Systèmes Embarqués 1 & 2: Travail écrit no 3. 4.6

Nom : Bard **Prénom :** Alexandre **Classe : T/2** **Date : 17.04.2014**

8

Problème n° 1 (interfaçage C - assembleur)

a) Codez en assembleur la fonction « foo » ci-dessous. Note : les « `int` » ont 32 bits.

```
typedef int (*bar_t) (int p1);
int foo (bar_t bar, int a1, int a2, int a3) {return a3 + a1 + bar (a2);}
```

```
foo: push {r4, lr}   r1  r2  r3
     add r4, r1, r3;
     mov r1, r0;
     mov r0, r2;
     blx r1;
     add r0, r4;
     bx lr;
     pop {r4, pc}
```

3,5

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

```
int baz (struct S* a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8)
  {return a2 + a3 + a4 + a5 + a6 + a7 + op2(a1, a8);}
```

Indiquez la valeur des paramètres a1 à a8.

3

low address	52
	44
	13
SP (à l'entrée de baz) →	36
	57
	98
	10
high address	14

R0	0
R1	45
R2	76
R3	87
R4	60
R5	21
R6	65

a1 : 0 a8 : 36
a2 : 45 a7 : 57
a3 : 76 a6 : 98
a4 : 87 ✓ a5 : 10 ✓

c) Citez les 2 techniques et les 2 mécanismes utilisées pour le passage d'arguments/paramètres à des fonctions.

Passage par valeur : la valeur de la variable passée en paramètre est copiée.

Passage par référence : c'est l'adresse de la case mémoire où est stockée la valeur qui est passé en paramètre.

1

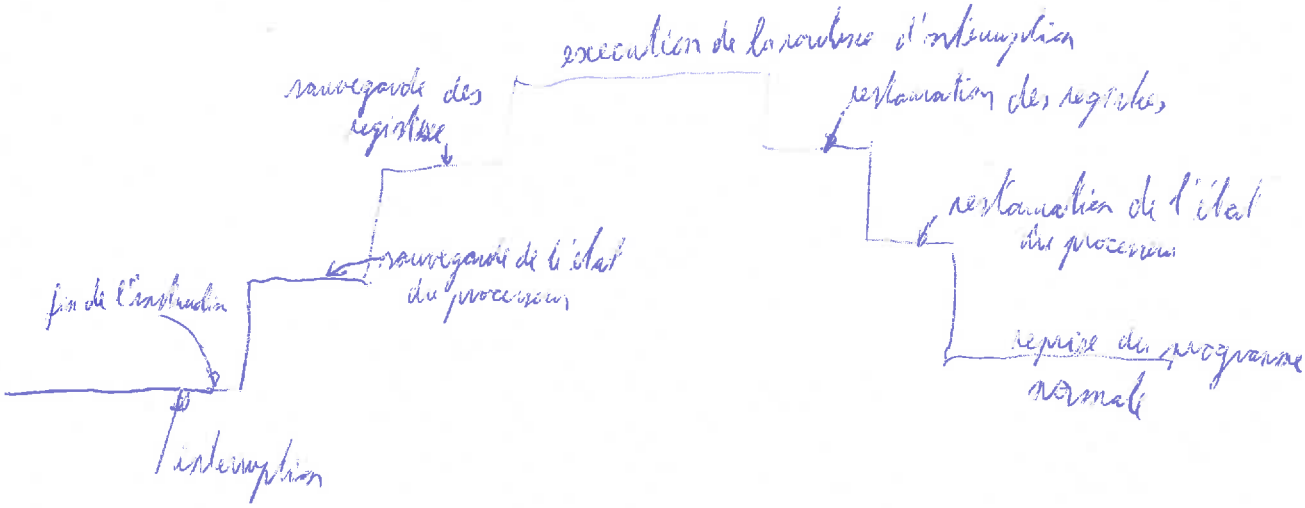
+ registre ou pile

Systèmes Embarqués 1 & 2: Travail écrit no 3.

8

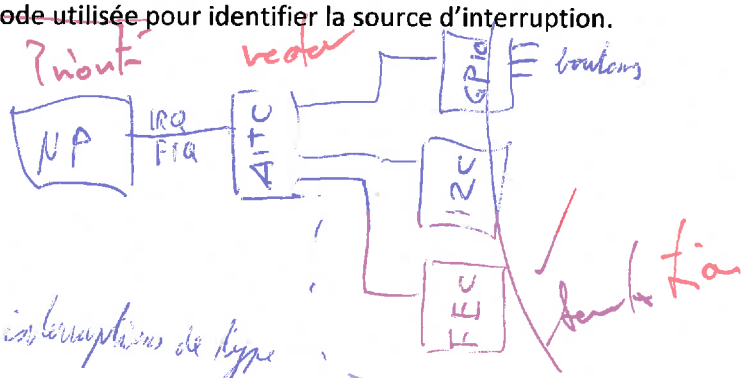
Problème n° 2 (Interruptions)

a) Expliquez à l'aide d'un graphique la commutation de contexte d'interruption (sauvegarde et restauration) ainsi que les opérations effectuées à chaque étape du traitement dès la levée de l'interruption.



