William ISABELLE et Ali GHAMMAZ
Le 29/10/2021

COMPTE RENDU TP PARTIE SOFT

Q1- En quelques phrases persuadez moi de la validité de l'algorithme.

L'algorithme procède par élimination : il s'agit de supprimer d'une table dont les valeurs sont initialisés tous à 1 tous les multiples d'un entier (autres que lui-même).
En supprimant tous ces multiples, à la fin il ne restera que les entiers constituant les indices du tableau qui ne sont multiples d'aucun entier à part 1 et eux-mêmes, et qui sont donc les nombres premiers.

Q2- Expliquez les caractères $@ et $< apparaissant dans le makefile

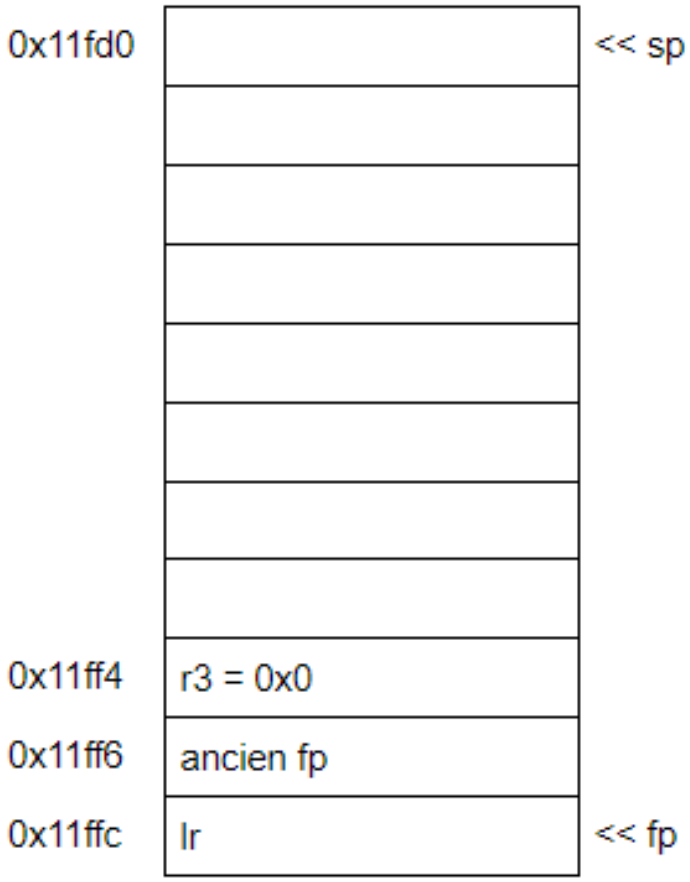
$@ est le nom de la cible générée
$< le premier prérequis (généralement un fichier source).

Q3- A quelle adresse commence la pile ? Comment le savez-vous ?

La pile commence à l'adresse 0x11ffc, en effet en exécutant le programme sous gdb et on se plaçant après l'empilement de lr, fp:
on retrouve (gdb) p /x $sp
$2 = 0x11ff8
puisque la pile contient 2 éléments ? la case en dessous est pointé par fp
et donc fp = 0x11ffc c'est l'adresse du commencement de la pile .

Q4- Dessinez l'état de la pile après la troisième instruction du main (sub sp, sp, #40). Précisez les cases pointées par sp et fp.

sub sp, sp, #40 : on alloue 10 cases dans la pile afin de créer un espace pour les variables locales .
schema de pile :



Q5- Donnez la valeur (en hexadécimal) de fp et sp à ce moment là.

(gdb) p /x $sp
$3 = 0x11fd0
(gdb) p /x $fp
$4 = 0x11ffc

Q6- Où sont stockées les trois variables locales du main: tab, i et j ? Donnez la valeur en hexadécimal de leurs adresses.

```
sub     r3, r11, #44
mov     r2, #1
ldr     r1, [r11, #-8]
mov     r0, r3
```

r3 pointe sur la case memoire de la pile 11 cases en dessus de fp
la valeur de r3 est par la suite attribué à r0 , donc tab est stocké dans r0

1 est stocké dans r2 :mov   r2, #1
r1 contient i   :ldr   r1, [r11, #-8]

(gdb) p /x $r0
$7 = 0x11fd0
(gdb) p /x $r1
$8 = 0x0 (pour i=0)
(gdb) p /x $r2
$9 = 0x1

