Conception et Exploitation des Processeurs

Ensimag 1A 2012–2013

Durée : 2h

Consignes générales :

Le barème donné est indicatif. Les exercices sont indépendants et peuvent être traités dans le désordre.

Les documents sont interdits, **sauf une feuille A4 manuscrite recto-verso**. Photocopies et documents imprimés interdits. Les appareils électroniques (*e.g.* ordinateurs, téléphones, clés USB, *etc.*) sont interdits.

Le sous-répertoire `docs` contient des versions PDF des pages wiki du cours ainsi que des documentations pouvant vous être utiles (dont la PC et la PO de départ du projet).

Dans tous les exercices, on demande de traduire **systématiquement** du code C en assembleur, comme le ferait un compilateur. Vous ne devez donc pas chercher à optimiser le code écrit et **vous devez placer et lire systématiquement les variables locales dans la pile d'exécution comme vu en TP**, sauf indication contraire dans les questions. Vous ferez particulièrement attention aux **types des données manipulées**, c'est à dire leurs tailles et leurs signes.

Pour chaque ligne de C traduite, vous recopierez en commentaire la ligne en question avant d'implanter les instructions assembleur correspondantes. Vous indiquerez aussi la position par rapport au registre `%rbp` de chaque variable locale, ainsi que les registres contenant les paramètres au début des fonctions implantées. Tous les commentaires additionnels sont les bienvenus.

A la fin de l'épreuve, vous devez fermer proprement votre session en cliquant sur

l'icône « Sauvegarder et terminer l'examen ». **Une fois déconnecté, vous ne pourrez plus vous reconnecter et votre code sera rendu automatiquement**. Attention, vous ne devez surtout pas vous déconnecter via le menu Ubuntu, au risque de perdre votre travail. On recommande de **sauvegarder régulièrement votre travail** grâce à l'icône « Sauvegarder l'examen sans quitter » présente sur le bureau.

Tout le travail demandé est à rendre dans les fichiers `projet.txt` et `fct_exo3.s` : le correcteur ne regardera pas les autres fichiers (vous avez le droit de les modifier pour vos tests).

Pour compiler vos programmes, vous utiliserez le `Makefile` fourni. Pour chaque exercice, vous pouvez compiler les sources en tapant simplement dans un terminal la commande `make exo#` (où `#` est le numéro de l'exercice, par exemple `make exo3`). Si vous voulez « nettoyer » le répertoire pour tout recompiler à partir de zéro, il suffit de taper la commande `make clean`. L'utilisation de `gdb` et `valgrind` est très fortement recommandée pour la mise au point de vos programmes.

Partie « Conception de Processeur »

Ex. 1 : Exercice préliminaire *(1 pt)*

Dans cet exercice particulièrement difficile, on vous demande de compléter le fichier `projet.txt` avec vos nom, prénom et numéro de groupe.

Ex. 2 : Questions sur le projet *(7 pts)*

Le fichier `projet.txt` contient le code VHDL des états de base `S_Init`, `S_Fetch_Wait`, `S_Fetch` et le début de `S_Decode`. On vous demande de compléter ce code (y compris les états fournis si nécessaire) pour implanter les instructions demandées ci-dessous.
Vous indiquerez en commentaire la description RTL (*Register Transfer Level*) des commandes que vous écrirez (comme on l'a déjà fait par exemple pour l'état `S_Fetch` en écrivant `IR := Memoire(PC)`).
Vous devez écrire toutes les commandes nécessaires pour implanter complètement ces instructions et revenir à l'état `S_Fetch`. Vous ajouterez tous les états intermédiaires qui vous semblent nécessaires.
On considère dans tout cet exercice qu'on ne gère ni les interruptions, ni les extensions, ni aucune autre instructions que celles demandées.

Question 1 Implanter l'instruction LW dont on rappelle la description :
- action : lecture d'un mot de la mémoire ;
- syntaxe : `lw $rt, imm($rs)` ;
- description : l'adresse de chargement est la somme de la valeur immédiate sur 16 bits, avec extension de signe, et du contenu du registre `$rs`. Le contenu de cette adresse est placé dans le registre `$rt` ;
- opération : $rt \leftarrow mem[imm + rs]$;
- format I.

Note : on ne vous demande pas de gérer les exceptions en cas d'adresse incorrecte, etc.

Partie « Exploitation des Processeurs »

Conseil : ne restez pas bloqué sur une question ! Chaque fonction à écrire en assembleur est constituée d'un ensemble de lignes de C que vous devez traduire le plus littéralement possible : certaines lignes sont faciles, d'autres difficiles. Il est possible de valider des points en écrivant les lignes faciles, même si la fonction ne fonctionne pas totalement.

