



Systèmes Embarqués 1 & 2: Travail écrit no 3.

Nom : Grossrieder

Prénom : Nadine

Classe : T-2/I-2

Date : 20.04.2015

Problème n° 1 (interfaçage C - assembleur)

a) Codez en assembleur la fonction « bar » ci-dessous. Note : les « long » ont 32 bits.

```
typedef long (*foo_t) (long a1, long a2);
long bar (long p1, foo_t foo, long p2, long p3, long p4, long p5)
{return p3 + p2 + foo (p5, p1) + p4;}
```

bar: stmfd sp!, {r3}

add r2, r3

add r2, [sp, #4]

mov r3, r4

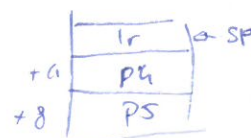
mov r4, r0

mov r0, [sp, #8]

blx r3

add r0, r2

ldmfd sp!, {PC}



bien.

Ok, mais l'ordre  
n'est pas respecté.

r2 n'est pas défini

pourquoi?

seulement quand appel à une interruption

b) Le graphique ci-dessous représente l'état du processeur (registres et pile sur 32 bits) à l'entrée de la fonction « f2 » (aucune instruction de « f2 » n'a encore été exécutée).

```
int f2 (int a1, int a2, struct S* a3) {return a2 + f3(a1, a3);}
```

Indiquez la valeur des paramètres a1 à a3.

low address	100
	414
	153
SP (à l'entrée de f2) →	0
	567
	898
	910
high address	1114

R0	10
R1	1020
R2	0
R3	45
R4	18
R5	90
R6	33

a1 : 10  
a2 : 1020  
a3 : 0

c) Implémentez en assembleur le passage par valeur de la variable « int var ; » dans le registre R0

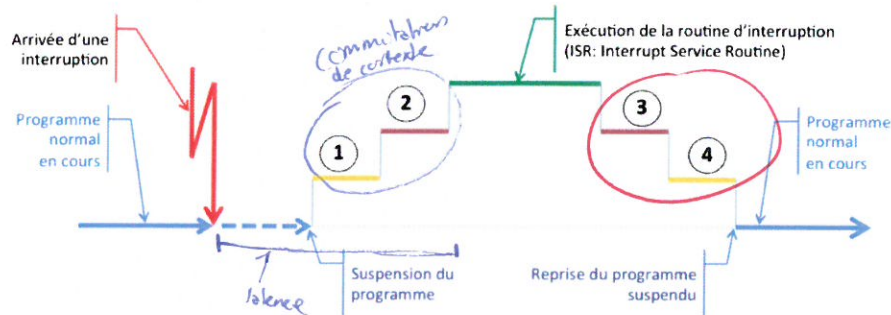
1 ldr r0, =var  
ldr r0, [r0]

8

## Systèmes Embarqués 1 &amp; 2: Travail écrit no 3.

## Problème n° 2 (Interruptions)

La figure ci-dessous représente les différentes transitions que le microprocesseur et son logiciel effectuent lorsque une interruption est soulevée avant de pouvoir exécuter la routine d'interruption (ISR).



- a) Décrivez succinctement les opérations effectuées lors des transitions 1 à 4. Précisez si celles-ci sont effectuées par le microprocesseur (hw) ou par le logiciel (sw).

① effectuée par HW  
sauvegarde de l'état du processeur  
recherche de l'adresse de la routine d'interruption

③ effectuée par SW  
Changement de contexte et restauration des registres

② effectuée par SW  
Changement de contexte et sauvegarde des registres

④ effectuée par HW  
restauration de l'état du processeur

- b) Implémentez les opérations assembleur que le microprocesseur devra exécuter pour sauvegarder et restaurer l'état du microprocesseur lorsqu'une interruption matérielle (IRQ ou FIQ) est levée. Indiquez le numéro de la transition où s'effectuent ces opérations

Numéro de la transition : ① 2

```
sub    lr, #-4  
stmdb  Sp!, {r0-r12, lr}  
ldmfd  Sp!, {r0-r12, PC}
```

- c) Représentez sur le graphique ci-dessus la commutation de contexte d'interruption.

Partie ① et ②

- d) Représentez sur le graphique ci-dessus la latence.

de l'arrivée de l'interruption jusqu'à la fin de la partie ②

- e) Donnez le terme technique de la variation du temps de latence et citez 2 exemples qui peuvent la faire varier fortement.

