UFR IMA

Informatique & Mathématiques Appliquées

Grenoble 1

UNIVERSITE
JOSEPH FOURIER

Sciences, Technologie, Médecine

Institut des Sciences et Techniques de Grenoble – RICM-1

Lundi 5 Janvier 2004

EXAMEN D'ARCHITECTURE LOGICIELLE ET MATERIELLE

Durée 3 heures
Tous documents autorisés
L'examen comporte deux parties («□soft□» et «□hard□») indépendantes.

Question 1: (10 points)

Voici le début et la fin d'un programme en langage C:

```
#include <stdio.h>

void main ()
{
int tab[9];
int i;


i=0;
while (…)
        {
        if (…)
        {tab[i]=1;}
        else
        {…}
        printf("indice %d, element %d\n",i,tab[i]);
        i=i+1;
}
}
```

Voici le résultat en assembleur ARM de la compilation de ce programme (gcc -S fibo.c):

```
        .file   "fibo.c"
        .section        .rodata
        .align  2
.LC0:
        .ascii  "indice %d, element %d\n\000"
```

```
        .text
        .align  2
        .global         main
        .type   main,function
main:
        mov     ip, sp
        stmfd   sp!, {fp, ip, lr, pc}
        sub     fp, ip, #4
        sub     sp, sp, #40
        mov     r3, #0
        str     r3, [fp, #-52]
.L2:
        ldr     r3, [fp, #-52]
        cmp     r3, #8
        ble     .L4
        b       .L3
.L4:
        ldr     r3, [fp, #-52]
        cmp     r3, #0
        beq     .L6
        ldr     r3, [fp, #-52]
        cmp     r3, #1
        beq     .L6
        b       .L5
.L6:
        ldr     r3, [fp, #-52]
        mvn     r2, #35
        mov     r3, r3, asl #2
        sub     r1, fp, #12
        add     r3, r3, r1
        add     r2, r3, r2
        mov     r3, #1
        str     r3, [r2, #0]
        b       .L7
.L5:
        ldr     r3, [fp, #-52]
        mvn     r2, #35
        mov     r3, r3, asl #2
        sub     ip, fp, #12
        add     r3, r3, ip
        add     r0, r3, r2
        ldr     r3, [fp, #-52]
        mvn     r2, #39
        mov     r3, r3, asl #2
        sub     r1, fp, #12
        add     r3, r3, r1
        add     r1, r3, r2
        ldr     r3, [fp, #-52]
        mvn     r2, #43
        mov     r3, r3, asl #2
        sub     ip, fp, #12
        add     r3, r3, ip
        add     r3, r3, r2
        ldr     r2, [r1, #0]
        ldr     r3, [r3, #0]
        add     r3, r2, r3
        str     r3, [r0, #0]
.L7:
```

```
        ldr     r3, [fp, #-52]
        mvn     r2, #35
        mov     r3, r3, asl #2
        sub     r1, fp, #12
        add     r3, r3, r1
        add     r3, r3, r2
        ldr     r0, .L8
        ldr     r1, [fp, #-52]
        ldr     r2, [r3, #0]
        bl      printf
        ldr     r3, [fp, #-52]
        add     r3, r3, #1
        str     r3, [fp, #-52]
        b       .L2
.L3:
        mov     r0, r3
        ldmea fp, {fp, sp, pc}
.L9:
        .align  2
.L8:
        .word   .LC0
.Lfe1:
        .size   main,.Lfe1-main
        .ident  "GCC: (GNU) 3.2.2"
```

Pour vous aider□

Les instructions assembleur du programme commencent à l'étiquette **main.**

sp est le registre pointeur de sommet de pile

fp est le pointeur de la base du contexte d'une procédure

mvn r2, #35 affecte r2 par le complément à 1 de 35, c'est-à-dire la valeur –36 en complément à 2. Exemple sur 4 bits□our la valeur 3: complément à 1 (0011)= 1100= -4 (codé en complément à 2)

De même mvn r2, #39 met la valeur –40 (codée en complément à 2) dans le registre r2

ldr r0,.L5 est équivalent à **ldr r0, =.L5**

printf est une fonction de la bibliothèque d'entrée/sortie standard du langage C permettant d'afficher une chaîne de caractères à l'écran. Le premier paramètre est une chaîne de caractères. Les paramètres suivants dépendent du contenu du premier paramètre. Une occurrence de %d dans la chaîne implique un paramètre de type entier supplémentaire. C'est la valeur de ce paramètre entier qui est affichée à l'écran à la place de %d.

Répondre aux questions suivantes :

1. A quoi servent les instructions mov ip, sp ; stmfd sp!, {fp, ip, lr, pc} et ldmea fp, {fp, sp, pc} ?
   Quels sont leurs effets ?

2. Où sont stockées les variables du programme (l'entier i et le tableau tab) ?
   Représenter l'état de la pile lors du passage à l'étiquette .L2:
   Pointeurs sp et fp
   Emplacement des variables du programme

3. Quelle est la condition du while apparaissant dans le programme ? Quelles sont les étiquettes utilisées pour traduire le while ? Expliquez comment est traduit le while.

