DUARTE TOMAZ DE SÁ Gustavo

1er novembre 2023

Exercices chaîne de compilation

Exercice 1:

Original code:

```
uint32_t a; // global variable
...
for (uint8_t i=0; i<=a; i++)
    g();
```

Assembly code:

```
.word a
mov r0, #0
ldr r1, a   @a should be created

loop:

cmp r0, r1
bgt end
bl g
add r0, r0, #1
b loop

end:
b end
```

Même question si i est un unsigned int. Conclusion ?

Dans ce cas, l'assembleur utilisera des instructions spécifiques de 8 bits, c'est-à-dire un octet. De plus, la variable occupera seulement un octet et n'aura pas d'espace alloué de 32 bits pour elle.

Exercice 2:

Translate into assembly:

```
uint32_t *a;
uint32_t *b;
uint32_t *c;
__attribute__((naked)) void f() {
 *a += *c;
 *b += *c;
}
```

Assembly:

```
av:
    .global a
    .word 0x12345678
bv:
    .global b
    .word 0x9abcdef0
cv:
    .global c
    .word 0x10000000

    ldr r0, =a
    ldr r1, =b
    ldr r2, =c

    ldr r3, [r0]
    ldr r4, [r1]
    ldr r5, [r2]

    add r3, r3, r5

    add r4, r4, r5

    str r3, [r0]
    str r4, [r1]

end
    b end
```

Pourquoi GCC charge-t-il deux fois le contenu de *c au lieu d'une seule ?

Depuis que les pointeurs *a et *c pointent vers la même adresse mémoire, après l'exécution de la première opération d'addition, la valeur stockée à l'adresse de *a et *c sera altérée, il faut donc recharger le contenu de *c une autre fois.

Exercice 1 :

1) The code is commented to show how I achieved and thought about the results. The results were:

.rodata: 0x564285ed501c

.data: 0x564285ed7010

bss: 0x564285ed7018

heap: 0x5642869f12a0

stack: 0x7ffd72896b4c

stack 2 (sense): 0x7ffd72896b24

With that, I concluded that the sense in order at the memory is(descending): stack -> heap -> bss -> .data -> .rodata.

2) La pile augmente en descendant, comme on peut le voir dans la deuxième variable allouée dans la pile appelée "stack 2 (sens)" a une adresse inférieure par rapport à la pile. Ainsi, nous pouvons conclure que la pile a un sens de croissance descendant.

Exercice 2 :

1) Compilez sans édition de lien ce code-ci (http://bit.ly/2ApXoDl) pour ARM avec une chaîne récente, et avec les optimisations suivantes : Os, O0, O1 et O2. Pour chaque niveau d'optimisation, justifiez la taille des sections de données que vous obtenez.

O0:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000b8  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000ec  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  000000f0  2**2
                  ALLOC
```

```
  3 .rodata       00000040  00000000  00000000  000000f0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      00000034  00000000  00000000  00000130  2**0
                  CONTENTS, READONLY
  5 .ARM.attributes 0000002a  00000000  00000000  00000164  2**0
                  CONTENTS, READONLY
```

La section .rodata stocke toutes les constantes liées à l'impression de données (qui sont les seules constantes dans ce code). Cela inclut la constante déclarée "Hello World!" ainsi que les constantes d'impression telles que "x %d" et autres.

Il est important de noter qu'il y a deux versions de la chaîne "Hello World!": l'une avec le retour à la ligne \n, et une autre sans le retour à la ligne. Le compilateur a remplacé printf() par puts() car il a transmis une chaîne constante en tant qu'argument. Pour accommoder ce changement, une version de mesg sans \n a été créée.

O1:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000006c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000a0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  000000a4  2**2
                  ALLOC
  3 .rodata.str1.4 00000030  00000000  00000000  000000a4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rodata       0000000e  00000000  00000000  000000d4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .comment      00000034  00000000  00000000  000000e2  2**0
                  CONTENTS, READONLY
  6 .ARM.attributes 0000002a  00000000  00000000  00000116  2**0
                  CONTENTS, READONLY
```

Ici, il y a le émergence de la section .rodata.str.1.4. La section ".rodata.str1.4" est une partie de l'espace de données d'un fichier exécutable qui stocke des

constantes de chaînes de caractères. Ces constantes sont utilisées par le programme, mais ne doivent pas être modifiées pendant l'exécution. La notation ".rodata" est couramment utilisée pour les sections en lecture seule qui contiennent des données constantes.

O2:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000000  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  00000038  2**2
                  ALLOC
  3 .rodata.str1.4 00000030  00000000  00000000  00000038  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text.startup 0000006c  00000000  00000000  00000068  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  5 .rodata       0000000e  00000000  00000000  000000d4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .comment      00000034  00000000  00000000  000000e2  2**0
                  CONTENTS, READONLY
  7 .ARM.attributes 0000002a  00000000  00000000  00000116  2**0
                  CONTENTS, READONLY
```

O3:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000000  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  00000038  2**2
                  ALLOC
  3 .rodata.str1.4 00000030  00000000  00000000  00000038  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text.startup 0000006c  00000000  00000000  00000068  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  5 .rodata       0000000e  00000000  00000000  000000d4  2**2
```

```
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .comment      00000034  00000000  00000000  000000e2  2**0
                  CONTENTS, READONLY
  7 .ARM.attributes 0000002a  00000000  00000000  00000116  2**0
                  CONTENTS, READONLY
```

