TP2

Processeur « Craps » : instructions de branchement

Les instructions de branchement permettent de se déplacer dans le code (passer d'une instruction vers une autre située en amont ou en aval, et référencée par son adresse. Il en existe deux types :

- Branchement inconditionnel (systématique) : ba adresse
- Branchement conditionnel : bcond adresse
  - La condition « cond » est évaluée par rapport aux indicateurs N, Z, V ou C, résultant de la dernière instruction qui les a validés (se terminant par CC)
  - Si la condition « cond » est vraie, on effectue un saut à l'adresse indiquée
  - Sinon, on passe à l'instruction suivante

Il existe 15 conditions de branchement dans craps. La table suivante en fournit la liste, et pour chacune d'elles, le code et l'état des indicateurs (flags) qu'elle doit vérifier.

Instruction de branchement	**cond (code)**	**Opération : branch**	**Flags**
ba	1000 (8)	always	1
beq (be, bz)	0001 (1)	on equal	Z
Bne (bnz)	1001 (9)	on not equal	Not Z
blu (bcs)	0101 (5)	less unsigned (carry set)	C
bgeu (bcc)	1101 (13)	greater or equal unsigned	Not C
bneg (bn)	0110 (6)	on negative	N
bpos (bnn)	1110 (14)	on positive	Not N
bvs	0111 (7)	on oVerflow set	V
bvc	1111 (15)	on oVerflow clear	Not V
ble	0010 (2)	on less or equal	(N xor V) or Z
bg (bgt)	1010 (10)	on greater	Not ((N xor V) or Z)
bl	0011 (3)	on less	N xor V
bge	1011 (11)	on greater or equal	Not (N xor V)
bleu	0100 (4)	on less or equal unsigned	Z or C
bgu	1100 (12)	on greater unsigned	Not (Z or C)

Les instructions de branchement permettent de mettre en place les structures de contrôle : Si Sinon, Répéter, TantQue, etc.. Par exemple, soit la boucle suivante en langage algorithmique :

```
Index ← 10
Répéter
        Action
        Index ← Index - 1
Jusqu'à Index=0
```

On peut choisir d'utiliser le registre r1 pour stocker Index, et on peut écrire :

```
Boucle :    …       // Action
            …
            subcc   %r1, 1, %r1
            bne     Boucle
            …
```

- Pour ne pas avoir à connaître l'adresse de l'instruction vers laquelle on veut effectuer le branchement, on lui associe une étiquette (label, nom) qui doit être mis en début de ligne et se terminer par :. L'assembleur calculera automatique le déplacement (distance relative par rapport à l'instruction ba ou bcond)
- bne (branch if not equal, ou bnz : branch if not Z) effectue le branchement si la condition « ne » est vraie, c'est à dire si l'instruction « subcc » a généré Z=0 (r1 <> 0).
- Il faut noter l'intérêt de l'instruction subcc, car sub ne mémorise pas les indicateurs.

Un autre exemple, qui calcule le max de deux valeurs non signées :

```
Max ← A
Si A < B Alors
        Max ← B
FinSi
```

On peut choisir d'utiliser le registre r1 pour stocker A, r2 pour B et r3 pour Max, et on peut écrire :

```
        add     %r1, 0, %r3     // r3 ← r1
        subcc   %r1, %r2, %r0
        bgeu    finsi           // (r1-r2) >= 0 non signé (u : unsigned)
                                // r1 >= r2 non signé => branchement à finsi
        add     %r2, 0, %r3
finsi : ba finsi                // boucle infini simulant la fin du programme
```

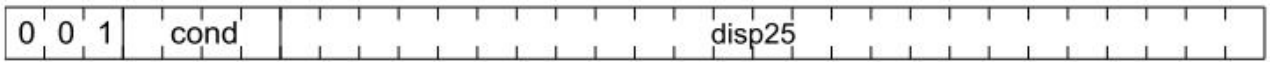
En langage assembleur, l'instruction cmp %r1,%r2 est équivalente à subcc %r1, %r2, %r0

