Table des matières

1 Description du jeu

CodeWar* est un jeu opposant des programmes s'exécutant sur une architecture rudimentaire. Le « plateau » de jeu est constitué d'une grille de processeur qui exécutent chacun un programme. La table 1 illustre un plateau de jeu de 256 processeurs répartis sur une grille de 16×16. Chaque processeur possède sa mémoire propre mais peut également adresser celle d'une partie des autres. La connexion de la grille est considérée thorique, c'est à dire que les processeurs de la bordure droite ont comme voisins de droite la bordure gauche et inversement (de même pour la bordure haute et

*Ce jeu est très largement inspiré de CyberWar par Gilles Grimaud (`http://www.lifl.fr/~grimaud`)

basse). Ainsi, F a comme voisin de droite G, G a comme voisin de gauche F, H a comme voisin du haut I et I a comme voisin du bas H.

	H													
								B						
G						C	A	D					F	
							E							
	I													

Tab. 1 – Le plateau de jeu

Le but du jeu est bien évidemment de contrôler un maximum de processeurs à la fin du combat. La durée du combat est limité à un nombre donné de cycles et donc à un nombre limité d'instructions. Un processeur sera considéré comme controllé par un joueur si la couleur de ce joueur est stockée à une adresse donnée dans la mémoire du processeur.

Chaque joueur commencera le jeu avec un processeur, situé aléatoirement dans la grille, dans lequel sera chargé son programme et sa couleur (également choisie aléatoirement par la plateforme de jeu) sera stockée à l'adresse dédiée (voir Section 2.3) pour la description de l'espace d'adressage. La difficulté du jeu résidera donc dans la capacité du programme à s'assurer que sa couleur soit stockée dans un maximum de mémoires et qu'elle le reste jusqu'à la fin du combat.

2 Description de l'architecture d'un processeur

Chaque processeur est un processeur 16 bits, possédant 256 octets de RAM et 8 registres.

2.1 Registres

Le processeur possède 8 registres, de R0 à R7. Tous ces registres peuvent être utilisés dans toutes les instructions manipulant des registres. Deux registres ont cependant un rôle particulier, R6 et R7.

R6 est le registre correspondant au compteur de programme et peut également être utilisé avec le nom PC. C'est dans ce registre qu'est contenu l'adresse de la prochaine instruction à exécuter.

R7 est le registre utilisé par le processeur comme registre de pile, il peut également être utilisé avec le nom SP. C'est toujours R7 qui sera utilisé par le processeur comme registre de pile « officiel ». Les instructions d'appel de sous-routines, de retour de sous-routines etc... utiliseront donc ce registre comme registre de pile. Le registre contient l'adresse du sommet actuel de la pile et la pile grandit dans le sens des adresses hautes vers les adresses basses. Empiler une valeur revient donc à décrémenter le registre de la taille de cette valeur et stocker la valeur à l'adresse obtenue. Dépiler consiste à lire la valeur située à l'adresse contenue dans SP puis à l'incrémenter de la taille de la valeur lue.

2.1.1 Registre d'état

Le registre d'état est constitué de différents flags qui représente chacun un état du processeur. La valeur de ces flags est modifiée après l'exécution d'une instruction en fonction du résultat de cette instruction. Ce registre n'est pas accessible directement par le programmeur. La taille de ce registre est également de 16 bits (16 bits seront donc empilés lors du déclenchement d'une interruption par exemple, voir Section 2.7). Cependant, seuls les 3 bits de poids faibles sont utilisés :

- le flag C (Carry flag) est le flag de retenue. Il permet de savoir si une opération a généré une retenue.
- le flag Z (Zero flag). Ce flag est mis à 1 si le résultat de l'opération est zéro ;
- le flag N (Negative flag). Ce flag est mis à 1 si le résultat de l'opération est négatif (ce flag prends donc la valeur du bit de poids fort du résultat de l'opération, ce bit étant le bit de signe en arithmétique complément à deux).

