Hes·so Haute Ecole Spécialisée de Suisse Occidentale

Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Microprocesseurs 1 & 2: Travail écrit no 3.

Nom : Monney **Prénom :** Loïc 5.5 **Classe : I/2** **Date : 22.04.2013**

8

Problème n° 1 (interfaçage C - assembleur)

a) Codez en assembleur la fonction « f1 » ci-dessous :

```
typedef int (*op_t) (int p1);
int f1 (int a1, int a2, op_t op1, int a3) {return a1 + a2 + op1(a3);}
```

r0 r1 r2 pointeur fonction 32 bits r3

r0-r3

valeur de retour dans r0

```
f1: nop
    push {r4, lr}
    mov r4, r0
    add r4, r1
    str r0, r3
    blx r2;
    add r0, r4
    pop {r4, pc}
```

3.1

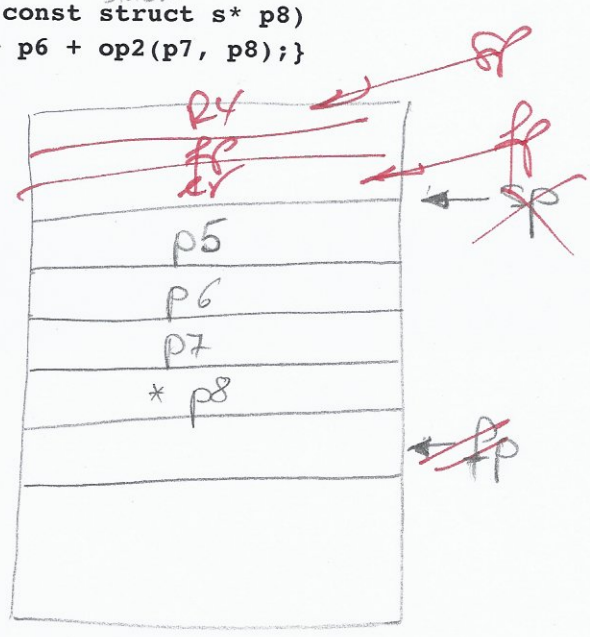
b) Dessinez l'état de la pile pour la fonction « f2 » ci-dessous juste avant l'appel de la fonction « op2 ».
Le compilateur utilise le « frame pointer » (fp) pour accéder les paramètres.
Indiquez sur le graphique la position du « fp » et du « sp ».

```
struct s {int s1; int s2; int s3;};
extern int op2(int a1, const struct s* a2);
int f2 (int p[4], int p5, int p6, int p7, const struct s* p8)
  {return p[0] + p[1] + p[2] + p[3] + p5 + p6 + op2(p7, p8);}
```

r0-r3 stack stack stack stack

Les paramètres sont mis sur la stack de droite à gauche.

2



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

2

Passage par valeur : les données sont copiées sur la stack / dans les registres.

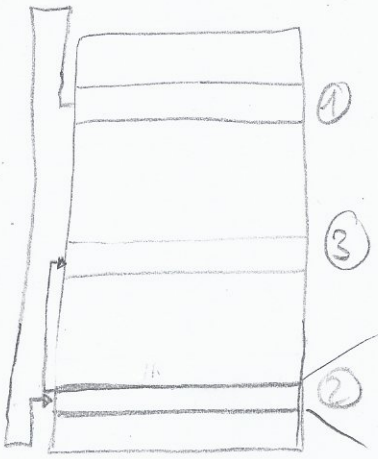
Passage par référence : les adresses sont passées en paramètres et sont copiées sur la stack / dans les registres.

Microprocesseurs 1 & 2: Travail écrit no 3.

