Architecture des Ordinateurs

Assembleur et adressage mémoire dans MIPS

2ème Année Licence Informatique

Travaux Dirigés #4

1. Contrôle préalable

Cette partie est conçue comme une vérification pour vous permettre de déterminer si vous comprenez les concepts abordés en cours ou non. Veuillez répondre par « Vrai » ou « Faux » aux questions suivantes et inclure une explication :

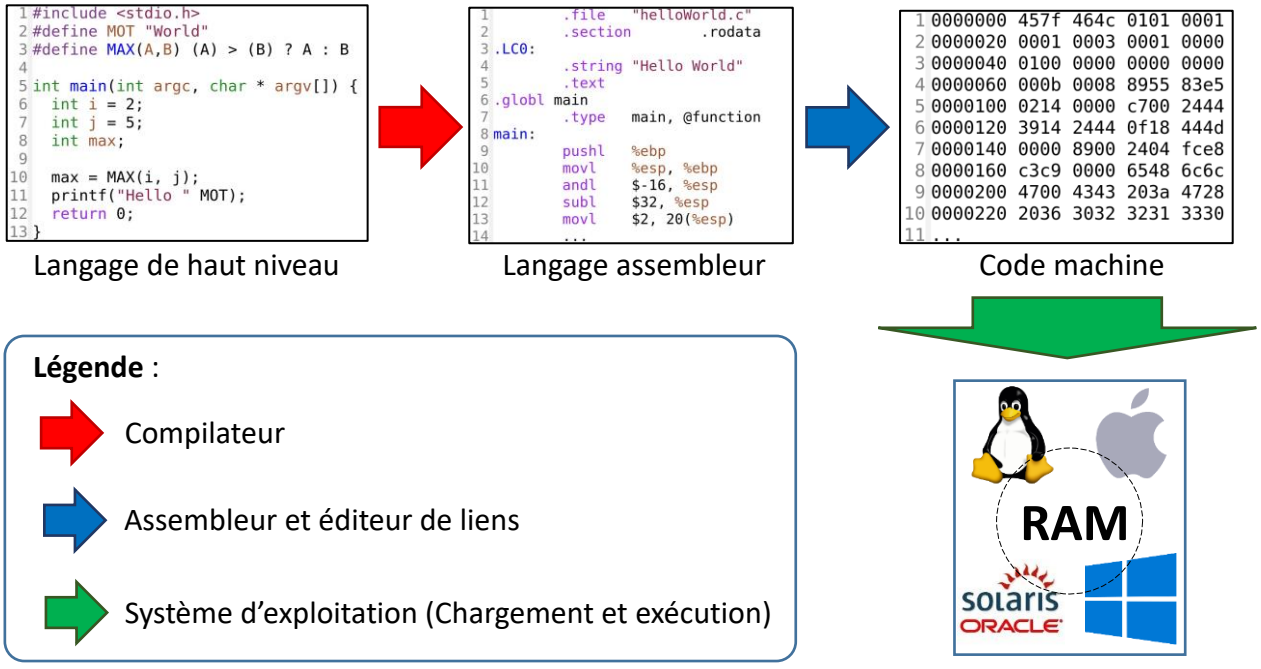
1.1. Le compilateur de langage évolué (C/C++, Fortran, …) peut générer des pseudo-instructions assembleur.

1.2. L'objectif principal de l'outil Assembleur est de générer un code machine optimisé.

1.3. Les adresses de destination de toutes les instructions de saut est complètement déterminée après l'édition de lien.

2. Compilation de programme

Voici un diagramme qui schématise le processus de création et d'exécution d'un programme à partir du code source de langage évolué comme le C.



2.1. Combien de passages dans le code l'outil Assembleur doit effectuer et pourquoi cela ?

2.2. Décrivez les six parties principales des fichiers objets générés par l'assembleur (En-tête, texte, données, table de relocalisation, table de symboles, informations de débogage).