2.2 Modes d'adressage

Les modes d'adressage sont les différentes façon qu'a le processeur d'obtenir une valeur pour effectuer une opération. De façon simplifiée, le processeur peut utiliser une valeur immédiate, c'est à dire une constante inscrite dans le programme, la valeur contenue dans un registre, la valeur de la zone mémoire dont l'adresse est donnée par une constante (adressage

direct), la valeur de la zone mémoire dont l'adresse est contenue dans un registre (adressage indirect). Les modes d'adressage disponibles sur le processeur ainsi que la syntaxe assembleur correspondante à chaque mode sont donnés dans le tableau suivant (R6 et R7 peuvent être également noté PC et SP respectivement) :

Mode d'adressage	Syntaxe	Commentaire
Registre	Rn	$n \in [0, 7]$
Valeur immédiate	#v	v, valeur entière sur 16 bits #v si v est codé en décimal #0xv si v est codé en hexadécimal #bv si v est codé en binaire
Adressage direct	@x	x, adresse mémoire sur 16 bits (codée en hexadécimal)
Adressage indirect	(Rn)	$n \in [0, 7]$
Adressage indirect avec post-incrément	(Rn)+	$n \in [0, 7]$. charge depuis/stocke à l'adresse Rn **puis** incrémente Rn de la taille d'un mot (2)
Adressage indirect avec pré-décrément	-(Rn)	$n \in [0, 7]$. décrémente Rn de la taille d'un mot (2) **puis** charge depuis/stocke à l'adresse Rn

2.3 Espace d'adressage

2.3.1 Construction d'une adresse

La mémoire du processeur **et** celles du voisinage peuvent être adressées de manière directe. L'octet de poids faible permet d'accéder aux 256 octets d'une mémoire tandis que l'octet de poids fort permet de sélectionner les coordonnées (relativement au processeur courant) de la mémoire que l'on veut accéder. Les 16 bits d'une adresse mémoire sont donc construits comme suit :

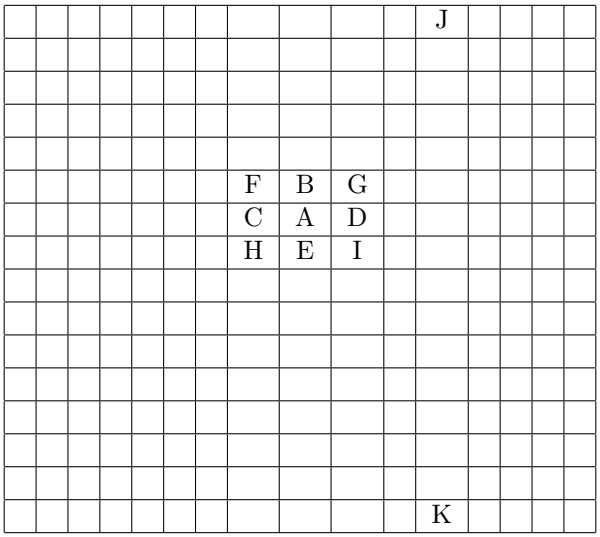
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
X	X	X	X	Y	Y	Y	Y	@	@	@	@	@	@	@	@

- les bits b_{15} à b_{12} précisent le déplacement horizontal
  - 0 sélectionne la mémoire d'un processeur de la même colonne
  - 1 sélectionne la mémoire d'un processeur de la colonne immédiatemment à droite de celle du processeur dans lequel est exécuté le code
  - -1 sélectionne la mémoire d'un processeur de la colonne immédiatemment à gauche
- les bits b_{11} à b_8 précisent le déplacement vertical
  - 0 sélectionne la mémoire d'un processeur de la même ligne
  - 1 sélectionne la mémoire d'un processeur de la ligne immédiatemment en dessous de celle du processeur dans lequel est exécuté le code
  - -1 sélectionne la mémoire d'un processeur de la colonne immédiatemment au dessus
- les bits b_7 à b_0 permettent d'adresser les 256 octets de la mémoire sélectionnée.