Une autre solution possible, mais un peu plus compliquée :

```
        add     %r1, 0, %r3     // r3 ← r1
        subcc   %r1, %r2, %r0
        blu     sesi            // branch if r1 less unsigned then r2
        ba      finsi           // Ne pas oublier ce ba, sinon seqsi sera exécutée
seqsi : add     %r2, 0, %r3
finsi : ba finsi
```

Si on souhaite calculer le max entre 2 valeurs signées, il suffit de remplacer bgeu par bge et blu par blt (less than)

Les instructions de branchement ont le format suivant :



- code indique le code de la condition (colonne 2 du tableau dessus)
- disp25 : représente le déplacement relatif entre l'instruction cible et l'instruction de branchement

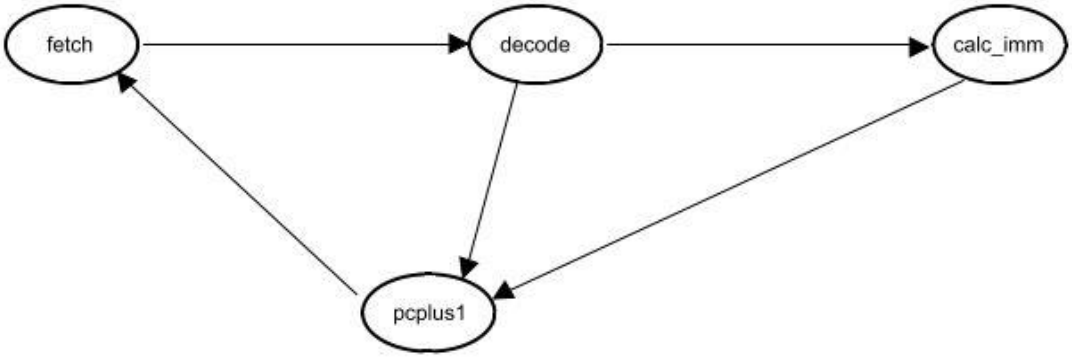
Leur exécution s'effectue de la façon suivante :

- si « cond » est vraie alors PC ← PC + disp25
- sinon PC ← PC + 1

Le module *branch(cond[3..0], N, Z, V, C : brok)* fourni, permet de générer la sortie « brok » qui indique si la condition « cond » est vraie ou non.

1- Compléter la partie séquenceur de craps en y intégrant le traitement des instructions de branchement :

- en complétant le graphe d'états vu en TD1 et TP1 : le nombre d'états ajoutés doit être le plus petit possible. L' ajout se fait à partir de l'état « decode ».



- en indiquant pour chaque transition, la condition, l'action réalisée, et les valeurs correspondantes pour areg, breg, dreg, cmd_ual, et oe_num
- et en ajoutant les états nécessaires dans le module craps, et en veillant à vérifier les incidences sur les états déjà présents.

2- Tester votre craps avec les exemples suivants :

Algorithme	Assembleur craps	Code hexadécimal

Index ← 2
Compteur ← 0
Répéter
Compteur ← Compteur + 1
Index ← Index -1
Jusqu'à Index=0
	// Index dans %r1
	// Compteur dans %r2
	bcle: add %r2, 1, %r2
	subcc %r1, 1, %r1
	bne bcle
	// ne : %r1 <> 0
	finb: ba finb
		
		0x8400a001
		0x82a06001
		0x33fffffe
		0x30000000

Tantque A < B Faire
A ← A + 1
FinTQ
	// A dans r1, B dans r2
	bcle: subcc %r1, %r2, %r0
	bgeu fintq
	add %r1, 1, %r1
	ba bcle
	fintq: ba fintq
		
		0x80a04002
		0x3a000003
		0x82006001
		0x31fffffd
		0x30000000

Si A > B alors
A ← A – B
Sinon
A ← A + 1
FinSi
	// A dans r1, B dans r2
	subcc %r1, %r2, %r0
	bgu si
	sinon: add %r1, 1, %r1
	ba finsi
	si: sub %r1, %r2, %r1
	finsi: ba finsi
		
		0x80a04002
		0x38000003
		0x82006001
		0x30000002
		0x82204002
		0x30000000

3- Ecrire et tester le programme qui calcule la somme des N premiers entiers, avec N dans r1 et le résultat dans r2.