3. Assemblage de code source MIPS

Soit un programme C qui contient une seule fonction `sum` qui calcule la somme des éléments dans un tableau. Voici ci-dessous une version compilée de ce programme en MIPS.

```
1 .import print.s       # print.s est un fichier différent
2 .data
3  array:  .word 1,2,3,4,5
4 .text
5  sum:    la   $t0, array
6          li   $t1, 4
7          move $t2, $0
8  loop:   beq  $t1, $0, end
9          addi $t1, $t1, -1
10         sll  $t3, $t1, 2
11         add  $t3, $t0, $t3
12         lw   $t3, 0($t3)
13         add  $t2, $t2, $t3
14         j    loop
15 end:    move $a0, $t2
16         jal  print_int # fonction définie dans print.s
17         ...
```

3.1. Quelles lignes contiennent des pseudo-instructions que l'assembleur doit convertir en des instructions réelles de l'ISA MIPS ?

3.2. Pour les instructions de branchement / saut, quelles étiquettes seront résolues lors de la première passe de l'assembleur ? et quelles autres dans la deuxième passe ?

3.3. Supposons que le code de ce programme, une fois charger en mémoire, commence à l'adresse `0x00400000` (voir ci-dessous). Pourquoi y a-t-il un saut de 8 entre la première et la deuxième ligne ?

```
0x00400000:  sum:    la   $t0, array
0x00400008:          li   $t1, 4
0x0040000C:          move $t2, $0
0x00400010:  loop:   beq  $t1, $0, end
0x00400014:          addi $t1, $t1, -1
0x00400018:          sll  $t3, $t1, 2
0x0040001C:          add  $t3, $t0, $t3
0x00400020:          lw   $t3, 0($t3)
0x00400024:          add  $t2, $t2, $t3
0x00400028:          j    loop
0x0040002C:  end:    move $a0, $t2
0x00400030:          jal  print_int
```

3.4. Donnez la table des symboles une fois que l’outil Assembleur a effectué toutes ses passes

4. Adressage MIPS

Il existe plusieurs modes d'adressage pour accéder à la mémoire dans MIPS :

- **Registre** : l'adresse est contenue dans le registre. e.g. « `jr $ra` »
- **Indexé** : l'opérande est une adresse de base contenu d'un registre à laquelle est ajouté une valeur de déplacement. e.g. « `lw $a1, 0($s0)` » , « `sb $t1, 3($a0)` » .
- **Indexé sur le PC** : Utilise le PC (en fait le PC actuel plus quatre) et ajoute l'immédiat de l'instruction (multiplié par 4) pour créer une adresse. Ce mode est utilisé par des instructions comme `beq`, `bne`. L'étiquette est remplacée par un immédiat qui indique le nombre d'instructions pour le branchement.
- **Pseudo-direct** : adresse obtenue par concaténation des quatre bits de poids fort du PC et les 26 bits [25:0] de l'instruction avec des bits LSB implicites de 00 (pouvez-vous dire pourquoi deux bits implicites de 00 ?). Ce mode est utilisé pas les instructions de type J.

4.1. Vous devez sauter à une adresse de $2^{28} + 4$ octets de plus que le PC actuel. Comment faire ? Pour cette question, l'adresse de destination exacte est supposée connue au moment de la compilation. (Indication : vous avez besoin de plusieurs instructions)

4.2. Vous devez maintenant sauter à une instruction $2^{17} + 4$ octets plus élevée que le PC actuel lorsque `$t0` vaut `0`. Vous pouvez supposer pour cette question que le saut ne s'effectue pas à une adresse appartenant un nouveau bloc de 2^{28} octets. Donnez les instructions MIPS ?

4.3. Soit les instructions MIPS suivantes (et leurs adresses associées), remplissez les champs vides pour les instructions indiquées (vous aurez besoin de votre fiche mips!)

```
0x002cff00:  loop:  addu $t0, $t0, $t0   | 0  |    |    |    |    |    |
0x002cff04:         jal foo              | 3  |                        |
0x002cff08:         bne $t0, $0, loop    | 5  | 8  |    |             |
      ⋮                        ⋮
0x00300004:  foo:   jr $ra               # $ra =
```