TP2 : Microprocesseurs

# 1. Microprocesseur élémentaire

Chargez le circuit **micro1.circ**. Ce circuit représente un microprocesseur élémentaire composé des éléments suivants :

- 1 horloge (en bas à gauche) produisant alternativement un 0 et un 1
- 1 composant cycleur prenant en entrée le signal de l'horloge et produisant en sortie 4 signaux s'activant à tour de rôle et permettant d'activer successivement les différents composants :
  - Activation de la mémoire pour lire la prochaine instruction à exécuter
  - Activation de l'UAL pour réaliser l'opération indiquée par l'instruction
  - Activation de l'écriture dans les registres pour stocker le résultat dans les registres
  - Incrémentation du compteur indiquant l'adresse de la prochaine instruction à exécuter
- 1 compteur (à gauche au milieu) dont la sortie O est reliée à l'entrée A (Adresse) de la mémoire. Ce compteur s'incrémente automatiquement à chaque cycle d'horloge afin d'indiquer l'adresse de la prochaine instruction à exécuter.
- 1 mémoire dont chaque case contient une instruction en langage machine.
- 1 unité arithmétique et logique (ALU) permettant de réaliser des calculs. L'ALU possède 2 entrées I0 et I1 et produit le résultat sur sa sortie O, lorsqu'elle reçoit le signal d'activation sur son entrée E. L'opération réalisée par l'ALU dépend des entrées C3,C2,C1 et C0. La table à droite indique l'opération réalisée en fonction des valeurs de C3,C2,C1,C0.
  Z, N et C sont des bits en sortie (appelés indicateurs ou drapeaux ou flags) dont la valeur dépend du résultat de la dernière opération réalisée par l'UAL.
  - Si le résultat est nul alors Z passe à 1 sinon il passe à 0.
  - Si le résultat est négatif alors N passe à 1 sinon il passe à 0.
  - Si le résultat comporte une retenue alors C passe à 1 sinon il passe à 0.
- 2 registres 16 bits A et B servant à stocker le résultat produit par l'UAL (à condition que l'écriture dans le registre soit activée par son entrée **en**). La valeur apparaissant à l'intérieur du registre est exprimée en hexadécimal. La sonde sur le fil à droite du registre permet d'afficher cette valeur en décimal. L'entrée des registres est D et la sortie est Q.
- 1 multiplexeur servant à choisir quelle valeur envoyer vers l'entrée I0 de l'UAL : soit le champ DATA ou Adresse, soit le registre A. Ce multiplexeur possède 2 entrées sur 16 bits I0 et I1, 1 entrée de contrôle sur 1 bit C, et 1 sortie sur 16 bits O. Si la valeur de l'entrée C est à 0 alors c'est son entrée I0 (DATA ou Adresse) qui est envoyée vers sa sortie O, sinon c'est I1 (Registre A).

Observez bien le circuit puis répondez aux questions suivantes :

Sur combien de bits est codée une instruction ?

Une instruction est codée sur 24 bitsExemple 140,000 en hexa = 0001 0100 0000

Sur combien de bits est codé le champ DATA ou adresse ?

Sur 16 bits

Quelles sont les positions, dans le code de l'instruction, des bits du champ DATA ou adresse (0 étant la position du bit de poids le plus faible) ?

De 0 à 15

Sur combien de bits est codé le code opératoire ?

Sur 8 bits

Quelles sont les positions dans le code de l'instruction, des bits du code opératoire (0 étant la position du bit de poids le plus faible) ?

Le code opératoire est codé de la postition 16 à 23.

Indiquez le rôle de chacun des bits du code opératoire

| Position | Nom | Rôle |
|---|---|---|
| Bit 0 | JMP | Jump fait un saut reinitialise le compteur |
| Bit 1 | Registre B | Ecriture dans B |
| Bit 2 | Registre A | Ecriture dans A |
| Bit 3 | Mux | l'opération qui consiste à faire circuler sur un seul conducteur, des infor |
| Bits 4,5,6,7 | Alu | Ecriture del la mémoire |

Ce microprocesseur supporte (entre autres) les instructions suivantes :

| Instruction (mnémonique assembleur) | Signification |
|---|---|
| LOAD_A #valeur | Copie une valeur dans le registre A |
| LOAD_B #valeur | Copie une valeur dans le registre B |
| LOAD_A_B | Copie le registre B dans le registre A |
| LOAD_B_A | Copie le registre A dans le registre B |
| ADD_A_B | Additionne les registres A et B et stocke le résultat dans le registre A |
| ADD_B_A | Additionne les registres A et B et stocke le résultat dans le registre B |
| NOT_A | Inverse (complément à 1) le contenu du registre A |
| NOT_B | Inverse (complément à 1) le contenu du registre B |
| INC_B | Incrémente le registre B |
| JMP <label> | Saut à l'instruction étiquetée par <label> |