Les coordonnées sont donnés en binaire signé sur 4 bits. Les différentes valeurs sont donc :

Valeur décimale	Valeur Binaire	Valeur Hexadécimale
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
-1	1111	0xF
-2	1110	0xE
-3	1101	0xD
-4	1100	0xC
-5	1011	0xB
-6	1010	0xA
-7	1001	0x9
-8	1000	0x8

Exemples Supposons le plateau de jeu suivant :



Si on considère du code s'exécutant sur le processeur A alors l'adresse :

`0x0000` (X=0, Y=0, @=0x00) désigne l'adresse 0 du processeur A ;

`0x0001` (X=0, Y=0, @=0x01) désigne l'adresse 1 du processeur A ;

`0x1000` (X=1, Y=0, @=0x00) désigne l'adresse 0 du processeur D ;

`0x11BC` (X=1, Y=1, @=0xBC) désigne l'adresse 0xBC du processeur I ;

`0x0FAA` (X=0, Y=-1, @=0xAA) désigne l'adresse 0xAA du processeur B ;

`0x3A00` (X=3, Y=-6, @=0x00) désigne l'adresse 0x00 du processeur J ;

`0x3900` (X=3, Y=-7, @=0x00) désigne l'adresse 0x00 du processeur K (à cause de l'aspect thorique de la grille, aller une case plus haut que le processeur J nous amène directement en bas de la grille et donc au processeur K).

C'est grâce à ce mécanisme d'adressage que les processeurs vont pouvoir « s'attaquer ».

2.3.2 Adresses réservées

Certaines adresses d'une mémoire ont un but bien particulier. Les adresses données sont uniquement les 8 bits de poids faible. En effet, ces zones existent sur tout les processeurs et peuvent donc être adressées par tous les voisins.

Adresse 8bits	Taille	Signification
@1	16 bits	*Couleur* du processeur. C'est cette adresse qu'il faut controller à tout prix Les 16 bits contiennent les composantes rouge, vert et bleu de la couleur 5 bits par composante : xrrrrrvvvvvbbbbb
@3-@9	–	Vecteur d'interruptions
@3	8 bits	adresse de la routine d'interruption d'instruction illégale (erreur au décodage de l'instruction)
@4	8 bits	adresse de la routine d'interruption du timer
@5	8 bits	adresse de la routine d'interruption générée par un autre processeur
@A-@D	32 bits	Configuration du timer (cf. Section 2.8)

2.4 Jeu d'instructions, description rapide

La syntaxe de l'assembleur est assez semblable à celle de l'assembleur 68000. D'une manière générale, les instructions se présentent sous la forme :

```
INST source, destination
```

où la source et la destination peuvent généralement être choisies parmis les modes d'adressage cités section 2.2.

Les sections suivantes donnent une description rapide des différentes instructions. Les spécifications détaillées de chaque instruction seront données par la suite.

2.4.1 Mouvement/Copie de données

Ces instructions permettent de lire, écrire ou manipuler des données contenues dans des registres ou dans la mémoire :

- MOVE : copie le contenu d'un registre ou d'une zone mémoire dans un registre ou une zone mémoire ;
- PUSH : empile une valeur ;
- POP : dépile une valeur ;

2.4.2 Opérations arithmétiques et logiques

Ces instructions effectuent des opérations arithmétiques et logiques sur des données contenues dans des registres ou dans la mémoire :

- ADD : effectue une addition ;
- CMP : compare deux valeurs et mets à jour les flags. Utilisé généralement conjointement à une instruction de branchement ;
- SUB : effectue une soustraction ;
- LSL, LSR : décalage logique vers la gauche (resp. vers la droite).
- AND : effectue l'opération ET bit à bit ;
- OR : effectue l'opération OU bit à bit ;
- XOR : effectue l'opération OU EXCLUSIF bit à bit ;
- NOT : effectue le complément à 1 (les bits à 0 sont mis à 1 et les bits à 1 sont mis à 0).

2.4.3 Contrôle de programme

Ces instructions permettent d'affecter le déroulement du programme :

- Bxx : sauts conditionnels relatifs ;
- Jxx : saut conditionnels absolus ;
- BSR : appel de sous-routines relatif ;
- JSR : appel de sous-routines absolu ;
- RTS : retour d'une sous-routine ;
- TRAP : génération d'une interruption (voir Section 2.7 pour le fonctionnement des interruptions) ;
- RTE : retour d'une routine d'interruption (voir Section 2.7).

