TP1

Introduction à la programmation en assembleur sur CRAPS

Objectifs

1. Découverte du microprocesseur CRAPS : fonctionnement, jeux d'instructions.
2. Ecriture et test de programmes simples en langage assembleur.

1- Initiation au microprocesseur CRAPS

CRAPS est un petit microprocesseur qui a été spécifié à l'N7, et que vous aurez l'occasion d'implanter durant les TP d'architecture des ordinateurs II du second semestre.

C'est un processeur basé sur une architecture RISC (Reduced Instruction Set Computer), et très largement inspiré du processeur SPARC (d'où le nom CRAPS !), l'un des premiers processeurs modernes basés sur cette architecture.

A- Modèle de fonctionnement

CRAPS repose sur le modèle Von Neumann, et fonctionne (exécute des programmes) selon le schéma séquentiel classique.
Un programme est un ensemble d'instructions machine stockées en mémoire, et dont l'exécution se fait de deux façons :

- exécution de l'instruction courante, puis passage séquentiel à l'instruction suivante,
- ou, pour les instructions de branchement : saut à l'instruction dont l'adresse est indiquée si la condition est vérifiée.

Exemple : soit la séquence suivante :

```
// op1 et op2 initialisés respectivement à 5 et à 0
Label :  add    op1, op2, op2        // op2 ← op1 + op2
         subcc  op1, 1, op1          // op1 ← op1 – 1 (on verra le rôle du cc plus loin)
         bne    Label                // b: branchement si le résultat de l'instruction sub est
         ...                         // différent (ne : not equal) de 0 à l'instruction portant l'étiquette 'Label' (add)
```

L'exécution de cette séquence se fait donc de la manière suivate :

- exécution de l'instruction add, avec comme résultat op2 = 5 (5+0)
- exécution de l'instruction subcc, avec comme résultat op1 = 4 (5-1)
- branchement à Label (instruction add), car résultat du subcc différent de 0
- …
- lors des 3 passages suivants op2 et op1 vont respectivement prendre les valeurs (9, 12, 14) et (3, 2, 1)
- à la fin du cinquième passage l'instruction subcc donnera un résultat égal à 0 dans op1, ce qui aura pour effet de ne pas activer le branchement (car 'ne' est faux) et de passer à l'instruction qui suit le 'bne'.

CRAPS dispose d'une trentaine d'instructions machine simples, que l'on découvrira plus loin. Une instruction machine peut avoir de 1 à 3 arguments et a un code spécifique de 32 bits.

Pour être exécuté, un programme a besoin de ressources pour stocker les instructions, les opérandes, les variables, etc. ; ressources que l'on va découvrir progressivement.

B- Ressources principales

Registres : Ce sont des mots mémoires fixes à accès direct très rapide, et qui servent à contenir les opérandes des instructions ou des informations nécessaires au fonctionnement de processeur (adresse de l'instruction courante, code de l'instruction courante, etc.)
CRAPS dispose d'une vingtaine de registres 32 bits, répartis en deux groupes :

- R0 (toujours =0), R1, R2, …, R25 : registres destinés à l'usage du programmeur et dont le rôle principal est de contenir des opérandes : valeurs numériques, adresses mémoires, etc.
- des registres réservés au fonctionnement du processeur :
  - PC (R30), Program Counter, est le registre qui contient exclusivement l'adresse de l'instruction courante,
  - IR (R31), Instruction Register : est le registre qui contient exclusivement le code de l'instruction courante,
  - d'autres registres : SP, RET, … seront vus plus tard.

Mémoire **:** CRAPS est doté d'une mémoire RAM (Random Acces Memory) organisée par mots de 32 bits. Chaque mot est accessible via une adresse (le premier mot à l'adresse 0). Cette mémoire est destinée à contenir les programmes à exécuter et leurs données.

Illustration ***:*** soit la séquence d'instructions qui effectue l'addition entre 2 variables VA et VB, et enregistre le résultat dans une troisième variable VC.