**Remarque** : dans toutes les instructions, le premier registre est le registre destination.

**Sans utiliser la simulation, mais en analysant uniquement le code binaire de chaque instruction**, donnez en assembleur le programme contenu dans la mémoire. Que fait ce programme ?

| Adresse mémoire | Contenu mémoire (en héxa) | Code Instruction (en binaire) | Instruction (mnémonique assembleur) |
|---|---|---|---|
| 0000 | 140000 | 0001 0100 0000 0000 0000 0000 | LOAD_A #0 |
| 0001 | 12000a | 0001 0010 0000 0000 0000 1010 | LOAD_B #a |
| 0002 | 5c0000 | 0101 1100 0000 0000 0000 0000 | ADD_A_B |
| 0003 | 010002 | 0000 0001 0000 0000 0000 0010 | JMP<adresse2> |
| Rôle du programme | | On met le résultat de l'addition de A et B dans A | |

**Vérifiez en faisant une simulation** : cliquez sur la main dans la barre d'outils, puis cliquez à plusieurs reprises sur l'horloge (ou appuyez sur CTRL-T). Pour réinitialiser la simulation appuyez sur CTRL+R

**Modifiez le programme contenu dans la mémoire pour avoir le résultat dans B au lieu de A**. Cliquez avec le bouton droit sur la mémoire puis choisir **Edit Contents**. Entrez ensuite le code héxa de chaque instruction dans la bonne case mémoire. La première colonne indique l'adresse de la première case située dans la 2ème colonne (les adresses des cases suivantes dans une même ligne seront déduites mentalement). Testez le bon fonctionnement de votre programme à l'aide d'une simulation.

| Adresse mémoire | Contenu mémoire (en héxa) | Code Instruction (en binaire) | Instruction (mnémonique assembleur) |
|---|---|---|---|
| 0000 | 140000 | 0001 0100 0000 0000 0000 0000 | LOAD_A #0 |
| 0001 | 12000a | 0001 0010 0000 0000 0000 1010 | LOAD_B #10 |
| 0002 | 5a0000 | 0101 1010 0000 0000 0000 0000 | ADD_B_A |
| 0003 | 010002 | 0000 0001 0000 0000 0000 0010 | JUMP<adresse2> |

**Ecrire un programme** qui charge dans le registre A une valeur de votre choix puis calcule dans le registre B son complément à 2 (complément à 1 + 1).

| Adresse mémoire | Contenu mémoire (en héxa) | Code Instruction (en binaire) | Instruction (mnémonique assembleur) |
|---|---|---|---|
| 0000 | 14000c | 0001 0100 0000 0000 0000 1100 | LOAD_A #12 |
| 0001 | 1a0000 | 0001 1010 0000 0000 0000 0000 | LOAD_B_A |
| 0002 | c20000 | 1100 0010 0000 0000 0000 0000 | NOT_B |
| 0003 | 320000 | 0011 0010 0000 0000 0000 0000 | INC_B |

# 2. Microprocesseur avec sauts conditionnels

Chargez le circuit **micro2.circ**

Sur combien de bits est codée une instruction ?

Sur 32 bits

Sur combien de bits est codé le code opératoire ?

Sur 16 bits

Sur combien de bits est codé le champ DATA ou adresse ?

Sur 16 bits

A quoi servent les 4 bits JMPZ, JMPNZ, JMPN, JMPPZ ?

| Bit | Rôle |
|---|---|
| JMPZ | saute si le dernier resultat est nul |
| JMPNZ | dernier resultat pas nul |
| JMPN | dernier negatif |
| JMPPZ | dernier positif ou nul |

**Donnez les codes binaires et hexa** des instructions suivantes :

| Instruction | Code Binaire | Code Héxa |
|---|---|---|
| NOP | 0000 0000 0000 0000 0000 0000 0000 0000 | 00000000 |
| LOAD_A #valeur | 0000 0000 0001 0100 0000 0000 0000 0010 | 01400002 |
| LOAD_B_A | 0000 0000 0001 1010 0000 0000 0000 0000 | 002A0000 |
| MUL_A_B | 0000 0000 1000 1100 0000 0000 0000 0000 | 008C0000 |
| DEC_B | 0000 0000 0100 0010 0000 0000 0000 0000 | 00820000 |
| JMP <label> | 0000 0000 0000 0001 0000 0000 0000 0000 | 00010000 |
| JMPZ <label> | 0000 1000 0000 0000 0000 0000 0000 0000 | 08000000 |
| JMPNZ <label> | 0000 0100 0000 0000 0000 0000 0000 0000 | 04000000 |

En utilisant les instructions ci-dessus, **écrire un programme** qui charge dans le registre A la valeur 5 puis calcule sa factorielle. Vous donnerez 2 versions. La première version utilisera l'instruction JMPNZ. La deuxième version utilisera les instructions JMPZ et JMP. Ecrivez les codes hexa trouvés dans la mémoire puis testez à l'aide de simulations.