2.5 Jeu d'instructions, description détaillée

Les sections suivantes présentent de façon détaillée chacune des instructions. Dans les description qui suivent, S représentera la valeur de la source avant l'exécution de l'instruction, D celle de la destination avant l'exécution et RES le résultat de l'opération. Ces lettres suivies d'un chiffre (par exemple S1) représente un bit donné. Par exemple, S4 représente le bit 4 (et donc le 5ième bit en partant du bit de poids faible, indicé 0) de la source. RES0 représente le bit de poids faible du résultat de l'opération, RES15 le bit de poids fort du résultat et ainsi de suite.

Pour chaque instruction, sa description se présentera de la manière suivante :

****** Nom de l'instruction ******

Description rapide de la fonction de l'instruction

Syntaxe
La syntaxe de cette instruction.

Modes d'adressages autorisés
Liste des modes d'adressages autorisés pour la source et la destination

Opération effectuée
Description précise de l'opération effectuée. S représente la source, D la destination

Flags
Les flags affectés et de quelle façon.

2.5.1 Mouvement/Copie de données

****** MOVE ******

Copie la source vers la destination. Il existe trois variation de cette instruction. MOVE, MOVE.l et MOVE.h. MOVE effectue la copie des 16 bits, MOVE.l la copie des 8 bits de poids faible de la source et MOVE.h la copie des 8 bits de poids fort de la source. Dans les deux derniers cas, si la destination est un registre, les 8 bits sont copiés dans les 8 bits de poids faible de la destination, les 8 bits de poids fort restant inchangés. Sinon, si la destination est une adresse mémoire, seul l'octet situé à l'adresse utilisée sera modifié. Il faut donc être très vigilant en utilisant MOVE.l ou MOVE.h

Syntaxe
MOVE[.l/.h] S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn) **et** D: Rn, (Rn), (Rn)+, -(Rn), @x

ou S: #x, @x **et** D: Rn, (Rn), (Rn)+, -(Rn). Il ne peut en effet y avoir qu'une seule valeur immédiate ou adresse parmis les opérandes

Opération effectuée
$D = S$

Flags

- C = 0
- N = S15 (ou S7 si MOVE.l est utilisé)
- Z = 1 si S = 0, 0 sinon (Attention, dans le cas de MOVE.l ou MOVE.h, on considère S comme étant les 8 bits de poids faible ou de poids fort respectivement)

****** PUSH ******

Empile une valeur sur la pile

Syntaxe
PUSH S

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn)

Opération effectuée
Cette instruction effectue l'équivalent de MOVE S,-(R7) à ceci près qu'elle ne supporte pas les valeurs immédiates

Flags
Même comportement que MOVE S, -(R7)

****** POP ******

Dépile une valeur de la pile

Syntaxe
POP D

Modes d'adressages autorisés
D: Rn, (Rn), (Rn)+, -(Rn)

Opération effectuée
Cette instruction effectue l'équivalent de MOVE (R7)+,D à ceci près qu'elle ne supporte pas une adresse en destination

Flags
Même comportement que MOVE (R7)+, D

2.5.2 Opérations arithmétiques

****** ADD ******

Ajoute la source à la destination et stocke le résultat dans la destination

Syntaxe
ADD S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
$D = S + D$

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = 1 si le résultat est supérieur à 0xFFFF

****** CMP ******

Compare un registre à une opérande

Syntaxe
CMP S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
L'opération sous jacente à la comparaison est une soustraction qui ne modifie ni la source ni la destination, les flags sont affectés en fonction du résultat de $RES = D - S$

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = 1 si D est inférieur à S, 0 sinon

****** SUB ******

Effectue une soustraction

Syntaxe
SUB S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
D = D - S

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = 1 si D est inférieur à S, 0 sinon

****** LSL ******

Effectue le décalage à gauche d'une valeur stockée dans un registre d'un nombre de bits égal à une valeur immédiate ou au contenu d'un registre

Syntaxe
LSL S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
$D = D << S$

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = la valeur du dernier bit à avoir été décalé

****** LSR ******

Effectue le décalage à droite d'une valeur stockée dans un registre d'un nombre de bits égal à une valeur immédiate ou au contenu d'un registre

Syntaxe
LSR S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
$D = D >> S$

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = la valeur du dernier bit à avoir été décalé

