4,9

Systèmes Embarqués 1 & 2: Travail écrit no 3.

Nom : **Prénom :** **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;}
```

```
stmfd sp!, {r4-r6, lr}
ldr r4, [sp, #16] //r4=p4
ldr r5, [sp, #20] //r5=p5
ldr r6, r3  //r6=p3
add r6, r2  //r6=p3+p2
ldr r3, r0  // r3 = p1
ldr r0, r5  // r0 = p5
ldr r2, r1  // r2 = foo
ldr r1, r3  // r1 = p1
bx r2  // r0 = foo(p5,p1)
add r6, r0  // r6 = p3+p2+foo(p5,p1)
add r6, r4  // r6 = " +p4
ldr r0, r6 // r0 = r6 = return
ldmfd sp!, {r4-r6, pc}
```

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

$a_1 = 10$

$a_2 = 1020$

$a_3 = 0$

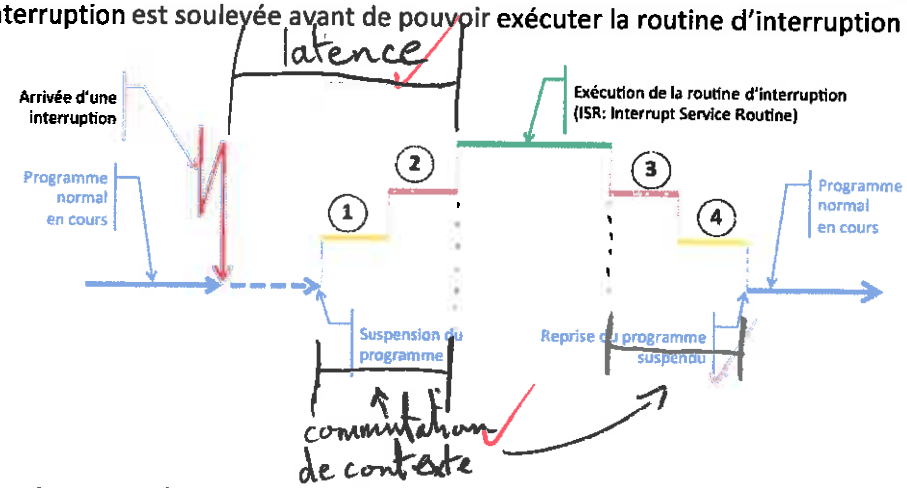
c) Implémentez en assembleur le passage par valeur de la variable « int var ; » dans le registre R0

```
ldr r0, =var
ldr r0, [r0]
```

Systèmes Embarqués 1 & 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 souleyée 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 sauvegarde l'état du µP (hw)
② sauvegarde des registres (sw)
③ restauration des registres (sw)
④ restauration de l'état du µP (hw)

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

② `stmfd sp!, {r0-r12, lr}`

③ `ldmfd sp!, {r0-r12, pc}^`

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

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

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

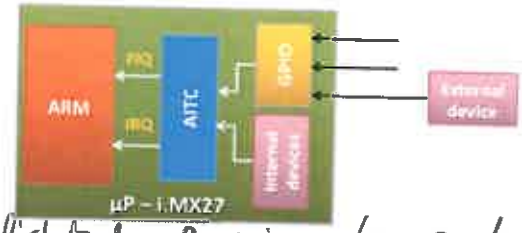
terme technique : gigue
exemples : - la méthode de détection d'interruption (scrutation, ...)
- la taille/quantité de reg à sauver

Systèmes Embarqués 1 & 2: Travail écrit no 3.

4

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 : rôle : relève l'interruption, sauvegarde l'état du µP, puis va lancer la routine correspondant à l'interrupt dans le isr.
  méthode : vectorisée
- AITC : rôle : récupérer l'interruption, en définir la priorité (FIQ > IRQ), l'envoyer au µP.
  méthode : vectorisée
- GPIO : rôle : relever les interruptions de périphériques externes, les transmettre à l'AITC.
  méthode : scrutation

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 :`

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

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.

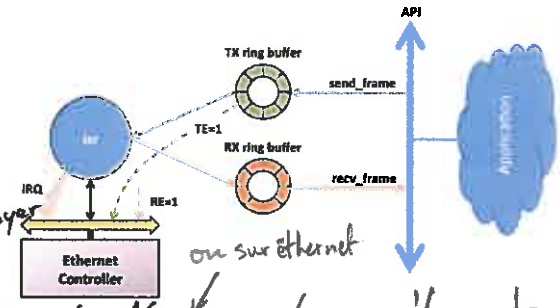
car on a :

```
ldr r0, =var
ldr r1, [r0]
add r1, #1
str r1, [r0]
```

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).

ils stockent les données reçues, et ne demandent à transmettre leurs data à l'application que lorsqu'ils sont remplis.

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

ETHC va générer des exceptions lorsqu'il reçoit des données, afin que RX les récupèrent. la routine isr va transférer ces data-là dans le buffer. Pareillement pour le transfert de données, une exception est levée et isr la gère.

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 ether_ctrl*)0x2001c000;
struct fifo {int in; int out, msg_t frames[512];} tx_fifo;

void send_frame (const msg_t* frame) {
    for (int i=0, i < fram.length ; i++) {
        tx_fifo.msg_t[tx_fifo.out] = frame[i];
        tx_fifo.out = (tx_fifo.out + 1) % 512;
    }
}
```

if (tx_fifo.out == tx_fifo.in) { ethernet_ctrl |= CTRL_TE }

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.

100 Mbit/s = 100'000'000 bit/s
= 100'000 bit/ms = 500'000 bit/5ms

1250 byte = 10'000 bits

nbr max de paquets 10'000 bits : 50

1 paquet 10'000 bits ⇒ 7 blocks

taille pour 15 ms ⇒ 7·50 = 300

taille pour 45 ms ⇒ 1200 blocs

ou 1250 mis à 125 ≙ 1500

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.

- Thread / process
- periodic timer
- heap manager
- gestionnaire transfert de messages
- gestionnaire d'interruptions
- gestionnaire de synchronisation (semaphore)
- scheduler

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

```
struct tcb {
    u32 regs[15]
    u32 psr
    tcb* next
    state
    stack[]
}
```

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

```
void transfer (struct tcb* former, uint32_t psr, struct tcb* new) ;
```

transfer :

```
stmia  r0, {r0-r12, sp, lr}
ldr    r1, [r2, #64]
msr    cpsr_xsfc, r1
ldmia  r2, {r0-r12, sp, pc}
```

d) Définissez la structure minimale d'un sémaphore.

struct semaphore {

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) {
```