Ex. 3 : Manipulations de tableaux et listes chainées *(12 pts)*

On travaille dans cet exercice sur l'algorithme de placement de pivot que vous avez déjà étudié en algorithmique au premier semestre. On rappelle le principe de cet algorithme qui travaille sur un tableau d'entiers :

- on prend le dernier élément du tableau initial, qu'on appelle pivot ;
- on parcours le tableau initial `tab`, et on range les éléments de telle-façon qu'à la fin de la fonction :
  - tous les éléments strictement inférieurs à la valeur du pivot soient rangés dans le sous-tableau `tab[0 .. prem-1]` ;
  - tous les éléments égaux à la valeur du pivot soient rangés dans le sous-tableau `tab[prem..der]` ;
  - tous les éléments strictement supérieurs à la valeur du pivot soient rangés dans le sous-tableau `tab[der+1 .. taille-1]`.
- où `prem` et `der` sont des indices du tableau et `taille` son nombre d'éléments.

Attention : on n'impose aucune contrainte sur l'ordre des éléments dans les sous-tableaux.

Par exemple, si initialement `tab = [2, -1, 4, 3, -2, 0, 1]`, alors à la fin

- `tab` *pourra contenir* `[0, -2, -1, 1, 3, 4, 2]`,
- `prem` vaudra forcément 3 et
- `der` vaudra aussi forcément 3.

Le fichier `exo3.c` contient le programme principal et des fonctions utiles pour tester votre code. On ne vous demande pas de rajouter des tests, mais vous pouvez bien sûr le faire si cela vous aide à mettre au point vos fonctions.

Le fichier `fct_exo3.s` contient le squelette des fonctions à implanter en assembleur. On donne à chaque fois le code C à traduire : on vous demande une traduction la plus littérale possible, sans chercher à optimiser le code que vous écrivez. Vous indiquerez systématiquement la ligne de C traduite en commentaire avant la suite d'instructions

correspondante.

Question 1 Compléter la fonction `echanger` : cette fonction prend en paramètre un tableau `tab` d'entiers signés sur 64 bits ainsi que deux indices `i` et `j` représentés par des entiers naturels sur 64 bits.
Cette fonction doit échanger le contenu des cases `tab[i]` et `tab[j]` en passant par une variable locale `tmp` (on rappelle que vous **devez** placer cette variable locale dans la pile).

Question 2 Compléter la fonction de placement de pivot `placer_pivot` : cette fonction prend en paramètre un tableau `tab` d'entiers signés sur 64 bits ainsi qu'un entier `cour` qui représente l'indice initial du pivot (on utilisera ce paramètre comme variable dans la fonction), et aussi deux **pointeurs** vers des entiers sur 64 bits, appelés `prem` et `der`, qui serviront à délimiter les 3 zones à la fin de l'exécution de la fonction.
Le programme C passe à cette fonction un tableau rempli d'entiers signés tirés aléatoirement, ainsi que l'indice du pivot (dans le code C fourni, on choisi le dernier élément du tableau). La fonction `placer_pivot` parcourt le tableau et en fonction de la valeur de la case courante, fait des échanges et met à jour les indices `prem` et `der` pour construire les 3 zones (zone des inférieurs, zone des égaux et zone des supérieurs). On note bien que `prem` et `der` sont des pointeurs vers des entiers naturels : en effet, on doit les renvoyer comme résultats à la fonction C appelante.
Là encore, vous devez placer les variables locales dans la pile.

Question 3 Compléter enfin la fonction `tab_vers_liste` : cette fonction prend en paramètre un tableau `tab` d'entiers signés sur 64 bits, la `taille` du tableau représentée par un entier naturel sur 64 bits, ainsi qu'un pointeur vers la `liste` résultat (c'est à dire un pointeur vers un pointeurs vers une cellule).
Cette fonction parcours le tableau pour créer la liste chainée correspondante. Le type des cellule est défini (dans le fichier C) simplement comme une structure contenant la valeur de la cellule (entier signé sur 64 bits) et un pointeur vers la cellule suivante.
La fonction doit allouer des nouvelles cellules en appelant `malloc`. On rappelle que `sizeof` est un opérateur interne au langage C qui renvoie la taille en octets de son paramètre : on ne peut pas l'appeler en assembleur, donc on passera directement à `malloc` le bon nombre d'octets à allouer : ici $8 + 8 = 16$.
Comme pour les autres questions, vous devez placer les variables locales dans la pile. N'oubliez pas de plus que la fonction `malloc` peut détruire les registres servant à passer des paramètres : vous devez donc sauvegarder ceux qui contiennent des valeurs importantes avant d'appeler `malloc`, et les restaurer ensuite.