****** AND ******

Effectue un ET logique bit à bit

Syntaxe
AND S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
D = S & D

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = 0

****** OR ******

Effectue un OU logique bit à bit

Syntaxe
OR S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
D = S | D

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = 0

****** XOR ******

Effectue un OU exclusif logique bit à bit

Syntaxe
XOR S, D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (5 bits)

D: Rn

Opération effectuée
D = S ^ D

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = 0

****** NOT ******

Effectue un NON logique bit à bit (aussi appelé complément à un)

Syntaxe
NOT S

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn)

Opération effectuée
S = ~S

Flags

- N = RES15
- Z = 1 si RES = 0, 0 sinon
- C = 0

2.5.3 Contrôle de programme

****** Bxx ******

Les instructions Bxx (uù xx est remplacé par deux lettres représentant la condition) permettent d'effectuer un branchement dans le programme en fonction de la valeur de certains flags. C'est grâce à ces instructions qu'on va pouvoir modifier le flot d'exécution du programme et donc effectuer des tests, des boucles etc...

Symbole	Nom long	Signification
BCC	Branch Carry Clear	Branche si C = 0
BCS	Branch Carry Set	Branche si C = 1
BEQ	Branch EQual	Branche si Z = 1
BNE	Branch Not Equal	Branche si Z = 0
BLT	Branch Less Than	Branche si C = 1
BLE	Branch Less or Equal	Branche si C = 1 ou si Z = 1
BGT	Branch Greater Than	Branche si C = 0
BGE	Branch Greater or Equal	Branche si C = 0 ou si Z = 1
BRA	BRanch Always	Branche dans tous les cas

Syntaxe
Bxx D

Modes d'adressages autorisés
S: Rn, (Rn), (Rn)+, -(Rn), #x (8 bits)

Opération effectuée
PC = PC + D si la condition est vérifiée. Le saut correspond simplement à une modification de la valeur de PC. Comme la valeur de D est ajoutée au PC, on parle de branchement relatif (le branchement s'effectue par rapport à la valeur actuel du PC). Lorsque que vous développerez l'assembleur, c'est lui qui se chargera de transformer un label (un nom explicite pour une ligne de votre programme) en une valeur immédiate utilisée pour D.

Flags
Non affectés

****** Jxx ******

Les instructions Jxx (uù xx est remplacé par deux lettres représentant la condition) permettent d'effectuer un branchement dans le programme en fonction de la valeur de certains flags. C'est grâce à ces instructions qu'on va pouvoir modifier le flot d'exécution du programme et donc effectuer des tests, des boucles etc...

Symbole	Nom long	Signification
JCC	Jump Carry Clear	Branche si C = 0
JCS	Jump Carry Set	Branche si C = 1
JEQ	Jump EQual	Branche si Z = 1
JNE	Jump Not Equal	Branche si Z = 0
JLT	Jump Less Than	Branche si C = 1
JLE	Jump Less or Equal	Branche si C = 1 ou si Z = 1
JGT	Jump Greater Than	Branche si C = 0
JGE	Jump Greater or Equal	Branche si C = 0 ou si Z = 1
JMP	JuMP always	Branche dans tous les cas

Syntaxe
Jxx D

Modes d'adressages autorisés
D: Rn, (Rn), (Rn)+, -(Rn), #x (8 bits)

Opération effectuée
PC = D si la condition est vérifiée. Le saut correspond simplement à une affectation de la valeur de PC. Comme la valeur de D copiée dans PC, on parle de branchement absolu (le branchement s'effectue à une adresse fixe en mémoire quelque soit la valeur précédente de PC). Lorsque que vous développerez l'assembleur, c'est lui qui se chargera de transformer un label (un nom explicite pour une ligne de votre programme) en une valeur immédiate utilisée pour D.