- Il faut noter que les instructions et les variables sont implantées en mémoire à des adresses plus ou moins choisies**.**
- Les labels (étiquettes) sont des noms symboliques que l'on associe à des adresses pour référencer facilement des instructions ou des variables sans avoir à connaître l'adresse d'implantation. C'est l'assembleur qui se chargera de la correspondance entre le label et l'adresse associée. Les Labels sont définis en début de ligne et se terminent par :

```
Debut:    ld      [VA], %r1       // lit le mot mémoire (load) situé à l'adresse VA et le stocke
                                  // dans le registre R1
          ld      [VB], %r2       // lit le mot mémoire d'adresse VA et le stocke dans R2
          add     %r1, %r2, %r2   // r2 ← r1 + r2, l'addition ne se fait que dans des registres
          st      %r2, [VC]       // stocke (store) le contenu de R2 à l'adresse VC
Fin:      ba      Fin             // boucle infinie, simulant l'arrêt du programme
VA:       .word   3               // associe le label VA à l'adresse courante, et initialise ce mot
                                  // mémoire à 3
VB:       .word   4               // .word est une directive interprétée lors de l'assemblage et
VC:       .word   0               // du chargement du programme en mémoire
```

Par défaut, ce programme est installé à l'adresse 0, mais on peut préciser une adresse différente avec la directive ***.org***, en mettant par exemple, avant la première instruction :

```
          .org    0x10            // pour les nombres : 10 est une valeur décimale, 0b10 est
```

une valeur binaire, 0x10 est une valeur hexadécimale (égale à 16 en base 10 : 1x16 + 0)

A noter que les variables VA, VB et VC sont installées en mémoire juste après la dernière instruction 'ba Fin' ; mais que l'on peut choisir de mettre à partir d'une adresse bien choisie, en insérant, par exemple, .org 0x100 avant la déclaration de VA. Dans ce cas, il faut veiller à choisir une adresse qui ne risque pas d'être écrasée par les dernières instructions du programme.

C- Jeu d'instructions

Instructions arithmétiques et logiques

Instruction	Effet
add %ri, op2, %rj	rj ← ri+ op2 ; op2 peut être un registre ou une constante sur 13 bits
sub %ri, op2, %rj	rj ← ri - op2
umulcc %ri, op2, %rj	rj ← ri * op2
and %ri, op2, %rj	rj ← ri AND op2 ; ET logique bit à bit
or %ri, op2, %rj	rj ← ri OR op2 ; OU logique bit à bit
xor %ri, op2, %rj	rj ← ri XOR op2 ; OU exclusif bit à bit

Les instructions précédentes, à l'exception de « umulcc », n'affectent pas les indicateurs :

- N (Négatif) : résultat négatif
- Z (Zéro) : résultat égal à 0
- V (oVerflow) : opération ayant engendré un débordement sur le bit de signe, donc résultat susceptible d'être faux
- C (Carry) : opération ayant engendré un débordement (retenue, emprunt) qui indique que le résultat ne tient pas sur 32 bits

Ces indicateurs servent pour vérifier la validité du résultat, et pour gérer les branchements conditionnels que l'on verra plus loin.

Toutes ces instructions, possèdent une version qui affecte les indicateurs et qui porte le même nom avec ''cc" à sa fin : addcc, subcc, andcc, …
Ceci offre plus de flexibilité dans l'utilisation de ces opérations.

Instructions de décalage

Instruction	Effet
sll %ri, op2, %rj	Décalage à gauche : rj ← ri décalé op2 fois, op2 '0' injectés à droite
slr %ri, op2, %rj	Décalage à droite : rj ← ri décalé op2 fois, op2 '0' injectés à gauche

Instruction d'accès mémoire

Lecture : ld [%ri+ opad2], %rj — le mot se trouvant à l'adresse %ri + opad2 est copié dans %rj
Ecriture : st %rj, [%ri + opad2] — le contenu de rj est enregistré à l'adresse %ri + opad2
opad2 peut être un registre ou une constante sur 13 bits

Instructions de branchement

Dans un programme, on a besoin de structures de contrôle de type si condition alors, pour I de 1 à N faire, tant que condition faire, etc.

Ces structures sont implantées en langage assembleur par l'intermédiaire d'instructions de branchement. CRAPS dispose d'une quinzaine d'instructions de branchement dont la forme générale est : bcond adresse (ou label équivalent à une adresse)
et qui fonctionnent selon le schéma suivant :

- si la condition 'cond' est vraie alors il y a branchement à l'instruction se trouvant à l'adresse indiquée en argument
- sinon, il y a passage automatique à l'instruction qui suit l'instruction bcond

Le tableau suivant présente l'ensemble des instructions de branchement de CRAPS.