Q7- Donnez en hexadécimal l'adresse de tab[N-1]. Complétez le dessin de la pile en donnant l'emplacement des variables locales du main. Précisez leur adresses en héxadécimal et par rapport au contenu du registre fp

l'adresse de tab[N-1]=r0 + (N-1) = 0x11fd0 + 0x29 = 0x11ff9

Q8: Expliquez comment est traduite l'instruction for (i=0; i<N; i++) en précisant le calcul de la condition d'arrêt de la boucle et les étiquettes utilisées.

```
mov     r3, #0
str     r3, [r11, #-8]
b       0x10078 <main+56>
ldr     r3, [r11, #-8]
cmp     r3, #29
ble     0x10058 <main+24>
...
ldr     r3, [r11, #-8]
add     r3, r3, #1
```

mov r3, #0 : r3 represente l'indice i qui s'incremente au cours de la boucle
r3 est intialisé à 0 (i=0)

str r3, [r11, #-8] : la valeur de r3 est en suite stockée dans la pile 2 cases en dessus de fp

b 0x10078 <main+56> : effectue un branchement à l'adresse 0x10078

ldr r3, [r11, #-8] : on charge r3 depuis la pile et donc r3=0

cmp r3, #29 : on compare r3=i avec N-1=29

si r3 est plus petit ou égale à 29 on effectue un branchement vers les instructions de la boucle .

la condition d'arrét est donc r3=N=30 , c'est la ou la boucle est terminée .

lorsque la fonction affecttab(tab,i,1) est appellée pour la i-eme fois on increment i=i+1 comme ainsi :
ldr r3, [r11, #-8] : on charge r3 par la i-eme valeur stockéé en mémoire .

add r3, r3, #1 : on incremente la valeur de r3 par 1 et donc on aura r3=i+1 .

<u>Q9: Comment sont passés les valeurs des paramètres à la fonction affecttab ? Donnez la valeur en hexadécimal la valeur de ces trois paramètres avant de rentrer dans la fonction.</u>

```
sub     r3, r11, #44
mov     r2, #1
ldr     r1, [r11, #-8]
mov     r0, r3
```

sub r3, r11, #44 : r3 vient pointer sur un case de la pile , 11 cases en dessus de fp
la valeur de r3 est retrouvée ainsi:
(gdb) p /x $r3
$1 = 0x11fd0
mov r2, #1 : on atrribue au registre r2 la valeur 1 , r2 represente le 3eme parametre de la fonction affectlab .

ldr r1, [r11, #-8] : on charge le registre r1 par la valeur correspondante à l'adresse r11, #-8

pour la premiere iteration r1=0. Or r1 représente i le 2eme parametre de la fonction affectlab qui va étre incrementer au fur et à mesure par la boucle for (i=0; i<N; i++)à travers le registre r3 comme on vient de montrer dans la question 8 .

mov r0, r3 : on attribue à r0 la valeur 0x11fd0 , r0 est donc un pointeur sur une case memoire de la pile .r0 reprenste tab le 1er parametre de la fonction affecttab .

les valeurs de ces parametres en hexadecimale sont:

```
(gdb) p /x $r2
$2 = 0x1
(gdb) p /x $r1
$3 = 0x0 (pour la 1ere itération)
(gdb) p /x $r0
$4 = 0x11fd0
```

Q10- Au début de la fonction affecttab, expliquez l'instruction push {fp} :

l'instruction push {fp} empile le registre fp dans la pile
c'est équivalent à linstruction : str r11, [sp, #-4] stockant l'ancien valeur fp de dans la case juste en dessus de sp .

Q12- Où sont stockés les paramètres de la fonction ? Précisez leurs emplacements en mémoire (en hexadécimal et par rapport au contenu de fp).

```
str r0, [r11, #-8]
str r1, [r11, #-12]
str r2, [r11, #-16]
```

```
(gdb) p /x $r11
$5 = 0x11fcc
```

la valeur de r0(1er parametre)est stockée 2 cases en dessus de fp ,donc dans l'adresse 0x11fc4
la valeur de r1(2ème parametre) est stockée 3 cases en dessus de fp ,donc dans l'adresse 0x11fc0
la valeur de r2(3ème parametre) est stockée 4 cases en dessus de fp ,donc dans l'adresse 0x11fbc

