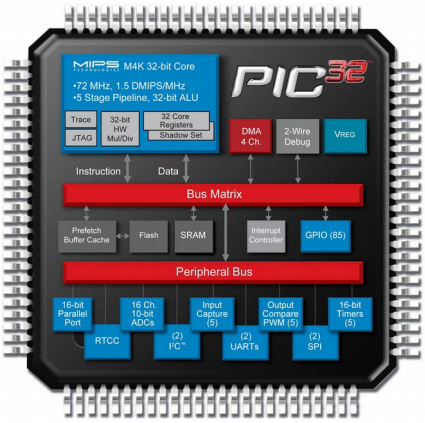
# ARCHITECTURE DES ORDINATEURS

# PROJET EN ASSEMBLEUR MIPS

**SUJET** : *LE JEU « PUISSANCE4 » programmé en assembleur mips*



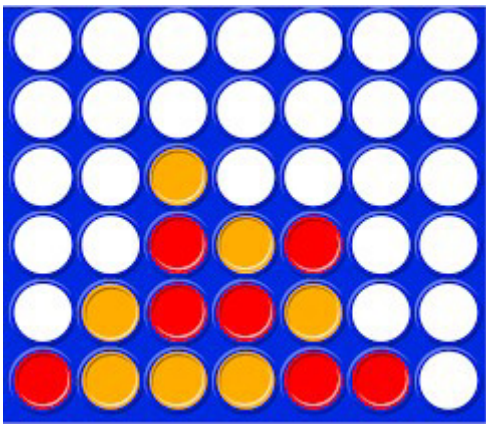

**Réalisé par** : *Pascal CHEN & Youssouf DIALLO*

**Encadré par** : **M. Olivier MARCHETTI et M. Guillaume MATHERON**

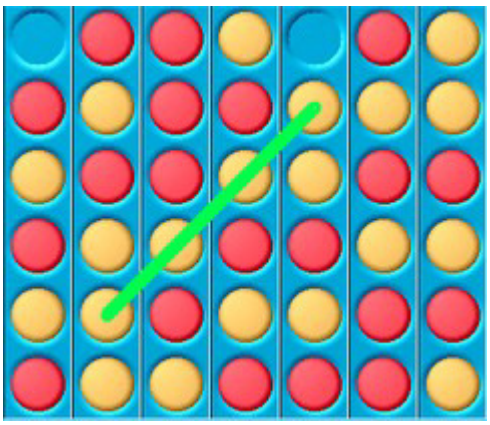
**Période** : semestre_1( EISE_3).
Janvier 2018

## I. LE JEU EN QUESTION

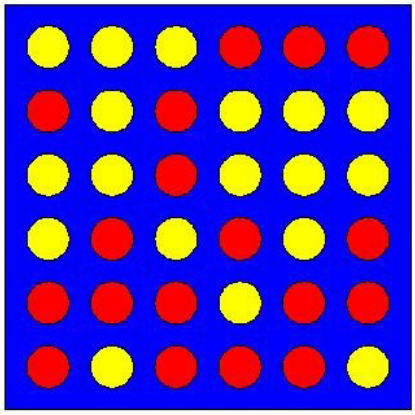
Le jeu <<puissance 4>> est un jeu qui se compose d'une grille de quarante deux cases réparties sur sept colonnes et six lignes. Il se joue à deux. Chaque joueur disposant de vingt et un jetons (d'une même couleur : jaune ou rouge), joue un coup à chaque tour. Pour chaque coup, le joueur insère un jeton dans une colonne (non pleine) et celui ci tombe jusqu'à occuper la case vide la plus basse de sa colonne. Pour remporter une partie, un joueur devra faire un alignement d'au moins quatre jetons de sa couleur (alignement vertical, horizontal ou diagonal). Si la grille est pleine et qu'aucun des joueurs n'a réussit à faire un tel alignement, alors le math est fini et c'est un match nul.