9

Problème n° 2 (Interruptions)

a) Décrivez succinctement le mécanisme implémenté par le processeur ARM pour trouver et appeler la routine de service d'interruption (ISR) lors de la levée d'une interruption (IRQ/FIQ/SWI) ou d'une exception (undef/data abort/prefetch abort)

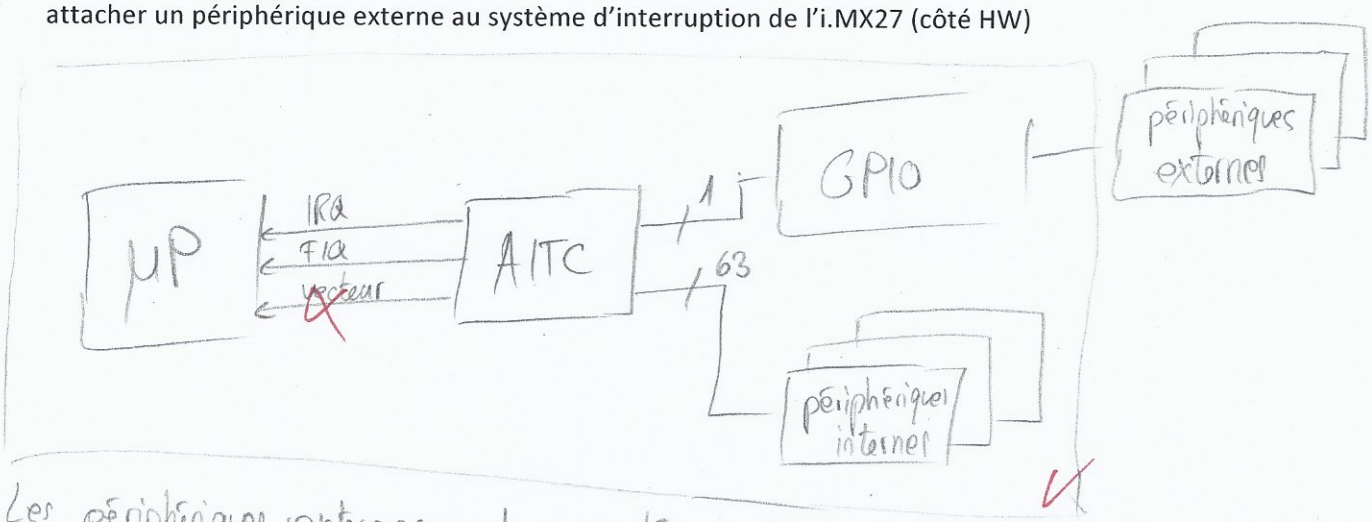


4

- Dans une mémoire ① en lecture seule, l'adresse d'une case mémoire inscriptible y est référencée (②). Dans cette deuxième case mémoire (②), qui est r-w, il nous faut aller inscrire l'adresse d'une routine de traitement.
- Lorsqu'une interruption se produit, le µP ira lire l'adresse ① puis lira l'adresse ② et sautera finalement à notre routine enregistrée

une case pour IRQ
une autre pour FIQ,
une autre pour SWI
... } Σ=6 ✓

b) Décrivez succinctement le système d'interruption de l'i.MX27 du point de vue HW et indiquez comment attacher un périphérique externe au système d'interruption de l'i.MX27 (côté HW)



3

4

Les périphériques externes sont connectés au GPIO, qui ira les interroger par scrutation lorsqu'une interruption est déclenchée par eux.

µP: –
AITC: –
GPIO: –
Périph: –

c) Décrivez succinctement la **commutation de contexte** et la **latence** d'interruption

2

commutation de contexte : sauvegarde et rétablissement des registres du processeur afin de préserver l'état des valeurs pour qu'elles ne soient pas altérée par l'exécution d'une autre série d'instructions. ✓

latence : temps entre le moment où l'interruption est survenue et le moment où le code de l'ISR est exécuté. ✓

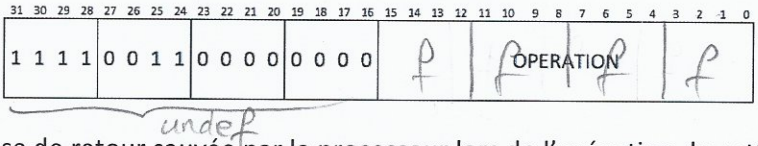
Microprocesseurs 1 & 2: Travail écrit no 3.

7

Problème n° 3 (Interruptions)

Dans le jeu d'instructions des processeurs ARMv5, codé sur 32 bits, il existe un certain nombre d'instructions qui ne sont pas définies. Ces instructions, si elles sont exécutées, génèrent l'exception « undef ».