**1ère version (JMPNZ)**

| Adresse | Instruction (mnémonique assembleur) | Contenu mémoire (en héxa) |
|---|---|---|
| 0000 | LOAD_A #5 | 0014 0005 |
| 0001 | LOAD_B_A | 001A 0000 |
| 0002 | DEC_B | 0042 0000 |
| 0003 | Début : MUL_A_B | 008C 0000 |
| 0004 | DEC_B | 0042 0000 |
| 0005 | JMPNZ début | 0400 0003 |
| 0006 | NOP | 0000 0000 |

**2ème version (JMPZ et JMP)**

| Adresse | Instruction (mnémonique assembleur) | Contenu mémoire (en héxa) |
|---|---|---|
| 0000 | LOAD_A #5 | 0014 0005 |
| 0001 | LOAD_B_A | 001A 0000 |
| 0002 | Début DEC_B | 0042 0000 |
| 0003 | JMPZ Fin | 0800 0006 |
| 0004 | MUL_A_B | 008C 0000 |
| 0005 | JMP début | 0001 0002 |
| 0006 | fin NOP | 0000 0000 |

# 3. Microprocesseur a 3 registres

Chargez le circuit **micro3.circ**. Ce microprocesseur comporte un registre supplémentaire (le registre C) et un multiplexeur supplémentaire (MUX1). Le code opératoire des instructions utilise 2 bits supplémentaires par rapport au précédent :

- Registre C (bit 13 du code opératoire) : permet d'activer l'écriture du résultat dans le registre C
- MUX1 (bit 12 du code opératoire): permet de choisir quoi envoyer à l'entrée I1 de l'UAL :
  - le registre B si MUX1=0
  - le registre C si MUX1=1

**Trouvez les codes binaires et hexa** des instructions suivantes :

| Instruction | Code Binaire | Code Héxa |
|---|---|---|
| LOAD_A #valeur | 0000 0000 0001 0100 0000 0000 0000 0000 | 00140000 |
| LOAD_B #valeur | 0000 0000 0001 0010 0000 0000 0000 0000 | 00120000 |
| LOAD_A_B | 0000 0000 0010 0100 0000 0000 0000 0000 | 00240000 |
| LOAD_B_C | 0001 0000 0010 0010 0000 0000 0000 0000 | 10220000 |
| LOAD_C_A | 0010 0000 0001 1000 0000 0000 0000 0000 | 20180000 |
| ADD_C_AB | 0010 0000 0101 1000 0000 0000 0000 0000 | 20580000 |
| JMP <label> | 0000 0000 0000 0001 0000 0000 0000 0000 | 00010000 |

En utilisant les instructions précédentes, **écrire un programme** qui charge dans le registre A la valeur 3 et dans le registre B la valeur 5 puis permute le contenu des 2 registres A et B. On utilisera le registre C comme variable temporaire pour effectuer la permutation. Ecrivez les codes hexa trouvés dans la mémoire puis testez à l'aide de simulations.

| Adresse | Instruction assembleur | Code Héxa |
|---|---|---|
| 0000 | LOAD_A #3 | 00140003 |
| 0001 | LOAD_B #5 | 00120005 |
| 0002 | LOAD_C_A | 20180000 |
| 0003 | LOAD_A_B | 00240000 |
| 0004 | LOAD_B_C | 10220000 |

On considère la suite de Fibonacci : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, … dans laquelle chaque élément est obtenu en faisant la somme des 2 éléments précédents (hormis les 2 premiers). **Ecrire un programme** qui charge dans le registre A la valeur 0 et dans le registre B la valeur 1 puis calcule dans le registre C les valeurs de la suite de Fibonacci à l'infini. Ecrivez les codes hexa trouvés dans la mémoire puis testez à l'aide de simulations.

| Adresse | Instruction assembleur | Code Héxa |
|---|---|---|
| 0000 | LOAD_A #0 | 00140000 |
| 0001 | LOAD_B #1 | 00120001 |
| 0002 | boucle ADD_C_AB | 2058 0000 |
| 0003 | LOAD_A_B | 00240000 |
| 0004 | LOAD_B_C | 10220000 |
| 0005 | fin JMP | 00010002 |