TP 5

3. 2. Quelle option permet de :

• Lancer seulement le préprocesseur C ? riscv-none-embed-gcc -E

• De compiler et d'assembler le code source C sans faire d’édition de lien? riscv-none-embed-gcc -c

• De compiler seulement le code source C ? riscv-none-embed-gcc -S

• De spécifier le nom du fichier de sortie ? riscv-none-embed-gcc -o <file>

3. 3. L’option –v --help permet d’avoir une aide plus détaillée.

Que permettent les options :

• -debug : Activer les sortir debug

• -march : Permet de de créée l’architecture RISC-V et de générer les attributs de l’architecture

• -fverbose-asm : Permet de rajouter des commentaires dans l’output de l’assembleur

• -fomit-frame-pointer : Permet de ne peut pas générer de stack frame quand c’est possible

• -mabi : permet de set la convention tel que ABI

4.1 : riscv-none-embed-gcc -S -o main1.s main1.c

4.2 : Il s’agit d’une fonction feuille car elle n’appelle pas d’autre fonction, il y a 0 variables local et 8 paramètre d’entrée)

4.3 :

|  |  |
| --- | --- |
| Adresse | Contenu |
| sp+32 |  |
| sp+28 | A |
| Sp+24 | B |
| Sp+20 | C |
| Sp+16 | D |
| Sp+12 | E |
| Sp+8 | F |
| Sp+4 | G |
| sp | H |

4.5 :

Fonction feuille, 2 variable local, 10 paramètres d’entrée.

4.6 : La variable ptr est alloué par le pointeur s0 en -20(s0) Comme la pile doit être aligné avec des blocs de 16 octet on voit que malgré els 4 octets demandé 16 sont alloué

Attention avec la fonction appelé en plus, on a alloué ra au début.

|  |  |
| --- | --- |
| Adresse | Contenu |
| Sp+52 | J |
| Sp+48 | I |
| Sp+44 | X |
| Sp+40 | Y |
| Sp+36 | NONE |
| Sp+32 | NONE |
| Sp+28 | A |
| Sp+24 | B |
| Sp+20 | C |
| Sp+16 | D |
| Sp+12 | E |
| Sp+8 | F |
| Sp+4 | G |
| Sp+0 | H |

Le trou de NONE est présent car les éléments se place au-dessus de la pile

4.6

• Comment est allouée la variable ptr ?

Elle est allouée dynamiquement avec la fonction alloc.

• Quel va être la structure de la pile lors de l’appel de cette fonction ?

Les instructions appelées lors de l’appel de la fonction alloc sont les suivantes :

– addi sp, sp, -16 #alloue un espace de 4 mots pour le pointeur ptr

– addi s5, sp, 8 #alloue dans a5 l’adresse sp+8

– addi a5, a5, 15 #allour à a5 l’adresse sp+8+15= sp+23

– srli a5, a5, 4 #d´ecalage à droite de a5 de 4bits

– slli a5, a5, 4 #d´ecalage à gauche de a5 de 4bits

– sw a5, -20(s0) #stocke l’adresse s0-20 dans a5

En réalité le code aurait pu se résumer aux lignes suivantes :

– addi sp, sp, -16 #alloue un espace de 4 mots pour le pointeur ptr

– sw a5, -20(s0) #stocke l’adresse s0-20 dans a5

En fait, cette fonction n’alloue que 4 mots d’espace et stocke dans a5 la

Position du pointeur de pile pour dire que le pointeur se situe à l’adresse

s0-16. La pile est donc théoriquement inchangée donc considérable comme vide. En revanche, l’ajout d’instructions intermédiaires par le compilateur

N’est pas nécessaire et donc peu optimale.

4.7

1. **Variable global**:
   * Définie avec .comm global,512,4, ce qui réserve 512 octets pour global avec un alignement de 4 octets.
   * Dans la fonction main, on voit global[10] = fct2(0,1,2,3,4,5,6,7,8,9);, où global[10] reçoit la valeur retournée par fct2.
     + lui a5,%hi(global) charge l'adresse haute de global dans a5.
     + addi a5,a5,%lo(global) ajoute l'adresse basse à a5 pour obtenir l'adresse complète.
     + sw a4,40(a5) stocke le résultat de fct2 dans global[10].
2. **Variable initialized\_global**:
   * Définie dans la section .sdata (données initialisées de petite taille), avec un alignement de 2 octets.
   * Initialisée avec une valeur de 1431677610 (.word 1431677610).
   * Dans main, initialized\_global = 0x12345678; modifie sa valeur.
     + lui a5,%hi(initialized\_global) et addi a4,a4,1656 préparent la valeur 0x12345678.
     + sw a4,%lo(initialized\_global)(a5) écrit cette valeur dans initialized\_global.
3. **Fonction fct2**:
   * fct2 est une fonction prenant plusieurs arguments, qui sont empilés sur la pile (sw a0,28(sp), etc.).
   * Elle effectue plusieurs opérations arithmétiques et retourne le résultat.

En résumé, ce code assembleur montre comment les variables globales sont déclarées, initialisées et manipulées en RISC-V. La variable globale est une variable globale non initialisée, tandis que initialized\_global est une variable globale initialisée, et leurs usages sont illustrés dans les fonctions fct2 et main.

5.1 Examinez td5.map, à quelles adresses se trouve les variables globales : global et initialized global ?

• Adresse de global :0x0000000010000000

• Adresse de initialized global :0x00000000100002004

5.2 Examinez main.s :

• à quelles valeurs sont initialisés les registres $sp et $gp ?

– $sp: 0x10002

– $gp: 0x10000

• à quelle adresse se trouvent le c runtime, la fonction main ?

– Adresse du c runtime (clr lp et cp lp) : 0x00000028

– Adresse de la fonction main : 0x00000110