Instruction	***Opération***		***Instruction***	***Opération***
ba	Branch Always		Bg (ou bgt)	Branch on Greater
beq (be, bz)	Branch on Equal		bge	Branch on Greater or Equal
bne (bnz)	Branch on Not Equal		bl (ou blt)	Branch on Less
bneg (bn)	Branch on Negative		ble	Branch on Less or Equal
bpos (bnn)	Branch on Positive		bgu	Branch on Greater Unsigned
bcs (blu)	Branch on Carry Set		bgeu	" on Greater or Equal, Unsigned
bcc (bgeu)	Branch on Carry Clear		blu	Branch on Less, Unsigned
bvs	Branch on oVerflow Set		bleu	Branch on Less or Equal Unsigned
bvc	Branch on V Clear			

La condition 'cond' (eq, ne, …) fait référence aux indicateurs N, Z, V et C, eux même positionnés en fonction du résultat de l'instruction précédente du type addcc, subcc, andcc, etc.

L'instruction 'cmp op1, op2' permet de réaliser une comparaison du premier opérande op1 par rapport au second op2. Cette instruction est implantée en réalité par 'subcc op1, op2, %r0'.
Et si elle est suivie par une instruction de branchement bcond, la condition 'cond' (par exemple 'lt') représente de façon équivalente op1 'cond' op2 (op1 < op2) ou (op1-op2) 'cond' 0 ((op1-op2) < 0).

Le tableau suivant présente une traduction des principales structures de contrôle en assembleur. Pour simplifier, on considère une condition simple entre deux opérandes op1 et op2.

Si (op1 'cond' op2) alors séquence alors sinon séquence sinon fin si	cmp op1, op2 b'cond' Seq_alors Seq_Sinon: …. ba Fin_si Seq_Alors: …. Fin_Si: ….
Répéter … Jusqu'à (op1 'cond' op2)	Bcle_Repeter: … cmp op1, op2 bNON'cond' Bcle_Repeter

Pour I de 1 à N faire … Fin pour	Bcle_pour: cmp I, N // I dans un registre bgu Fin_Pour … ba Bcle_pour Fin_pour: ….
Tant que (op1 'cond' op2) faire … Fin tant que	Bcle_tant_que: cmp op1, op2 bNON'cond' Fin_tant_que …. ba Bcle_tant_que Fin_tant_que: ...

Instructions synthétiques

Ce sont des instructions définies au niveau du langage assembleur pour faciliter la programmation. Elles reposent sur une ou plusieurs instructions machine. Le registre r0 qui est toujours égal à 0.

Instruction	***Effet***	***implémentation***
clr %ri	met à zéro %ri	orcc %r0, %r0, %ri
tst %ri	teste nullité et signe de %ri	orcc %ri, %r0, %r0
negcc %ri	calcule opposé de %ri	subcc %r0, %ri, %ri
mov %ri,%rj	copie %ri dans %rj	orcc %ri, %r0, %rj
inccc %ri	incrémente %ri	addcc %ri, 1, %ri
deccc %ri	décrémente %ri	subcc %ri, 1, %ri
set val31..0, %ri	copie *val* dans %ri	sethi val31..8, %ri orcc %ri, *val*7..0, %ri
cmp %ri, op2	compare %ri et op2	subcc %ri, op2, %r0

2- Exercices

1- Ecrivez et testez le programme qui calcule le pgcd de deux nombres se trouvant dans les registres %r1 et %r2.

2- Ecrivez et testez le programme qui calcule la somme des N premiers entiers, avec N se trouvant dans le registre %r1.

3- Ecrivez et testez le programme qui calcule la somme des éléments d'un tableau de 10 entiers. Le tableau sera déclaré et initialisé à la fin du programme avec la directive .word qui peut prendre plusieurs valeurs en arguments (.word 3, 6, -9, 5, 2, 11, 67, 54, 32, 55). On considère que le premier élément du tableau a pour indice 0.

4- Ecrivez et testez le programme qui recherche le maximum d'un tableau de 10 entiers et stocke ce maximum et son indice respectivement dans les registres %r3 et %r4

5- Ecrivez et testez le programme qui calcule l'indice de la première occurrence de la valeur 0 dans un tableau de N éléments. Si cette valeur n'existe pas, l'indice calculé est égal à N.