4. Quelle est la condition du if apparaissant dans le programme ? Quelles sont les étiquettes utilisées pour traduire le if ? Expliquez comment est traduit le if.

5. Comment est calculée l'adresse de tab[i] ? Donnez la suite d'instructions qui calcule cette adresse et commentez là. Précisez l'ordre des éléments du tableau en mémoire.

6. Compléter le programme C. Quel est le contenu du tableau tab à la fin du programme ?

7. Les instructions qui suivent l'étiquette .L7 sont la traduction de l'appel à la fonction printf « printf("indice %d, element %d\n",i ,tab[i]); ».
   Où sont stockés les 3 paramètres de la procédure printf ? Quelles sont les valeurs de ces trois paramètres lors de l'exécution du programme au moment de l'appel à printf ?

8. Que faudrait-il changer dans ce programme pour que tab soit un tableau d'entiers codés sur un seul octet ?

9. Quelles seraient les modifications à faire dans ce programme ARM si la variable i était déclaré en entier non-signé (unsigned int i ;) ?

10. De même quelles seraient les modifications si le tableau tab était un tableau d'entier sur 4 octets codés en base 2 (non signé : unsigned int tab[9] ;) ?

Question 2: (10 points) Machine algorithmique

On veut réaliser un circuit permettant de calculer les éléments de la suite de Fibonacci définit par $U_n = U_{n-1} + U_{n-2}$

Ceci jusqu'à ce que U_n dépasse une valeur donnée X. On prend $U_0=U_1=1$.

Soit l'algorithme suivant qui effectue ce calcul :

Lexique :

X : un entier > 0 ;

U_{n-1} et U_{n-2} : deux entiers > 0 ; { les données } ;

U_n : un entier ≥ 0 ; { le résultat }

Algorithme :

$U_{n-1} = U_{n-2} = U_n = 1$

Tant que $U_n < X$ faire

Debut

$U_n = U_{n-1} + U_{n-2}$

$U_{n-2} = U_{n-1}$

$U_{n-1} = U_n$

Fin

On utilise une architecture « partie opérative/partie contrôle » pour ce circuit. On ne s'intéresse pas à l'initialisation de X. X est supposé donné dans un registre au départ. Le circuit délivre les U_n successifs. Il délivre un signal **fin** pour signifier la fin des calculs. L'entrée **init** permet d'initialiser le circuit.

1- Donnez les 6 premières valeurs de U_n

2- Enumérez les opérations et les variables apparaissant dans cet algorithme. Quels sont les comptes-rendus que la partie opérative doit envoyer à la partie contrôle ?

3- En déduire une partie opérative permettant de réaliser tous les calculs et affectations de variables apparaissant dans cet algorithme.

Donnez un nom à chacun des signaux de commandes de cette partie opérative et faites les apparaître sur le dessin de la PO. Une ébauche d'une partie opérative est

donnée en annexe, mais vous pouvez la changer à votre guise. On pourra rendre l'annexe complétée.
On précisera le codage des commandes de l'UAL (OpUAL) pour chacune des opérations qu'il effectue.

On précisera comment est effectuée l'initialisation des registres U_n, U_{n-1} et U_{n-2}

4- Le graphe d'un automate indiquant les calculs à effectuer dans la PO est donné en annexe. Précisez pour chaque état la valeur des commandes de la PO. Précisez les entrées de cet automate en fonction des sorties de la PO.

5- Réalisez le circuit réalisant cet automate à l'aide de bascules D et de portes logiques. Pour cela précisez le codage des états. Donnez les équations des sorties et des variables d'états. Dessinez l'architecture du circuit (bascules, calcul de l'état suivant et calcul des sorties).

6- Complétez le chronogramme donné en Annexe. Les bascules D utilisées dans les registres sont sensibles au front montant de l'horloge (CK). **Etat** est la valeur de l'état courant de la partie contrôle (sortie des bascules). **NouvelEtat** est la valeur de l'état suivant (entrée des bascules). **ChargerUn** est la commande de chargement du registre U_n (Enable des bascules). X est bien sûr supposé supérieur à 1.