Flags
Non affectés

****** BSR ******

Appelle une sous-routine dont l'adresse est donnée de façon relative à la valeur courante de PC. Empile l'adresse de l'instruction suivante (dans la pile dont le sommet est donné par SP) et branche à l'adresse de la sous routine

Syntaxe
BSR D

Modes d'adressages autorisés
D: Rn, (Rn), (Rn)+, -(Rn), #x (8 bits)

Opération effectuée
PC est empilé puis PC = PC + D

Flags
Non affectés

****** JSR ******

Appelle une sous-routine dont l'adresse est donnée de façon absolue. Empile l'adresse de l'instruction suivante (dans la pile dont le sommet est donné par SP) et branche à l'adresse de la sous routine

Syntaxe
JSR D

Modes d'adressages autorisés
D: Rn, (Rn), (Rn)+, -(Rn), #x (8 bits)

Opération effectuée
PC est empilé puis PC = D

Flags
Non affectés

****** RTS ******

Termine une sous routine

Syntaxe
RTS

Modes d'adressages autorisés
N/A

Opération effectuée
Dépile l'adresse de retour et y branche (PC = Valeur de retour)

Flags
Non affectés

****** TRAP ******

Déclenche une interruption

Syntaxe
TRAP D

Modes d'adressages autorisés
D: Rn, (Rn), (Rn)+, -(Rn), #x (8 bits)

Opération effectuée
Déclenche une interruption sur un processeur, les 8 bits de poids faible de D représente les coordonnées relatifs du processeur sur lequel déclencher l'interruption (les coordonnées sont codés de la même façon que les 8 bits de poids fort d'une adresse mémoire)

Flags
Non affectés

****** RTE ******

Retourne d'une routine d'interruption. Dépile le registre d'état afin de restaurer les flags, dépile le PC et branche à l'adresse de l'intruction qui devait être exécutée avant le déclenchement de l'interruption

Syntaxe
RTE

Modes d'adressages autorisés
N/A

Opération effectuée
Dépile les flags, dépile l'adresse de retour et y branche (PC = valeur de retour)

Flags
Affectés à la valeur dépilée

2.6 Codage des instructions

La plupart des instructions sont codées sur deux octets à l'exception de l'instruction MOVE, plus complexe, qui est codée sur quatre octets. Dans tous les cas, les 5 bits de poids fort représente l'instruction à exécuter. Les 11 bits restant codent la valeurs des opérandes et ce codage est donc différent en fonction du nombre d'opérandes.

2.6.1 Code des instructions

Instruction	Code
MOVE	00
PUSH	01
POP	02
ADD	03
CMP	04
SUB	05
LSL	06
LSR	07
AND	08
OR	09
XOR	0A
NOT	0B
BCC/BGT	0C
BCS/BLT	0D
BEQ	0E
BNE	0F
BLE	10
BGE	11
BRA	12
BSR	13
JCC/JGT	14
JCS/JLT	15
JEQ	16
JNE	17
JLE	18
JGE	19
JMP	1A
JSR	1B
RTS	1C
TRAP	1D
RTE	1E

2.6.2 Instructions à zéro opérande

Le format d'une instruction à zéro opérande est le suivant :

i	i	i	i	i	0	0	0	0	0	0	0	0	0	0	0

Les 5 bits de poids fort sont le code de l'instruction, les autres bits sont mis à zéro.

2.6.3 Instructions à un opérande

Le format d'une instruction à un opérande est le suivant :

i	i	i	i	i	t	t	t	v	v	v	v	v	v	v	v

- Les 5 bits de poids fort (i) sont le code de l'instruction ;
- Les 3 bits suivants (t) décrivent le type de l'opérande, les valeurs possibles sont :

Valeur	Type d'opérande
000	Registre : `Rn`
001	Registre pré-décrémenté : `-(Rn)`
010	Adressage indirect : `(Rn)`
011	Registre post-incrémenté : `(Rn)+`
100	Valeur immédiate : `#val`
101	Adresse : `@val`

- Les 8 bits de poids faible (v) sont la valeur de l'opérande. En fonction du type, cet octet désigne :
  - Le numéro du registre, entre 0 et 7
  - La valeur immédiate, sur 8 bits uniquement
  - Une adresse, sur 8 bits uniquement

