Selim K. & Alex W.F.

Sujet

Projet assembleur MIPS : Calculatrice

Le but de ce projet est d'écrire un programme qui réalise les fonctions d'une calculatrice. A partir d'une chaîne de caractère entrée au clavier. On pourra gérer les chaînes de caractères, celles représentant un calcul en notation infixe, en notation préfixe, ou en notation polonaise inverse.

Quelques définitions :

La notation infixe est la notation couramment utilisé dans les formules arithmétiques. Elle est caractérisée par la mise en place d'opérateurs entre opérandes, comme le signe plus dans "2 + 2".

La notation préfixe, aussi connu comme notation polonaise, est une forme de notation arithmétique. Sa caractéristique principale est qu'il place les opérateurs à la gauche de leurs opérandes. On remarque aussi que la notation de l'opération peut avoir une syntaxe sans parenthèses.
Exemple : "* (+ 2 3) 1" équivalent à "*+ 2 3 1" équivalent aussi à : "(2+3)*1 "

La notation polonaise inverse correspond simplement à la notation préfixe avec les opérandes et opérateurs inversés.
Exemple : "(2 3 +) 1* " équivalent à "2 3+1 *" équivalent aussi à : "(2+3)*1 "

Analyse du travail

Trouver un moyen d'arranger le programme afin de pouvoir effectuer le calcul de toutes les expressions et leur notation ==> deux tableaux un corresponds aux opérateur, l'autre aux opérandes

Le calcul a effectué sera saisi par clavier, donc c'est une chaîne (chaîne_input), et pour effectuer les calcul ils nous faut des valeur numérique. Il faudra donc une fonction pour convertir les opérandes initialement de type caractères en entier ou flottant.

Nous avons donc choisis d'utiliser deux tableau, le premier qui contiendra les valeurs numériques, tab_int, et l'autre les indices d'opération en caractères, tab_op. Puis nous effectuons les calculs donnés par l'indice d'opération du tableau tab_op et par rapport au position de chaque opérateur dans le tableau tab_int et leur ordre de priorité.

Après le calcul, les deux tableaux doivent être rafraîchis, c'est a dire le tableau d'opérandes doit effacer les opérandes utiliser pour le calcul et ajouter le résultat dans le bon emplacement du tableau, de même dans le tableau d'opération, on supprimera l'opération.

A propos des parenthèses, l'exécution des étapes du programme de calcul commence tout d'abord par une recherche de parenthèse, on trouve d'abord la toute dernière parenthèse « (» que l'utilisateur a saisi, le contenu entre cette parenthèse et la parenthèse fermante va correspondre au tout premier calcul a effectuer, après le calcul, le programme remplacera dans la chaîne_input, toute la parenthèse cible par le résultat puis , le programme recommencera à effectuer la recherche de parenthèses, et les étapes précédentes jusqu'à il ne reste plus de parenthèse dans la chaîne_input

Quand il ne reste plus du tout de parenthèse le programme effectuera une dernière fois des calcul simple et retournera sur l'écran le résultat final .

Enfin , comme l'utilisateur peut mettre ce qu'il veut en entrée, il faut vérifier l'entrée pour éviter que le programme ne plante, pour cela il faut faire un partie gestion des erreurs, ou on traite les problèmes dut au mauvaises utilisations.

Description des fonctions les plus intéressantes réalisées

fonction clean tableaux

Cette fonction est utilisée pour réinitialiser les tableaux tab_int, tab_op, tab_bis et tab_bis2 a leur valeur initiale, c'est a dire ce qui sont écrit dans le .data.

fonction pour rafraîchir les tableaux

Cette fonction fera les remplacements à effectuer dans le tableau opérande tab_int, et aussi enlèvera l'opération concerné du tableau d'indice d'opération, tab_int, cette fonction prendra en argument le résultat du calcul effectuer et l'indice d'opération du calcul.

fonction atoi

Cette fonction effectuera la conversion d'un nombre de type caractère en un nombre entier qui correspond à la valeur du caractère.

fonction_tab_int {fonction atoi }

Cette fonction repère les opérandes entrée par l'utilisateur, puis les converti en nombre entier en utilisant la fonction atoi, et enfin elle charge ces nombres dans le tableau dédié aux opérandes dans l'ordre, tab_int.

interpréteur_commande { fonction_tab_int }

Cette fonction lit le tableau de caractère et exécute les opérations en fonction de leur priorité, elle se charge aussi de reconnaître les deux opérandes mis en jeu, elle réalise le calcul grâce a ses sous

fonctions (addition,soustraction,....,conversion binaire,conversion hexadécimale.)

fonction chercher opération bis

Elle repère les indices d'opération et les charge dans le tableau d'indice opération dans l'ordre.

fonction int to alpha

Cette fonction réalise la conversion d'un chiffre entre 0 et 9 en le caractère qui le corresponds, la raison pour laquelle cette fonction est récursive par jump, c'est parce que la fonction ne peut pas faire la conversion en caractère d'un nombre composé de plus d'un chiffre en une seule fois, ainsi elle effectue la conversion de chaque chiffre du nombre de droite vers la gauche, en store dans un tableau tab_bis, afin de ranger les chiffre dans le bon ordre, la fonction effectuera une nouvelle boucle qui place le contenu de tab_bis dans l'ordre dans un nouveau tableau tab_bis2, la fonction termine en retournant le tableau tab_bis2

fonction pour rafraîchir chaîne {fonction int to alpha, interpreteur_commande }

Cette fonction vérifie d'abord si la chaîne est correctement entrée par l'utilisateur, en cherchant la parenthèse fermé qui est associée à la parenthèse d'ouverture « (»(pointée par le registre $a0). Puis on supprime cette parenthèse ouverte et le contenu entre ces parenthèses en les décalant jusqu'à hors du tableau chaine_inpout (utilisation d'un double while), puis il effectue les calcul avec la fonction interpreteur_commande, et retourne le résultat dans $v1.

