superviseur.

Un programmeur décide d'utiliser cette instruction pour créer un nouveau jeu d'opérations pour le noyau. Ces nouvelles opérations ont la signature suivante « *int (*fnct) (int a1, int a2)* » et seront toutes contenues dans un tableau appelé « *fnct* ». Le résultat de l'opération sera retourné à l'application par le registre « *R0* ».

Comme indiqué sur la figure ci-dessous, les bits 0 à 7 sont utilisés pour le 2ème argument, les bits 8 à 15 pour le 1er argument et les bits 16 à 23 pour sélectionner la fonction du noyau.



a) Implémentez en assembleur la routine d'interruption permettant de traiter les 256 opérations contenues dans le champ « fonction » et appelez la routine correspondante contenue dans le tableau « *fnct* ». La valeur du champ fonction sert d'index dans le tableau.
Pour rappel, le processeur sauve l'adresse de retour (sans offset) lors de l'exécution de l'instruction swi.

```
swi_handler:
    stmfd   sp!, [r1-r12, lr]      @ save context
    ldr     r0, [lr, #-4]          @ load swi instruct
    and     r2, r0, #0x00ff0000    @ compute operation
    ldr     r1, =fcnt              /16-2
    ldr     r2, [r1, r2, lsr #14]
    and     r1, r0, #0x000000ff    @ compute 2 argument
    and     r0, r0, #0x0000ff00    @  "  1 argument
    mov     r0, r0, lsr #8
    blx     r2
    ldmfd   sp!, [r1-r12, pc]^
```

b) Citez les étapes/la marche à suivre pour installer une routine d'interruption sur le microprocesseur

2. créer un espace mémoire pour contenir la pile pour le mode choisi
b. changer dans le mode choisi
c. initialiser le registre
:

Microprocesseurs 1 & 2: Travail écrit no 3.

7

Problème n° 3 (Interruptions)

a) Décrivez succinctement 2 grands principes (implémentation hardware) pour déterminer l'origine d'une source d'interruption (vous pouvez utiliser un schéma) et citez le avantages et inconvénients de ces solutions.

par scrutation

le µP contrôle continuement l'état des périphériques, si il y a une donnée à émettre et le buffer d'emission n'est pas plein, alors il doit l'émettre. Sinon, s'il y a une donnée à lire, il doit la lire

1

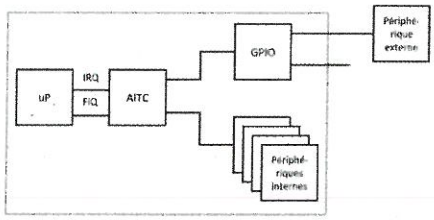
par interruption

le µP est utilisé que lors du changement d'état des périphériques

pas la question

- scrutation
- priorité d'interruption
- vectorisé

La figure ci-dessous représente le système d'interruption de l'i.MX27 du point de vue HW.



b) Indiquez la technique/méthode utilisée pour déterminer l'origine de l'interruption (vecteur) par chaque composant du système d'interruption de l'i.MX27 ci-dessus

µP : vecteur d'interruption — vectorisé / priorité d'interruption

2 AITC : vecteur d'interruption " / "

GPIO : par scrutation

périphériques : dépend de l'implémentation

c) Indiquez par quel moyen il est possible d'activer et de désactiver les interruptions au niveau du processeur ARM (le core) (CPSR)

2 Dans le registre de statut il y a 2 bits "F" et "I" qui sont utilisés justement pour activer / desactiver les FIQ et les IRQ.

F==1 ou I ==1 → Interruptions désactivés

d) Décrivez la commutation de contexte lors d'une interruption

2 Lorsqu'une interruption survient, les actions d'enregistrer l'adresse de retour et le PSR, de modifier le mode de fonctionnement du µP, de chercher la routine d'interruption adéquate et d'enregistrer les registres ou de restaurer ces valeurs lorsque l'interruption a été traité, composent la commutation de contexte.

Microprocesseurs 1 & 2: Travail écrit no 3.

9

Problème n° 4 (Entrées/Sorties)

a) Décrivez lu traitement des entrées/sorties par interruptions. Utilisez un schéma de principe pour soutenir la description.

lorsqu'une information arrive ou
lorsqu'on est prêt d'émettre une information et le buffer d'emission
n'est pas plein, la périphérique génère une interruption qui est transférée
au GPIO que à son tour le transmet à l'AITC que l'envoie au
µP à travers l'un des cycles de connexion (FIQ ou IRQ). Selon
le type d'interruption, le µP va chercher dans la table des vecteurs
d'interruption la routine pouvant la traiter et la démarre. Celle-ci se chargera
enfin de regarder dans le GPIO quelle périphérique a généré l'interruption.

4

b) Citez les avantages, désavantages et les contraintes reliés à ce mode traitement

⊕ µP n'est utilisé que lors d'une interruption (pas de busy waiting où le µP est continuement occupé, il peut effectuer d'autres tâches)
⊕ temps de réaction
⊖ Temps de traitement
⊖ Pas de gestion des surcharges
⊖ code plus complexe

Contraintes: µP nécessite une logique d'interruptions

3

c) Implémentez la routine « void putchar(char c) » permettant d'écrire un caractère sur une interface série au travers du contrôleur ci-dessous. L'écriture se fera par scrutation.

```
#define STATUS_TR 1<<1  /* transmitter ready: character could be sent  */
#define STATUS_RR 1<<0  /* receiver ready: character has been received */
struct serial_ctrl_t {
   uint32_t rxbuf ;     /* registre contenant les données reçues  */
   uint32_t txbuf;      /* registre pour l'émission des données   */
   uint32_t stat;       /* registre de statut voir bits ci-dessus */
   uint32_t ctrl;       /* registre de contrôle du périphérique   */
}* serial = (struct serial_ctrl_t)0x10019000;
```

```
void putchar(char c){
  while(1){
    if((serial->txbuf) & (STATUS_RR) != 0)
      serial->txbuf = c;
  }
}
```

1.5

Microprocesseurs 1 & 2: Travail écrit no 3.

6

Problème n° 5 (Systèmes temps-réel)

a) Hormis les noyaux temps réel, citez 2 autres techniques pour traiter plusieurs tâches en parallèle.

SCL - Simple control loop
ICS - Interrupt controlled systems

0

b) Citez les composantes principales d'un noyau temps réel

- Processus -
- ordonnanceur
- mécanisme de synchronisation -
- mécanisme de gestion des interruptions
- sortie
- compteur périodique

2

c) Citez les états d'un processus/thread

2 READY, RUNNING, BLOCKED, TERMINATED

d) Implémentez en assembleur la fonction de transfert de contexte entre deux threads

```
/**
*@param f_tcb reference to the former thread control block
*@param psr new psr value of the former thread
*@param n_tcb reference to the new thread control block
*/
extern void transfer(struct tcb* f_tcb, unsigned long psr, struct tcb* n_tcb);
```

transfer:

```
stmia r0, {r0-r12, sp, lr}
str   r1, [r0, #15*4]
ldr   r1, [r2, #15*4]
msr   spsr, r1
ldmia r2, {r0-r12, sp, pc}^
```

svc_handler:

```
sub   sp, #4*15
stm   sp, {r0-r12}
mrs   r0, spsr
str   r0, [sp, #15*4]
stmfd sp!, {r0-r12, sp, lr}
ldmfd sp!, {r0-r12, sp, pc}^
ldr   r0, [sp, #15*4]
msr   spsr, r0
add   sp, #4*15
```

1