Q13- Comment est réalisée l'affectation *(tab+i)=val
Donnez la valeur en hexadécimal de tab+i.

```
ldr     r3, [r11, #-12]
ldr     r2, [r11, #-8]
add     r3, r2, r3
ldr     r2, [r11, #-16]
and     r2, r2, #255
strb    r2, [r3]
```

Dans la ieme iteration On charge dans r3 la valeur de i puis dans r2 on charge la valeur du pointeur tab =0x11fd0
puis par l'instruction add r3, r2, r3 on aura dans r3 : tab+i c'est l'adresse pointant sur la ieme valeur du tableau .
Ensuite par l'instruction ldr r2, [r11, #-16] on recupere dans r2 la valeur 1
Enfin par

```
and     r2, r2, #255
strb    r2, [r3]
```

*(tab+i) reçoit la valeur 1 .

la valeur en hexadécimal de tab+i :
1ere itération i=0 : tab+i =0x11fd0
2eme itération i=1 : tab+i =0x11fd1
...
...

Q14- Observez sous gdb la pile à ce moment là, et donnez la valeur et l'adresse (en hexadécimal) de la case du tableau tab modifiée.

pour i=0
l'adresse de la case tu tableau modifiée est tab+0= tab= 0x11fd0

tout les cases du tableau sont modifiés en 1

Pourquoi l'instruction strb est utilisée ici ?
Quel est l'intérêt de l'instruction and r2, r2, #255 ?

Q15- Expliquez les trois dernières instructions de la fonction affecttab. Donnez la valeur X en hexadécimal du registre lr au moment de l'instruction bx lr. Quelle est l'instruction qui se trouve en mémoire à l'adresse X ?

```
add     sp, r11, #0
pop     {r11}
bx      lr
```

add sp, r11, #0 : le registre sp vient pointer sur la case de la pile pointée par fp .
pop {r11} : on dépile la case du sommet de la pile(ancienne valeur de fp), et on attribue à fp sa ancienne valeur .
bx lr :on effectue un branchement dans l'adresse contenue dans lr .et donc on retourne vers main à partir de l'instruction :<main+44> ldr r3, [r11, #-8]

```
(gdb) p /x $lr
$1 = 0x1006c
```

la valeur de lr est donc 0x1006c

Q16- Dans la fonction main retrouvez et commentez la traduction en ARM de if (tab[i]):

```
sub     r2, r11, #44
ldr     r3, [r11, #-8]
add     r3, r2, r3
ldrb    r3, [r3]
cmp     r3, #0
beq     0x100e8 <main+168>
```

sub r2, r11, #44 : le registre r2 pointe sur la case qui se situe à 11 cases en dessus de fp .la valeur de r2 est donc : 0x11fd0

ldr r3, [r11, #-8] : on charge le registre r3 par 2 stocké precedemment .

```
(gdb) p /x $r3
$2 = 0x2
```

add r3, r2, r3 : r3=r2+r3= 0x11fd2

ldrb r3, [r3] ldrb= charge une registre de 8 bits .la nouvelle valeur de r3 est 0x1

cmp r3, #0: on compare le registre r3 avec la valeur 0

beq 0x100e8 <main+168> : beq= branch if equal , donc on effectue le branchement à l'adresse 0x100e8 si r3=0 où on incremente la valeur de i par 1 .sinon on passe au instructions de la condition if .

Q18- Retrouvez et commentez la traduction en assembleur de j= i*2;

```
ldr r3, [r11, #-8]
lsl r3, r3, #1
str r3, [r11, #-12]
```

ldr r3, [r11, #-8] : on charge le registre r3 par la valeur 2

lsl r3, r3, #1 LSL fournit la valeur d'un registre multipliée par une puissance de deux, en insérant des zéros dans les positions de bits libérées. Dans notre cas on multiplie r3 par 2 on retrouve r3=4 .

str r3, [r11, #-12] : la nouvelle valeur de r3 est donc stockée dans la pile 3 cases en dessus de fp .

Q19- Donnez les adresses et les valeurs du tableau tab à la fin du programme main, retrouvez les nombres premiers à l'aide de celui-ci.

tab=0x11fd0

A la fin du programme les valeurs de tableau sont :
[1,1,1,1,0,1,0,1,0,0,0,1.....]
les nombres premiers sont les indices du tableaux dont la valeur est egale à 1.