2.6.4 Instructions à deux opérandes

Le format d'une instruction à deux opérandes est le suivant :

i	i	i	i	i	r	r	r	t	t	t	v	v	v	v	v

- Les 5 bits de poids fort (i) sont le code de l'instruction ;
- Les 3 bits suivants (r) sont le numéro du registre de destination (toutes les instructions à deux opérandes à l'exception de MOVE ne peuvent avoir qu'un registre comme destination)
- Les 3 bits suivants (t) décrivent le type de la source. Les valeur possibles sont les même que pour une instruction à une opérande à la différence qu'on ne peut pas avoir d'adresse
- Les 5 bits suivants (v) sont la valeur de l'opérande. En fonction du type, ces bits désignent :
  - Le numéro du registre, entre 0 et 7
  - La valeur immédiate, sur 5 bits uniquement

2.6.5 Cas particulier, l'instruction MOVE

L'instruction MOVE permettant des choses plus complexe, sont codage est spécifique. Elle nécessite quatre octets pour pouvoir être codée. Ce deux octets sont les suivants :

Mot 1	i	i	i	i	i	h	l	t1	t1	t1	t2	t2	t2	r	r	r
Mot 2	v	v	v	v	v	v	v	v	v	v	v	v	v	v	v	v

- Les 5 bits de poids fort (i) sont le code de l'instruction ;
- Les deux bits suivants (h et l) permettent de différencier un `MOVE`, un `MOVE.h` et un `MOVE.l`. Pour un `MOVE`, ces deux bits sont à 1. Pour un `MOVE.h`, seul le bit h est à 1 et pour un `MOVE.l`, seul le bit l est à 1
- Les 3 bits suivants (t1) définissent le type de la source (c.f. tableau des intructions à un opérande)
- Les 3 bits suivants (t2) définissent le type de la destination (c.f. tableau des intructions à un opérande)
- Les 3 derniers bits (r) du premier mot codent un numéro de registre. Dans le cas où la source est une valeur immédiate ou une adresse, ces trois bits codent le numéro de registre de la destination (si la source d'un `MOVE` est une valeur immédiate ou une adresse, la destination ne peut être qu'un registre ou un adressage par registre (indirect, pré-décrémenté ou post-incrémenté)). Sinon, ces trois bits codent le numéro du registre de la source ;
- L'intégralité du deuxième mot (v) code la valeur de la destination dans le cas où la source n'est ni une valeur immédiate ni une adresse (dans le cas contraire, ce mot contient la valeur immédiate ou l'adresse en question). Dans le cas où ce mot code la destination, les valeurs possibles sont :
  - Un numéro de registre entre 0 et 7
  - Une adresse sur 16 bits

2.7 Mécanisme d'interruptions

Le mécanisme d'interruptions permet au processeur de stopper un traitement en cours pour réagir à un événement. Dans notre cas, il existe trois événements :

- Le processeur décode une instruction illégale (toute valeur qui ne correspond pas à une instruction) ;
- Le timer est arrivé à expiration (voir Section 2.8) ;
- Un processeur a déclenché une interruption via l'instruction TRAP.

A chaque événement est associé un traitement appelé routine d'interruption. Cette routine est une suite d'instructions correspondant au traitement à effectuer si l'événement se produit. Ces routines sont stockées en mémoire et le processeur se réfère au vecteur d'interruptions pour connaître l'adresse mémoire de la routine associée à un événement. Le vecteur d'interruption est tout simplement un « tableau » d'adresse mémoire, chaque case de ce tableau correspondant à un événement :

- si le processeur décode une instruction illégale, il va chercher l'adresse de la routine qui doit traiter cette événement à l'adresse @3 (la première case du vecteur d'interruption) ;
- si le timer arrive à expiration, le processeur va chercher l'adresse de la routine à l'adresse @4 (la deuxième case du vecteur d'interruption) ;
- enfin, si un autre processeur à déclenché une interruption via l'instruction TRAP, le processeur qui reçoit l'interruption va chercher l'adresse de la routine à l'adresse @5.

On peut remarquer que les adresses stockées dans le vecteur d'interruption sont uniquement de un octet. En effet, une interruption ne peut déclencher qu'un branchement vers du code situé dans la mémoire propre du processeur. L'adresse sur 16 bits est donc calculée automatiquement en mettant 0 dans l'octet de poids fort.