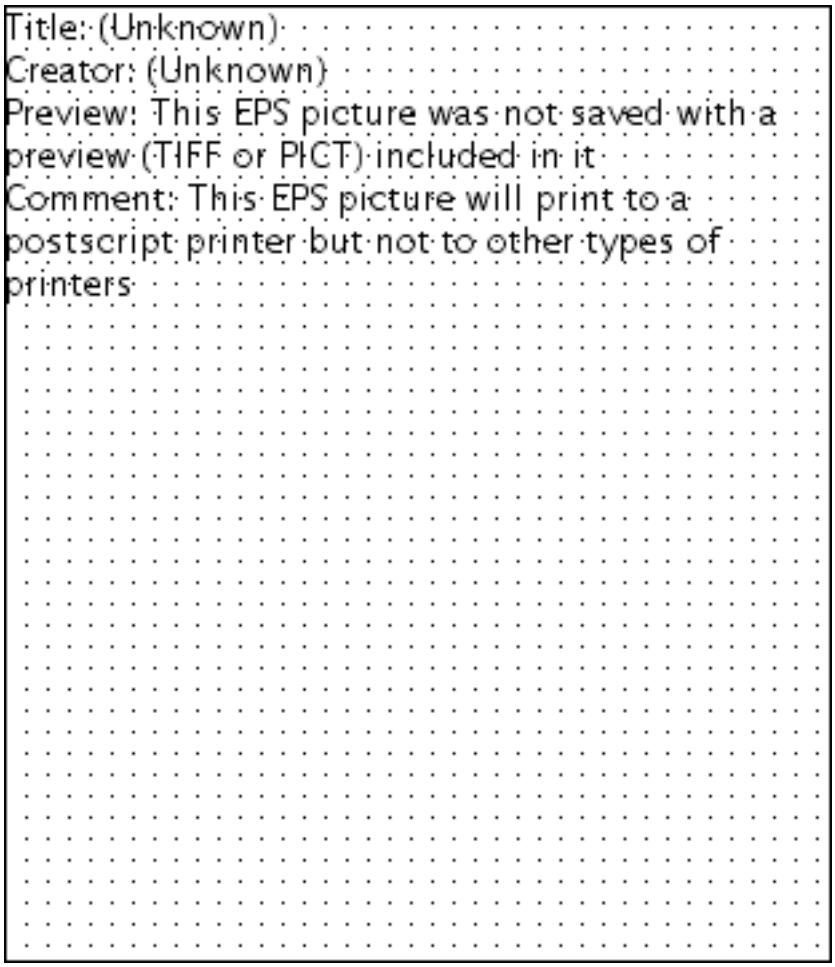
7- Combien de coup d'horloge faut-il pour calculer un U_n ? Si l'horloge du circuit a une fréquence de 1 giga hertz combien faut-il de temps pour calculer 1 million d'éléments de la suite ?

8- Si les registres et les bus de votre circuit sont sur 32 bits, quelle est la valeur maximale de U_n que peut calculer le circuit ? Quelle est la valeur maximale que l'on peut donner à X pour être sûr que tous les U_n délivrés par le circuit soient justes? On précisera le codage utilisé pour les variables de l'algorithme.

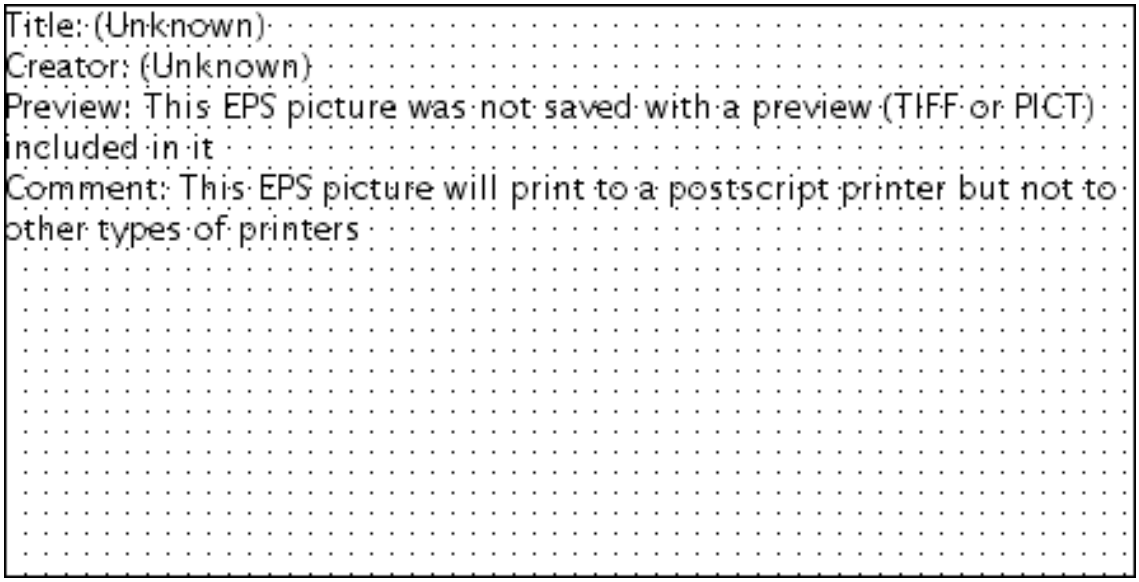
9- Quelles sont les modifications à apporter à la PO si l'on veut pouvoir effectuer les trois calculs $U_n = U_{n-1} + U_{n-2}$, $U_{n-2} = U_{n-1}$ et $U_{n-1} = U_n$ dans un même état de la PC ?

ANNEXES

Graphe de l'automate de la partie contrôle



Ebauche de la partie opérative



Chronogrammes