4

b) Décrivez à l'aide d'une figure le système d'interruption matériel permettant de connecter des périphériques internes (I2C, UART, SPI, ...) et des périphériques externes (boutons poussoir, ...) au cœur du microprocesseur de l'i.MX27 et citez la méthode utilisée par chaque composant du système d'interruption, ~~la méthode utilisée~~ pour identifier la source d'interruption.



3

Le système gère les interruptions de type FIQ et IRQ à travers l'aide. Quand une IRQ ou FIQ est traitée, il demande à l'aide le type de vecteur qui a causé l'interruption. Si c'est le GPIO, il demande alors au GPIO le numéro de la ligne qui a généré l'interruption.

c) Décrivez succinctement la latence d'interruption et citez la raison principale qui peut la faire varier. Donnez le terme technique de cette variation.

C'est le temps entre le moment où l'interruption est signalée et le moment où la première instruction de la routine d'interruption est exécutée.

1

Systèmes Embarqués 1 & 2: Travail écrit no 3.

3

Problème n° 3 (Interruptions)

a) Décrivez les opérations à effectuer pour autoriser et/ou bloquer les interruptions au niveau du microprocesseur

0

b) Citez la marche à suivre pour initialiser, configurer et installer une routine d'interruption au niveau le plus basé sur le microprocesseur (1er étage de traitement d'une interruption).

- déclarer la routine d'interruption
- copier son adresse à l'adresse mémoire du type d'interruption (IRQ, FIQ, ...) à traiter.
- activer les interruptions

0,5

c) Implémentez les opérations assembleur que le microprocesseur devra effectuer pour sauvegarder et restaurer l'état du microprocesseur lorsqu'une interruption matérielle (IRQ ou FIQ) est levée

~~STMFD r0!,~~ MRS r0, CPSR

0

d) Un programmeur désire utiliser une partie des instructions non définies du jeu d'instruction du microprocesseur pour simuler des fonctions d'un coprocesseur. Pour ce faire, il a défini le format de ces nouvelles instructions comme suit :

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 (F)	1 0 1 1 (B)	0 0 1 0 (2)	OPCODE

Sachant que l'adresse de retour (adresse suivant la nouvelle instruction) est contenue dans le registre « r0 », implémentez la fonction « handle_undef » permettant de traiter ces nouvelles instructions en appelant la routine « int handle_opcode (int opcode) ». Si le format de la nouvelle instruction ne correspond pas au format défini, la valeur -1 est retournée dans « r0 ».

```
int handle_undef (uint32_t inst){
    if(inst & 0xFFF00000 != 0xFB200000) return -1;
    return handle_opcode(inst & 0xFFFFF);
}
```

2

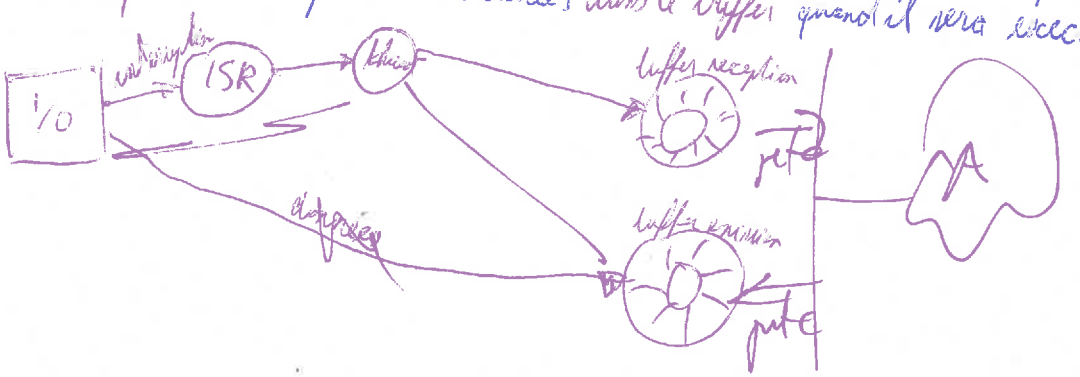
3

Systèmes Embarqués 1 & 2: Travail écrit no 3.