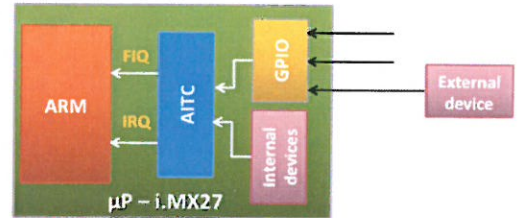
Gigue d'interruption ou jitter interrupt

- ① le temps d'exécution de l'instruction la plus longue  
② le temps où les interruptions sont désactivées

Systèmes Embarqués 1 & 2: Travail écrit no 3.

Problème n° 3 (Interruptions)

La figure ci-contre représente le système d'interruption du microprocesseur i.MX27.



- a) Décrivez succinctement le rôle des composants ci-dessous et indiquez la méthode utilisée par le composant pour identifier la source d'interruption.

- ARM : *priorité d'interruption* -  
*Active l'interruption et cherche l'adresse de la routine d'interruption*
- AITC : *Interruption vectorisée* ✓  
*Reçoit une interruption soit du GPIO soit des périphériques internes*
- GPIO : *scrutation logicielle* -  
*reçoit l'interruption d'un périphérique externe et détermine de quel périphérique il provient*

- b) Implémentez en assembleur la fonction « void init\_sp (int mode, void\* sp); » permettant d'initialiser les pointeurs de piles pour les différents modes du μP (registre SP).

*init\_sp : stmfd sp!, {r0-r12, r13}*  
*mrs r8, SPS-C* *mou r4, r3*  
*bic r3, #3F*  
*add r3, r0*  
*mstr SPS-C, r3*  
*mou sp, r4*  
*mstr SPS-C, r4*  
*ldmfd sp!, {r0-r12, r13}* ✓

*IF*  
*M*  
*00 010010*  
*M*  
*01110*  
*r3*  
*00 000000*  
*00 01100*

- c) Indiquez le type de pile utilisée et comment celle-ci fonctionne.

*pile full descending* ✓

*On commence à stocker les données à l'adresse la plus grande et on remonte vers les adresses plus petites*

- d) Sachant que « var » est une variable de type « int », décrivez pour quelle raison l'opération « var++; » n'est pas atomique et indiquez les instructions à effectuer pour rendre cette opération atomique.

*On ne sait pas quand l'incrément va être fait (car il peut y avoir une interruption entre deux)*  
*il faut désactiver les interruptions avant, faire l'opération var++ puis réactiver les interruptions*  
*entre deux quoi?*

*car le*

*travail sur read modify write*

*ldr*  
*add*  
*str*

*plusieurs instructions seules pour incrémenter*

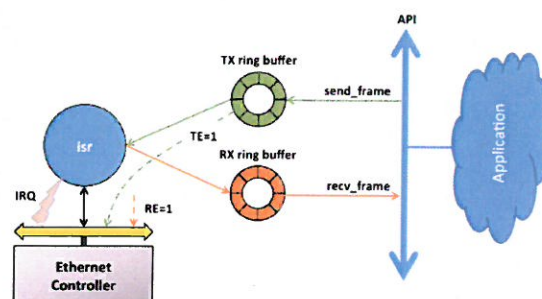
*et une interruption peut intervenir entre deux instructions*



Systèmes Embarqués 1 & 2: Travail écrit no 3.

Problème n° 4 (Entrées/Sorties)

La figure ci-contre représente le schéma de principe pour le traitement par interruption d'un contrôleur pour une interface Ethernet à 100Mbit/s full-duplex.



- a) Décrivez la fonction des tampons de réception et d'émission (RX ring buffer et TX ring buffer).

Les tampons permettent de stocker les informations pour que l'application ou le périphérique ne soit pas obligé de tout le temps scruter pour obtenir ou envoyer des données

- b) Décrivez le rôle de la routine d'interruption (ISR).