Un programmeur décide d'utiliser ces instructions pour créer un nouveau jeu d'opérations. La figure ci-dessous représente le format du jeu d'opérations supplémentaires. Les bits 31 à 16 permettent d'identifier le nouveau jeu d'opérations. Le numéro de l'opération à exécuter est codé sur les bits 15 à 0. Les registres R1, R2 et R3 permettent de passer trois arguments (a1, a2 et a3) à l'opération. Le résultat de celle-ci est finalement retourné dans le registre R0.



Pour rappel, l'adresse de retour sauvée par le processeur lors de l'exécution de cette instruction n'a pas d'offset / de décalage.

a) Implémentez en assembleur la routine d'interruption permettant de traiter le nouveau jeu d'opérations en appelant la routine « int process_opers (int opération, int a1, int a2, int a3) ». Si le contenu des bits 31 à 16 ne correspond pas au pattern ci-dessus, la valeur -1 sera retournée.

5

```
isr: nop
     push {r4-r12, lr}
     sub lr, #4
     ldr r0, [lr]
     mov r4, #f300
     ldr r5, r0
     lsr r5, #16
     cmp r5, r4
     bneq error
     mov r4, #0xffff
     and r0, r4
     ldr r6, =process_oper
     bl r6
     b end

error:
     mov r0, #-1
     b end

end: nop
     pop {r4-r12, pc}^
```

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

1. Lire la documentation afin de trouver à quelle adresse mémoire aller écrire l'adresse de isr (pour undef).
2. mettre l'adresse de isr à cette adresse.

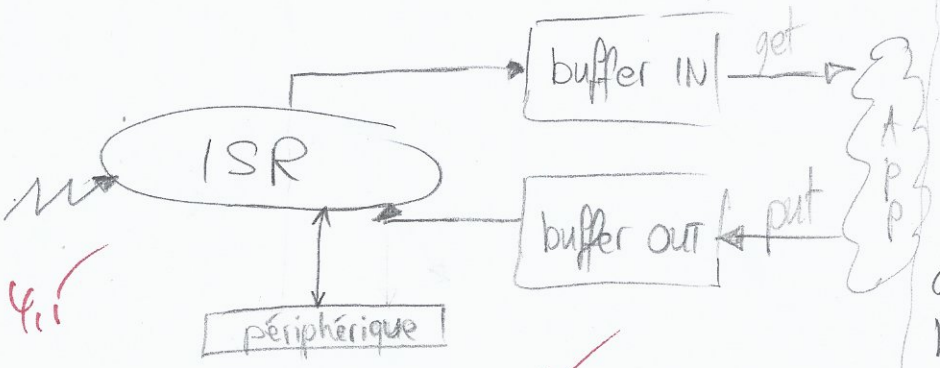
1.5

Microprocesseurs 1 & 2: Travail écrit no 3.

10

Problème no 4 (entrées/sorties)

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



Le périphérique lève des interruptions lorsqu'il est prêt à envoyer/lire des données.
Avec une routine d'interruption on place les données reçues dans le buffer d'entrée et/ou les données à envoyer du buffer de sortie vers le périphérique

4,5 ✓

b) Calculez la latence maximale autorisée côté application pour la réception de paquets d'un contrôleur Ethernet à 1Gbps full-duplex. Le pilote gérant le contrôleur Ethernet est implémenté en mode d'interruption. Il dispose d'un tampon circulaire de 500 entrées avec une taille de 1500 bytes par entrée. Côté réseau, les paquets de 250 octets sont émis à intervalle de 10 us.