Problème n° 4 (Entrées/Sorties)

a) Décrivez succinctement le traitement des entrées/sorties par interruption et traitement par thread. Utilisez un schéma de principe pour soutenir la description.

Quand des données sont reçues, elles sont stockées dans un buffer et une interruption est levée. Cette interruption va créer un thread qui va récupérer les données dans le buffer quand il sera exécuté.



b) Calculez la latence maximale autorisée côté application pour la réception de trames reçues d'un contrôleur Ethernet à 100Mb/s en mode full-duplex. Le pilote gérant le contrôleur Ethernet est implémenté en mode par interruption. Le tampon de réception est composé de 300 entrées permettant de réceptionner 250 bytes chacune. Si la taille des paquets reçus dépasse la taille maximale d'une entrée, alors plusieurs entrées seront utilisées pour stocker le paquet dans le tampon de réception. Côté réseau des paquets de 1250 bytes (framing compris) sont émis en burst de 2 ms à plein débit et suivi ensuite d'une pause de 2 ms.

taille tampon = 300 entrées · 250 bytes/entrée = 75000 bytes

Nbr bytes reçus en 2 ms = $\frac{100\,Mb/s}{8} \cdot 2\,ms = \frac{10^8 \cdot 2 \cdot 10^{-3}}{8} = \frac{10^5}{4} = \frac{100000}{4} = 25'000$ Bytes = nbr bytes émis en 4 ms

latence max = $\frac{75000\,bytes}{25'000\,bytes/4\,ms} = 75000\,bytes \cdot \frac{4\,ms}{25000\,bytes} = 12\,ms$

c) Implémentez la fonction « `void puts(const char* s)` » permettant d'émettre un caractère sur une interface série au travers du contrôleur ci-dessous. L'émission se fera par scrutation.

```
#define STAT_TRDY (1<<7)  // transmitter ready: character could be sent
#define STAT_RRDY (1<<0)  // receiver ready: character has been received
static volatile struct uart_ctrl {
        uint16_t stat;        // status register
        uint16_t ctrl;        // control register
        uint16_t txbuf;       // transmit buffer
        uint16_t rxbuf;       // receive buffer
} * uart = (struct uart_ctrl*)0x1001c000;
```

```
void puts(const char *s) {
    while (*s != 0) {
        while (uart->stat & STAT_TRDY == 0);
        uart->txbuf = *s;
        s++;
    }
}
```

Systèmes Embarqués 1 & 2: Travail écrit no 3.

9

Problème n° 5 (Systèmes d'exploitation)

a) Décrivez succinctement la fonction du scheduler du noyau d'un système d'exploitation.
Citez les deux types de scheduler avec une brève explication

Le scheduler a pour fonction de gérer la transition entre les processus. C'est lui qui en choisi un quand celui en cours d'execution s'arrête.

3

coopératif : le scheduler peut seulement executé les threads, puis il doit attendre qu'ils veuillent bien laisser la main à un autre.

pré-emptif : le scheduler peut interrompre un processus quand il veut pour laisser s'executer un autre processus.

b) Décrivez succinctement la commutation de contexte entre deux threads.
Indiquez les éléments importants à sauvegarder et restaurer lors de la commutation de contexte.

Pour passer d'un thread à un autre, l'état du processeurs et des registres doivent être sauvés dans la stack du thread.

1 ½

Puis l'état du processeur et des registres sont restaurés à partir de la stack du thread à executé.

c) Décrivez succinctement le fonctionnement d'une sémaphore et de ses attributs principaux.

Une sémaphore est un élément ~~de synchronisation~~. Il permet la synchronisation de plusieurs threads de façon à ce qu'un nombre défini puisse accéder à une zone critique.

2

Ils sont composé d'un compteur permettant de connaître le nombre de place libres et d'une liste de thread en attente.

d) Implémentez la fonction « `void sema_wait (int id);` » permettant de prendre le sémaphore.
Quelques éléments de réalisation :

```
struct tcb {/*...*/ enum states state; };
struct tcb* running_tread;
struct semaphore {/*...*/} sema [200];
void reschedule();
void sema_append_thread (struct semaphore*, struct tcb*);
```

```
void sema_wait (int id){
    ~~struct semaphore * s = sema[id];~~ interrupt_disable();
    sema[id].counter--;
    if(sema[id].counter < 0){
        sema_append_thread(sema[id], running_thread);
        running_thread->state = BLOCKED;
        reschedule();
    }
    interrupt_enable();
}
```

2