La routine d'interruption va recevoir les interruptions du périphérique et va aller stocker ou chercher les informations dans les buffers

- c) Implémentez la routine « void send\_frame (const msg\_t\* frame) ; » permettant à l'application d'émettre des trames.

```
#define CTRL_TE (1<<1) // transmitter interrupt enabled
static volatile struct ethernet_ctrl {
    uint16_t ctrl; // control register
    /* ... other registers... */
} * ether = (struct ethernet_ctrl*)0x2001c000;
struct fifo {int in; int out, msg_t frames[512];} tx_fifo;
```

```
void send_frame (const msg_t* frame) {
```

~~ether->ctrl &= ~CTRL\_TE~~ // désactive les interruptions  
 while (tx\_fifo->in == tx\_fifo->out) { // on vérifie qu'il y a de la place dans le buffer  
 tx\_fifo->in = tx\_fifo->frames[tx\_fifo->in] = \*frame  
 tx\_fifo->in++  
 }  
 ether->ctrl |= CTRL\_TE // active les interruptions

- d) Pour le contrôleur Ethernet ci-dessus et le mode de traitement par interruption, dimensionnez le tampon de réception afin que la latence maximale autorisée côté application soit au minimum de 45 ms.

Afin d'économiser la taille mémoire nécessaire, le tampon de réception sera formé de blocs de 200 bytes chacun. Si la taille du paquet dépasse la taille maximale d'un bloc, celui sera stocké sur plusieurs blocs.

Côté réseau des paquets de 125 bytes et/ou 1250 bytes (framing compris) sont émis en burst de 5 ms à plein débit et suivi ensuite d'une pause de 10 ms.

débit = 100 Mbit/s

1500 buffer de 200 bytes

15ms

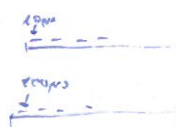
$$x \cdot 200 \text{ bytes} = \text{buffer}$$

$$125 \text{ bytes} = 1000 \text{ bits}$$

$$1250 \text{ bytes} = 10000 \text{ bits}$$

$$\frac{10000}{100 \cdot 10^6} = 10 \mu\text{s}$$

$$\frac{100000}{100 \cdot 10^6} = 100 \mu\text{s}$$



$$\frac{15 \text{ ms}}{10 \mu\text{s}} = 1000 \text{ paquets}$$

en 15ms

$$\frac{15 \text{ ms}}{100 \mu\text{s}} = 100 \text{ paquets}$$

$$15 \text{ ms} = 15000 \mu\text{s} \Rightarrow y = 3$$

$$x \cdot 200 = 375000 \Rightarrow x = 1875 \text{ entrées}$$

$$3000 \cdot 125 = 375000 \text{ bytes}$$

$$3000 \cdot 1250 = 3750000 \text{ bytes}$$

Systèmes Embarqués 1 & 2: Travail écrit no 3.

Problème n° 5 (Systèmes temps-réel)

a) Citez les composants principaux du noyau d'un système d'exploitation.

- ~~Processeur~~
- Ordonnanceur ✓
- Periodic Timer ✓
- Mécanisme de synchronisation ✓
- Mécanisme de gestion des interruptions ✓
- Mécanisme d'échange d'information ✓
- Gestionnaire de mémoire ✓

b) Définissez la structure minimale du TCB (Thread Control Block)

```
struct tcb {
    uint32_t register ✓
    uint32_t lr ✓
    uint32_t sp ✓
    uint32_t pc ✓
    tcb* next ✓
    char* state ✓
}
```

stack

c) Implémentez la fonction de transfert implémentant la commutation de contexte entre deux threads.

void transfer (struct tcb\* former, uint32\_t psr, struct tcb\* new) ;  
transfer :

```
stmia r0, {r0-r12, lr, sp}
mrs r1, [r0, #4*15]
mrs r1, [r2, #4*15]
ldmia r2, {r0-r12, pc, sp}
```

Dans exam oral

d) Définissez la structure minimale d'un sémaphore.

```
struct semaphore {
```

```
    tcb* running_thread
    bool is_use ✓
    tcb* next_thread ✓
}
```

counter

Dans exam oral

e) Implémentez la fonction « void sema\_signal(int id) ; » permettant de libérer le sémaphore.

Quelques éléments de réalisation :

```
struct semaphore sema [200];
```

```
struct tcb* sema_extract_thread (int sema_id); // extrait 1er tcb contenu dans la
liste du sémaphore, retourne 0 si aucun tcb n'est dans la liste
```

```
void sema_signal(int id) {
    if (id < 200 && sema[id] != 0) {
        while (sema_extract_thread(id) != 0) ;
        sema[id] = 0;
    }
}
```

```
sema[id].counter++;
struct tcb* tcb = sema[id].thread[id];
if (tcb != 0) {
    tcb->state = WAITING;
}
```