Jeu non fini.



Joueur jaune a gagné cette partie.



C'est un match nul

## II. TRAVAIL À FAIRE

Nous allons faire un programme en assembleur mips32, qui permet de jouer à ce jeu en saisissant au clavier un numéro de colonne (de 0 à 6) pour placer un jeton ( les deux joueur se partageant le clavier de l'ordinateur).
Pour ce faire nous utiliserons le logiciel MARS pour implémenter notre code.

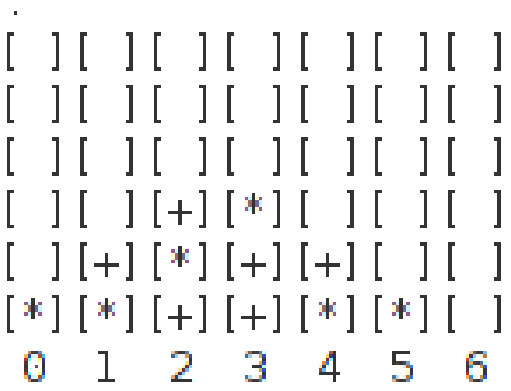
## III. CHOIX STRATÉGIQUES DE RÉALISATION

Pour modéliser le jeu, on a imaginé un tableau à deux dimensions : qui est implémenter sur 84 octets ( contiguës dans la mémoire). Chaque case correspond à deux octets (des demi-mots : half).

Pour représenter une case où on a pas encore joué, nous utilisons une case vide de ce tableau.
Pour représenter un jeton jaune, nous utilisons une case contenant le symbole étoile « [*] ».
Pour représenter un jeton rouge, nous utilisons une case contenant le symbole plus : « [+] ».

Exemple de notre modèle :

Voilà un exemple de notre modèle :




## IV. DESCRIPTION DES DIFFÉRENTES FONCTIONS RÉALISÉES

### a) **AFFICHAGE DE LA GRILLE** :

***afficherCase*** :
selon le contenu du registre $a0 (vide, * ou +), cette fonction affiche de manière symbolique une case.

***AfficherGrille*** :
grace à des boucles et à la fonction afficherCase, cette fonction affiche, de manière symbolique, le contenu courant de notre modèle.

### b) **MANIPULATION DE LA GRILLE** :

***demanderCouleurDeLaCaseXY*** :
Cette fonction retourne le contenu d'une case : vide, * ou +.
Et pour l'utiliser on doit mettre dans les registres $a0 et $a1 les coordonnés de la case en question( selon la vue de l'utilisateur). En suite, le retour de la fonction, c'est à dire le contenu de la case, se trouve dans le registre $v0.

***nbDeJetonsColonne*** :
A partir du numéro d'une colonne, qu'on aura préalablement mis dans le registre $a0, cette fonction retourne le nombre de jetons (= nombre de cases non vides) se trouvant sur cette colonne. Cette valeur de retour se trouve dans le registre $v0

***ajouterJeton :***
Cette fonction permet d'ajouter un jeton dans notre modèle de grille et dans une colonne particulière( qui devra être spécifiée dans le registre $a0). Elle positionnera correctement le jeton dans la colonne.

### c) **REALISATION DES FONCTIONS DU JEU**

***estCoupInvalide*** :
cette fonction permet de vérifier que la colonne jouée (numéro de colonne dans $a0) est conforme ou non à un coup valide : numéro de colonne compris entre 0 et 6, la colonne n'est pas déjà pleine. Retourne 0 ( dans le registre $v0) si c'est le cas et 1 sinon.

***jouerCoupPuissance4*** :
après avoir affiché l'état de la grille et un message invitant à jouer le joueur dont c'est le tour, saisira au clavier cette valeur jusqu'à ce qu'elle corresponde à un coup valide. La fonction retournera alors la valeur choisi dans le registre $v0.

