Assembleur

TP 1

# Objectif :

L’objectif de ce TP est de réviser les connaissances en C et de se familiariser avec le jeu d’instruction x86 32bits.

# Prérequis :

* Connaissances en C

# A savoir :

* Quand mon programme assembleur se lance, le code assembleur est stocké dans des segment de la ram et ainsi, le programm est accessible pour le proc.

En suite, la stack c’est le context d’execution du program, sur la ou il en est

# Étape 1 : Révision de C

|  |
| --- |
| **Quels sont les types de base du langage C ?** |
| [Int, char, int, long, void, short, float, double] (chacun en version signed ou unsigned), & (address memoire) |

|  |
| --- |
| **Combien d’octets sont utilisés pour encoder les types sur une architecture 32 bits ?** |
| De 1 à 8 octets |
| **Quelle est la taille d’un pointeur ?** |
| **4 octet** |

|  |
| --- |
| **Quel est le bit utilisé pour stocker le signe d’un entier ?** |
| **Le bit de poids fort (1…. en big endian, ….1 en little endian)** |

|  |
| --- |
| **Quel est la différence entre un ‘’signed int’’ et ‘’unsigned int’’ ?** |
| **Un signed int peut avoir une valur negative, un unsigned int ne peut pas. Si le signed int est compris entre les valeur (** -32 767 et 32 767) , unsigned int est comprit entre 0 et 65534 |

|  |
| --- |
| **Par quoi se termine une chaine de caractère en C ?** |
| \0 |

|  |
| --- |
| **Quelle est la fonction qui permet de réserver de la mémoire dynamiquement ?** |
| Malloc ( on peut utiliser des api de l’os pour faire du specific aussi) |

|  |
| --- |
| **Complétez le tableau suivants** |
| |  |  |  | | --- | --- | --- | | **Fonctions en C** | **Définition** | **Valeur de retour (le cas échéant)** | | SCANF | Lis sur l’entré stangard |  | | FGETC | Lis le prochain charatcer d’un buffer | Return readed value as unsigned char | | STRCPY | Copy la chaine de charactere src dans dest | void | | STRCAT | Concatene deux string | Un pointeur sur char egale a la concatenation des deux string | | STRCMP | Compare des chaine de charatere jusqu’au character \0 | 0 if equal  -x s1 is less than s2  +x s2 is less than s1 | | STRLEN | Renvoi la longueur d’une chaine de characeter | Renvoi un int du nombre de char avant \0 | | MEMCPY | Copy les byte de src dans dest pour une longueur donné | void | | MEMCMP | Compare des espace memoire | La difference entre les bytes de char | | ATOI | Ascii to int | Integer of asci value | | RAND | Renvoi une valeur random | Random int | | SRAND | Init iv of random | void | |

|  |
| --- |
| **Dessinez la représentation mémoire à l’adresse 0x42100 du tableau suivant :**  char tab[20] ;  **On initialise le tableau avec les éléments suivants ‘’abcdefghijklmnopqrst’’**  **Si chaque case mémoire représente un octet ?** |
| |  |  |  | | --- | --- | --- | | Adresse | Code en C | Valeur | | **0x42100** | tab[0] | **a** | | **0x42101** | tab[1] | **b** | | **0x42102** | tab[2] | **c** | | **0x42103** | tab[3] | **d** | | **0x42104** | tab[4] | **e** | | **0x42105** | tab[5] | **f** | | **0x42106** | tab[6] | **g** | |
| **Dessinez la représentation mémoire à l’adresse 0x42100 du même tableau mais en changeant la taille de la case mémoire à 4 octets :**  **Les processeur x86 utilise le little-endian[[1]](#footnote-2).** |
| 0x00000061 |

# Étape 2 : Notions importantes

|  |
| --- |
| **Quels sont les registres généraux en Assembleur x86 32 bits ?** |
| **EAX EBX ECX EDX** |
| **Expliquez avec vos mots le fonctionnement du registre ESP :** |
| **Le registre ESP pointe sur l’element au sommet de la stack** |
| **Expliquez avec vos mots le fonctionnement du registres EBP :** |
| **Le registre EBP pointe sur le sommet de la base de la frame courante. La frame courante est au somut de la stack, vue qu’on travail dessus, c’est au dessus de la stack** |
| **Quelle est la taille du registre AL ?** |
| **AL est de 1 octet et représente les bit de poids faible de AX** |

|  |
| --- |
| **De quel type de liste la pile est-elle ?** |
| FILO (first in last out) == LIFO (last in first out) |

|  |
| --- |
| **Dans quelle partie de la mémoire est stockée les variables locales d’une fonction ?** |
| **Dans la stack. La stack contient des vlaur qui sont stocké de 4 en 4, la stack est donc une pile d’espace memoire, un long tableau de INT, qui contient.**  **On y trouve les argument, les variable local, et aussi le resitre de l’ebp parent** |

|  |
| --- |
| **Expliquez avec vos mots pourquoi on a besoin du tas ?** |
| **Pour acceder a des espace mémoire non ordoné et y stocker nos valeur, on garde une référence grace au pointeur. Pour toute allocation dynamique** |

|  |
| --- |
| **Qu’est qu’une frame pour un fonction assembleur ?** |
| **Une partie de la stack qui appartient a un contexte d’execution ‘un fonciton** |

|  |
| --- |
| **Quelles sont les instructions qui permettent d’ajouter ou de retirer des valeurs de la pile ?** |
| **PUSH <registre>**  **POP**  int var ; = push Ecx  char tab[20] ; = sub esp,20 ; // pour deplacer l’address de esp de 20 address et ainsi reserver les case memoire entre esp initial et l’esp modifier (esp soustrait de 20) |