Une fois que le processeur connait l'adresse à laquelle il doit brancher, il doit sauvegarder son état afin de pouvoir revenir exactement où il en était au moment où l'événement s'est produit. En effet, un événement pouvant se produire au milieu d'un traitement, une fois l'événement traité, le processeur doit continuer son traitement comme si de rien n'était.

Le processeur doit donc sauvegarder le contenu du registre de compteur de programme (comme dans le cas d'un appel de sous routine classique) mais **aussi** le registre d'état. Cette sauvegarde est bien évidemment faite à l'aide de la pile. Une fois que le processeur à empilé le registre d'état et le registre PC, il branche à l'adresse de la routine d'interruption. L'équivalent de ce comportement peut être écrit comme la suite d'instruction suivante (on suppose que le processeur réagit à l'événement timer, donc la routine situé à l'adresse @4 de la mémoire) :

```
PUSH PC     ; on empile le compteur de programme
; on empile le registre d'état (2 octets en plus sur la pile)
MOVE @4, PC ; on branche à l'adresse de la routine d'interruption du timer
            ; la dernière instruction peut également être écrite JMP @4,
            ; l'instruction JMP n'étant ni plus ni moins qu'une modification de PC
```

La routine de traitement d'une interruption se termine par l'instruction RTE. Cette instruction est similaire à l'instruction RTS (qui permet de retourner d'une sous routine) à la différence qu'elle dépile, en plus du PC, le registre d'état. L'équivalent de ce comportement peut être écrit comme la suite d'instructions suivante :

```
; on dépile le registre d'états
POP PC       ; on dépile PC (ce qui a pour effet de brancher à l'instruction
             ; qui aurait du être exécutée si l'événement ne s'était pas produit
```

2.8 Configuration du Timer

Le timer est un mécanisme permettant de programmer un traitement pour qu'il s'exécute dans un nombre de cycle donné. Il est configuré par les valeurs situées aux 3 adresses suivantes :

- @A : l'octet situé à cette adresse est la valeur de l'horloge à laquelle l'événement de timer doit être déclenché. Cette octet est défini par le programmeur et n'est jamais changé autrement que par le programmeur ;
- @B : cet octet représente la valeur de l'horloge du processeur. Cette valeur est mise à jour par le processeur. Cette à cette valeur qu'est comparée la valeur précédente. Quand les deux sont égales, l'interruption du timer est déclenchée.
- @C : cet octet représente le nombre de cycles processeur pour chaque incrément d'horloge. Pour simplifier, si cet octet vaut 4, tous les quatre cycles, la valeure de l'horloge est augmentée de 1. S'il vaut 8, l'horloge est incrémentée tous les huit cycles etc...
- @D : cet octet vaut 0 quand le timer est désactivé. Passer cette valeur à 1 active le timer. Quand l'interruption timer est déclenchée, cet octet repasse automatiquement à 0 (le timer est donc désactivé)

Par exemple, si on souhaite exécuter un traitement dans 2000 cycles, on peut configurer le timer à l'aide des instructions suivantes :

```
XOR R0, R0
MOVE R0, @B     ; on initialise l'horloge à 0

MOVE.l #200, R0
MOVE.l R0, @C   ; on range la valeur 200 à l'adresse @C
                ; (donc l'horloge sera incrémentée tous les 200 cycles)
MOVE.l #10, R0
MOVE.l R0, @A   ; on range l'octet 10 à l'adresse @A, donc quand l'horloge
                ; vaudra 10, l'interruption sera déclenchée.
                ; Or, l'horloge vaudra 10 au bout de 10*200 = 2000 cycles

MOVE.l #1, R0
MOVE.l R0, @D  ; on démarre le timer
```

2.9 Initialisations

Lorsque que le jeu commence, les processeurs sont initialisés de la façon suivante :

- La valeur 0x10 est affectée au PC
- Deux fichiers de code sont chargés dans la mémoire de deux processeurs choisit aléatoirement
- Les couleurs de ces deux processeurs sont affectées également aléatoirement
- Les autres processeurs chargent les octets 0x10 et 0xD4 aux adresses 0x10 et 0x11. Ces deux octets correspondent à l'instruction `JMP @10` qui effectue une boucle infinie
- L'adresse @D de configuration du timer est initialisée à 0 (timer désactivé)