Os:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000000  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  00000038  2**2
                  ALLOC
  3 .rodata.str1.1 0000002d  00000000  00000000  00000038  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text.startup 00000068  00000000  00000000  00000068  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  5 .rodata       0000000e  00000000  00000000  000000d0  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .comment      00000034  00000000  00000000  000000de  2**0
                  CONTENTS, READONLY
  7 .ARM.attributes 0000002a  00000000  00000000  00000112  2**0
                  CONTENTS, READONLY
```

Nous pouvons constater que, pour tous les niveaux d'optimisation, la taille des sections bss et .data reste constante. En revanche, la taille de la section .text est modifiée pour chaque niveau d'optimisation, et de plus, de nouvelles sections apparaissent, telles que .rodata.str1.4 et .text.startup, qui seront abordées ultérieurement. Cela arrive parce que la section .data stocke uniquement 32 bits de données pour la variable x et la section .bss stocke 32 bits pour la variable y et 8 bits, donc il occupe une taille de 5 octets en mémoire.

Les sections .comment et .ARM.attributes contiennent des informations de compilation et d'autres données qui ne sont pas directement liées à la construction du code, donc nous ne les examinerons pas.

La réduction de la section .text est due à l'optimisation. À un niveau d'optimisation plus élevé, la section sera compressée. À un certain point, la section .text finit par avoir une taille de 0, mais la section .text.startup apparaît, ce qui est une forme d'optimisation du compilateur, plaçant les fonctions initiales (comme main) dans cette section.

Enfin, l'analyse spécifique des contenus de .rodata et .rodata.str1.4 a été effectuée lors du premier objdump qui a révélé l'apparition de .rodata.str1.4.

2.2 Remplacez const char mesg[] par static const char mesg[]. Expliquez les différences dans les sections de données par rapport à la question précédente (elles dépendent ici aussi des optimisations).

O0:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000b8  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000ec  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  000000f0  2**2
                  ALLOC
  3 .rodata       00000040  00000000  00000000  000000f0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      00000034  00000000  00000000  00000130  2**0
                  CONTENTS, READONLY
  5 .ARM.attributes 0000002a  00000000  00000000  00000164  2**0
                  CONTENTS, READONLY
```

O1:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000006c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000a0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  000000a4  2**2
                  ALLOC
  3 .rodata.str1.4 00000030  00000000  00000000  000000a4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

```
  4 .comment      00000034  00000000  00000000  000000d4  2**0
                  CONTENTS, READONLY
  5 .ARM.attributes 0000002a  00000000  00000000  00000108  2**0
                  CONTENTS, READONLY
```

O2:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000000  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  00000038  2**2
                  ALLOC
  3 .rodata.str1.4 00000030  00000000  00000000  00000038  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text.startup 0000006c  00000000  00000000  00000068  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  5 .comment      00000034  00000000  00000000  000000d4  2**0
                  CONTENTS, READONLY
  6 .ARM.attributes 0000002a  00000000  00000000  00000108  2**0
                  CONTENTS, READONLY
```

O3:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000000  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  00000038  2**2
                  ALLOC
  3 .rodata.str1.4 00000030  00000000  00000000  00000038  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text.startup 0000006c  00000000  00000000  00000068  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  5 .comment      00000034  00000000  00000000  000000d4  2**0
                  CONTENTS, READONLY
  6 .ARM.attributes 0000002a  00000000  00000000  00000108  2**0
                  CONTENTS, READONLY
```

OS:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000000  00000000  00000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000005  00000000  00000000  00000038  2**2
                  ALLOC
  3 .rodata.str1.1 0000002d  00000000  00000000  00000038  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text.startup 00000068  00000000  00000000  00000068  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  5 .comment      00000034  00000000  00000000  000000d0  2**0
                  CONTENTS, READONLY
  6 .ARM.attributes 0000002a  00000000  00000000  00000104  2**0
                  CONTENTS, READONLY
```

Comme on peut le voir, rien ne change lorsque aucune optimisation n'est appliquée. Cependant, pour toute optimisation au-dessus de O1 (y compris O1), la section .rodata disparaît. Cela se produit car il n'y a aucune raison de conserver la version d'origine contenue dans .rodata, car elle n'a été utilisée que lors de l'appel de la fonction, c'est-à-dire qu'elle n'a pas besoin d'être utilisée dans un autre contexte. Nous pouvons donc la conserver uniquement dans la section spécifique aux arguments, .rodata.str.14.

3) Remplacez const char mesg[] par const char *mesg. puis par const char * const mesg. Expliquez les différences dans le code généré et les sections de données par rapport à la question 2.

A première chose qui se produit est que nous changeons le type du message de "const" en un pointeur qui pointe vers quelque chose de constant, modifiant ainsi l'emplacement de stockage des données. Par conséquent, il passera de la section .rodata à la section .data qui pointe vers une section .rodata, où se trouve le message constant "HELLO WORLD!". Il est intéressant de noter que la section .text a également été réduite d'un octet lors de l'optimisation maximale lorsqu'il s'agit de travailler avec des pointeurs, ce qui implique naturellement que la manière de

travailler avec les instructions a été modifiée, car la manière de travailler avec la variable a également été modifiée en C.