2

nb d'éléments par ms : 100
On a 500 entrées => ils faut 5ms pour remplir les 500 entrées
Il faut qu'on aille au moins vider le buffer tous les 5ms (plus souvent mieux c'est) ✓

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

```
#define STATUS_RXRD (1<<0)  /* receiver ready: character has been received */
#define STATUS_TXRD (1<<1)  /* transmitter ready: character could be sent  */
struct serial_ctrl {
      uint8_t status;  /* registre de statut voir bits ci-dessus */
      uint8_t control; /* registre de contrôle du périphérique   */
      uint8_t tx;      /* registre pour l'émission des données   */
      uint8_t rx;      /* registre contenant les données reçues  */
};
static volatile struct serial_ctrl* serial = (struct serial_ctrl)0x10008000;
```

3

```
void putchar(char c){
  while((serial->status & STATUS_TXRD)==0); //périphérique occupé
  serial->tx = c;
}
```

✓

Microprocesseurs 1 & 2: Travail écrit no 3.

10

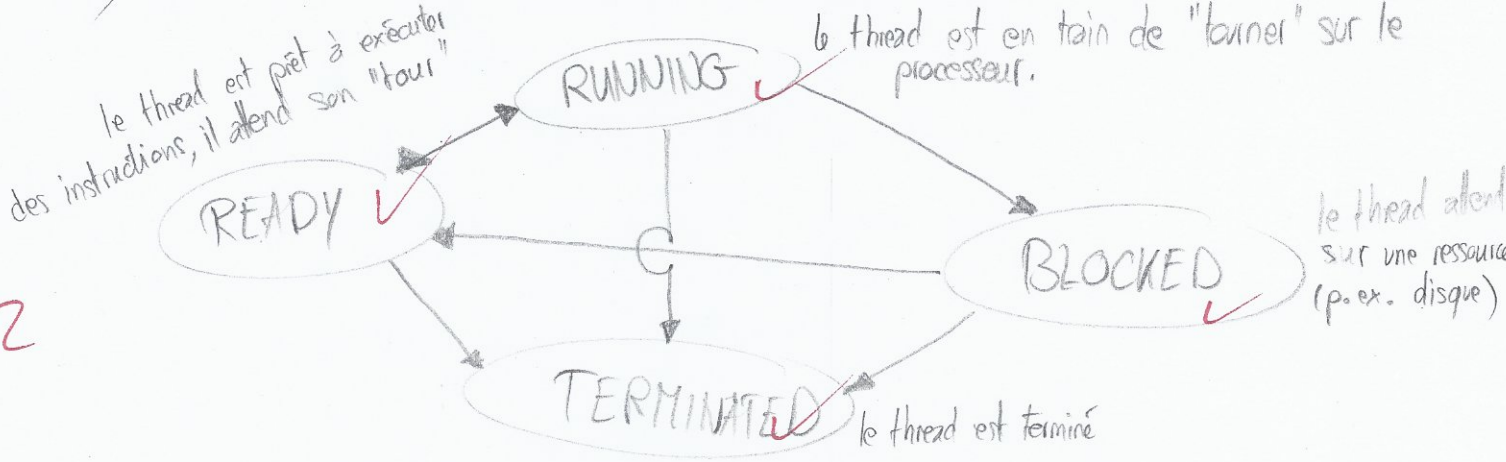
Problème n° 5 (système temps réel)

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

2,5

- scheduler
- mécanisme de synchronisation
- gestionnaire de mémoire
- mécanisme de passage de message
- timer
- mécanisme de traitement des interruptions
- Process/thread

b) Citez, à l'aide d'une machine d'états, les états d'un thread et indiquez leur fonction.



2

c) Sur la base des structures et variables ci-dessous, implémentez la fonction « void sema_wait (int id) », permettant de prendre le sémaphore.

```
struct tcb {/*...*/ enum states state; struct tcb* chain; /*...*/};
struct semaphore {long count; bool is_used; struct tcb* chain;};
struct tcb* running_thread;
struct semaphore sema[200];
extern rtos_reschedule();
```

5

```
void sema_wait (int id)
{
  //désactive les interruptions HW
  rtos_disable();
  sema[id].count--;
  if(sema[id].count<0)
  {
    if(sema[id].chain==0)
    { sema[id].chain = running_thread;
    }
    else
    {
      struct tcb* last = sema[id].chain;
      while (last->chain != 0){
        last = last->chain;
      }last->chain = running_thread;
      running_thread->state = STATE_BLOCKED;
      rtos_reschedule();
    }

    //réactivation des int. HW
    rtos_enable();
  }
```