Après avoir récupéré le résultat, la fonction rafraîchir_chaîne fait appel à la fonction itoa, pour convertir le résultat en une chaîne de caractères.

Ensuite, elle ajoute cette chaine au bon endroit de la chaine_input, pour effectuer cela, elle détermine d'abord le nombre de chiffre qui compose le resultat du calcul, et le nombre de cases entre d'adresse contenu par $a0 initaliment et la fin du tableau chaîne_input.

Après avoir déterminer ces données, la fonction effectue un décalage du tableau chaine_input, jusqu'à avoir assez d'espaces pour ajouter la chaîne de résultat contenu dans tab_bis2. Ainsi la nouvelle chaine_input est crée.

fonction_ parenthèse { fct clean tableaux, fct chercher operation bis, fonction pour rafraichir chaine}

Cette fonction est très important selon nous, car elle est composé de quatre autres fonctions cités ci dessus, et de plus cette fonction est une fonction récursive. La récursivité de cette fonction est effectuée pour la recherche de la dernière parenthèse ouverte de la chaine_input, car le contenu entre cette parenthèse et la parenthèse fermée associée correspond au tout premier calcul à effectuer. Ainsi la fonction effectuera la recherche et déterminera l'adresse du tableau qui contient cette parenthèse d'ouverture.

Après avoir déterminer cette adresse, on effectue les étapes de stockages des données, en appelant les fonctions fct_find_op_bis, et fonction_tab_int, le stockage de donnée s'arrêtera au moment où on arrive sur la parenthèse fermée «) », après avoir effectué les stockages, elle fait appel à la fonction fct_refresh_chaine pour rafraîchir la chaine_input c'est à dire a enlever les parenthèses déterminer auparavant et remplacer le contenu par le résultat. Enfin après ces étapes, la fonction finie par

retourner sur elle même(récursivité) pour effectuer une nouvelle fois la recherche de parenthèse, si elle en trouve d'autre , elle répétera les étapes décrit ci-dessus, sinon elle se termine et on retourne le pointeur pc sur les instructions de la fonction main.
Remarque : *Vers la fin de cette dernière fonction, afin de déterminer si la chaine_input n'est seulement qu'un nombre sans opération demandé, on effectue une recherche d'indice d'opération dans la chaine_input, s'il n'y en a pas, alors on renvois directement la chaine_input comme résultat de calcul.*

Une liste des difficultés rencontrées dans la gestion de ce devoir

1 - La première difficulté et l'un des problèmes qu'on a pas pu gérer dans ce devoir est l'utilisation de flottant pour la division. (nous avons commencé par travailler avec des entiers , et essayant réarranger on a été confronté a des difficultés , de plus la fonction atoi a été déjà faite pour les entiers ainsi que tout les calculs.)

2 - La deuxième difficulté est la modularité , comme nous nous avons reparti chacun des fonctions différentes à écrire, et que nous faisions chacun de son coté ces fonctions, ainsi souvent lorsqu'il faut intégrer un bout de notre code dans le fichier code du binôme, des problèmes surviennent et un debeugage s'impose : cela nous a causé des pertes de temps. (sans la convention d'appel des fonctions, cela aurait été encore plus problématique.)

3 - La gestion des piles pour chaque fonction était un gros problème, car souvent le pointeur d'adresse pointe dans la mauvaise adresse et donc charge la mauvaise contenu de $sp dans le registre demandé. Mais cela a été finalement réglé par des approfondissement de connaissance.

4 – Nous avons eu des difficultés pour écrire la fonction itoa. En effet, par rapport à la fonction atoi, nous n'avons jamais étudier la fonction itoa de près en cours. Ainsi il a été pas si facile de trouver une méthode de conversion d'un entier en caractère, pour cela nous avons recherché sur internet des fonctions itoa chacune plus ou moins différentes pour comprendre leur fonctionnement avant d'écrire la notre.

Une éventuelle liste de remarques sur votre pratique de la programmation assembleur.

- Au début de la programmation, certaine fonctions crée par Alex, ont été d'abord écrites en C, puis en MIPS, mais comme les fonctions devenaient de plus en plus longue, on a donc décidé d’écrire directement en MIPS sans passer par le C, ainsi on n'avons pas de codes sources C.

- L'utilisation de commentaire, délimiter les parties, mettre en évidence les différentes fonctions est obligatoire pour pouvoir travailler en binôme.

- L’utilisation des macros pour simplifier certaines commandes répétitifs tel que l'affichage pour

vérification. ainsi on test le programme au fur et à mesure sans utiliser des lignes supplémentaire pour tester

- Notre programme est loin d'être très optimisé, on fait plein de parcours sur la chaîne d'entrée pour différent traitement, nous pourrions faire moins de parcours, en combinant le traitement dans un nombre minime de passage de boucle. Par contre cela nous demandera plus de complexité dans les méthodes utilisé.

- La sécurité est aussi importante, le programme plante facilement , si l'entrée n'est pas traitable, un travail de vérification du bon déroulement du programme est obligatoire pour assurer la robustesse du programme.

- Nous avons décidé d'utiliser le caractère "f" pour « fin » du tableau dans le tableau d'indice d'opération tab_op.

- Dans le cas général, on peut saisir des opération de calcul avec des espaces entre opération, mais il existe des cas assez rares où le programme plante et provoque une boucle infinie.

- Nous avons pas effectuer la vérification pour la conversion hexadécimale et binaire, car il est plus facile pour nous de prendre des entiers comme entrée.