# Étape 2 : Pratiquer l’assembleur

|  |
| --- |
| **Ecrivez en assembleur l’opération : 4 + 9 :** |
| **mov eax, 4**  **add eax, 9** |

|  |
| --- |
| **Ecrivez en assembleur l’opération : 5 - 3 :** |
| **mov eax 5**  **sub eax, 3** |

|  |
| --- |
| **Ecrivez en assembleur l’opération : 2 \* 10 :** |
| **mov eax 2**  **mov edx 10**  **mul edx // multiplie eax et edx et stock dans eax** |

|  |
| --- |
| **Quel est le résultat de l’instruction « xor eax, eax »** |
| **Le registre eax met 0x0000 car on xor deux valeur egale, elle s’annule.** |

|  |
| --- |
| **Quelle est le résultat de l’instruction « div BL » si AX vaut 10 et BL vaut 2 ?** |
| **5** |
| **Dans quel registre se trouve le résultat ?** |
| **EDX** |

|  |
| --- |
| **Commet se traduit les écritures i++ et i-- en assembleur ?** |
| Mov EAX i  INC EAX  et  Mov EAX i  DEC EAX |

|  |
| --- |
| **Comment stocker la valeur 0x1337 dans le registre EDX ?** |
| MOV EDX, 1337h |

|  |
| --- |
| **Quelle est la signification de l’instruction « MOV ESI, offset ‘’ESIEA’’ »?** |
| **Prend l’address memoi de la variable ESIEA dans ESI** |
| **Quelle est la représentation hexadécimal de la chaine ESIEA et esiea ?** |
| **En little endian ESIEA = 0x4553494541, esiea = 0x4553494541 (soustraire en fonction de la table asci pour les minuscule)** |
| **Quelle est la taille en mémoire de cette chaine de caractère ?** |
| **6 octets si on compte le \0 de fin de chaine** |
| **Ecrivez en assembleur, l’opération pour transformer la lettre R en r ?** |
| EAX 52  ADD 20h |

|  |
| --- |
| **Comment stocker le contenu du registre ESI à l’adresse mémoire 0x1000 ?** |
| **MOV PTR[1000h] ESI** |

|  |
| --- |
| **Expliquez l’instruction : « MOV BYTE PTR [edi+ecx\*4], 0xAF » ?** |
| Deplace la valeur hexadecimal 0xAF à la 4 eme address suivante l’address stocké dans le registre EDI |

|  |
| --- |
| **Expliquez cette instruction «MOV DWORD PTR[eax], 53494541h » ?** |
| Les crocher c’est qu’on manipule des addres, PTR c’est qu’on  DWORD c’est qu’on manipule 4 octet  **Donc la, il va a la case mémoire correspondant a l’address contenue dans eax, et y met la valeur address** |

|  |
| --- |
| **Expliquez la différence entre MOV et LEA ?** |
| MOV deplace des valeur d’un registre a l’autre ou dans une addresse, LEA charge une addresse effective |

|  |
| --- |
| **Quelle est l’instruction qui permet d’appeler des fonctions ?** |
| **CALL <address>** |
| **Comment fonctionne-t-elle ?** |
| L'instruction CALL en assembleur est utilisée pour effectuer un appel de fonction. Lorsque CALL est rencontré, l'adresse de l'instruction suivant l'instruction CALL est sauvegardée sur la pile, puis le programme est dirigé vers la nouvelle adresse spécifiée. Cela permet d'exécuter une séquence de code spécifique (fonction) et de revenir à l'emplacement d'origine après l'exécution de la fonction. |
| **Comment s’effectue le passage des arguments ?** |
| **Les argument sont placer dans un des registres**  **cdecl (parent nettoie la frame enfant), stdcall (enfant nettoie sa frame), fastcall** |
| **En fonction de EBP, à quel offset se trouvera le 1er argument d’une fonction ?** |
| -8  \_\_\_\_\_\_  ebp -1 (will make -4 cause int)  \_\_\_\_\_\_  eip – 1 (will make -4 cause int)  \_\_\_\_\_\_  arg 0 (will has the first arg)  \_\_\_\_\_\_  arg 1  \_\_\_\_\_\_ |
| **En fonction de EBP, à quel offset se trouvera le 2e argument d’une fonction ?** |
| 1 |
| **En fonction de EBP, à quel offset se trouvera le 3e argument d’une fonction ?** |
| 2 |

|  |
| --- |
| **Expliquez la convention d’appel « cdecl » ?** |
|  |
| **Expliquez la convention d’appel « stdcall » ?** |
| passes les arguments right-to-left, et return la valeur dans EAX. |
| **Dans quel registre, on passera le retour d’une fonction ?** |
| **RET pour retourner la valeur de eax** |

|  |
| --- |
| **Quelles instructions représentent le if … else en C ?** |
| CMP pour if (comparaison) JNZ pour else |
| **A quoi correspond JZ ou JE ?** |
| **JZ = Jump if zero , JE = jump if equal** |
| **A quoi correspond JNZ ou JNE ?** |
| **JNZ = Jump if not zero , JNE = jump if not equal** |
| **A quoi correspond JB ou JNA?** |
| **JB = jump if value is below, JNAE = Jump if not above ?** |
| **A quoi correspond JA ou JNAE ?** |
| JA = JUMP if Above, Jump if nor above or equald |

1. <https://fr.wiktionary.org/wiki/little-endian> [↑](#footnote-ref-2)