***estCoupGagnantDiag*** :
cette fonction, étant donné le dernier coup joué (= numéro de colonne que se trouve dans le registre $a0), détermine si cette insertion a produit un alignement successif de quatre jetons d'une même couleur sur l'une des deux diagonales de notre modèle. La valeur de retour (0 ou 1) se trouve dans le registre $v0

***estCoupGagnantHori*** :
étant donné le dernier coup joué (= numéro de colonne que se trouve dans le registre $a0), cette fonction détermine si cette insertion a produit un alignement successif de quatre jetons d'une même couleur sur la ligne horizontale à partir de la position du dernier jeton. La valeur de retour (0 ou 1) se trouve dans le registre $v0

***estCoupGagnantVert*** :
étant donné le dernier coup joué (= numéro de colonne que se trouve dans le registre $a0), cette fonction détermine si cette insertion a produit un alignement successif de quatre jetons d'une même couleur sur la ligne verticale à partir de la position du dernier jeton. La valeur de retour (0 ou 1) se trouve dans le registre $v0.

***estCoupGagnant*** :
étant donné le dernier coup joué , détermine si cette dernière a produit une configuration gagnante horizontalement grâce à la fonction estCoupGagnantHori, verticalement grâce à la fonction estCoupGagantVert et diagonalement grâce à la fonction estCoupGagnantDiag.
La valeur de retour se trouve dans le registre $v0.

***jouerPartiePuissance4*** :
cette fonction permet de jouer une partie du jeu. Ceci grâce à des appels répétitifs( une boucle) de la fonction « jouerCoupPuissance4 ». A chaque appel fait jouer un des deux joueurs.

Après chaque coup , elle vérifie si :

-> le coup est gagnant ( grâce à la fonction estCoupGagnant). Si c'est vrai alors elle affiche un message indiquant celui qui a gagné et termine le jeu.

-> ce dernier coup a rempli la grille ( le nombre de coup maximal, 42, est atteint ). Si c'est le cas et qu'aucun des deux joueur n'a obtenu une configuration gagnante, alors elle affiche un message « match nul » et on termine le jeu.

## V. LISTE DES DIFFICULTÉS RENCONTRÉES

La principale difficulté rencontrée était qu'il y a peu de documentation sur le sujet de la programmation en assembleur. Et sur le logiciel MARS ce qui nous a empêcher d'avoir un jeu beaucoup plus réaliste. Notamment en utilisant des couleurs (jaune et rouge) pour les jetons.

## VI. REMARQUE SUR NOTRE PRATIQUE DE LA PROGRAMMATION ASSEMBLEUR

La réalisation de ce jeu a été d'une grande importance pour nous. Car ça nous a permis de pratiquer les notions théoriques vues en cours. Ça a été une occasion pour nous de bien pratiquer la programmation assembleur.

## VII. LE CODE PRODUIT

Le programme du jeu est dans le fichier « puissance4 .asm » qu'on a joint à ce rapport. On a testé toutes les fonctions du programme et le jeu fonctionne bien.

**BIBLIOGRAPHIE**

Origine des images présentent dans ce rapport :

**Les image de la première page** :

I**mage de gauche** :

du site microchip :

http://www.microchip.com/design-centers/32-bit/architecture/pic32mx-family

**Image de droite** :

Page wikipedia sur le langage machine

https://fr.wikipedia.org/wiki/Langage_machine

**Les trois images du jeu puissance4 à la page 2**:

**Image de gauche** :

Auteur Howard Wexler

Sur son article wikipedia

https://fr.wikipedia.org/wiki/Puissance_4

**image du milieu** :

Auteur M.Éric Du Colombier,

Analyste-Programmeur

Siteweb : https://www.rx14.info/public/index.php?a=Puissance4&s=3

**Image de droite** :

Pris sur le site de ilemaths.net

https://www.ilemaths.net/sujet-enigmo-66-l-anti-puissance-4-237132.html