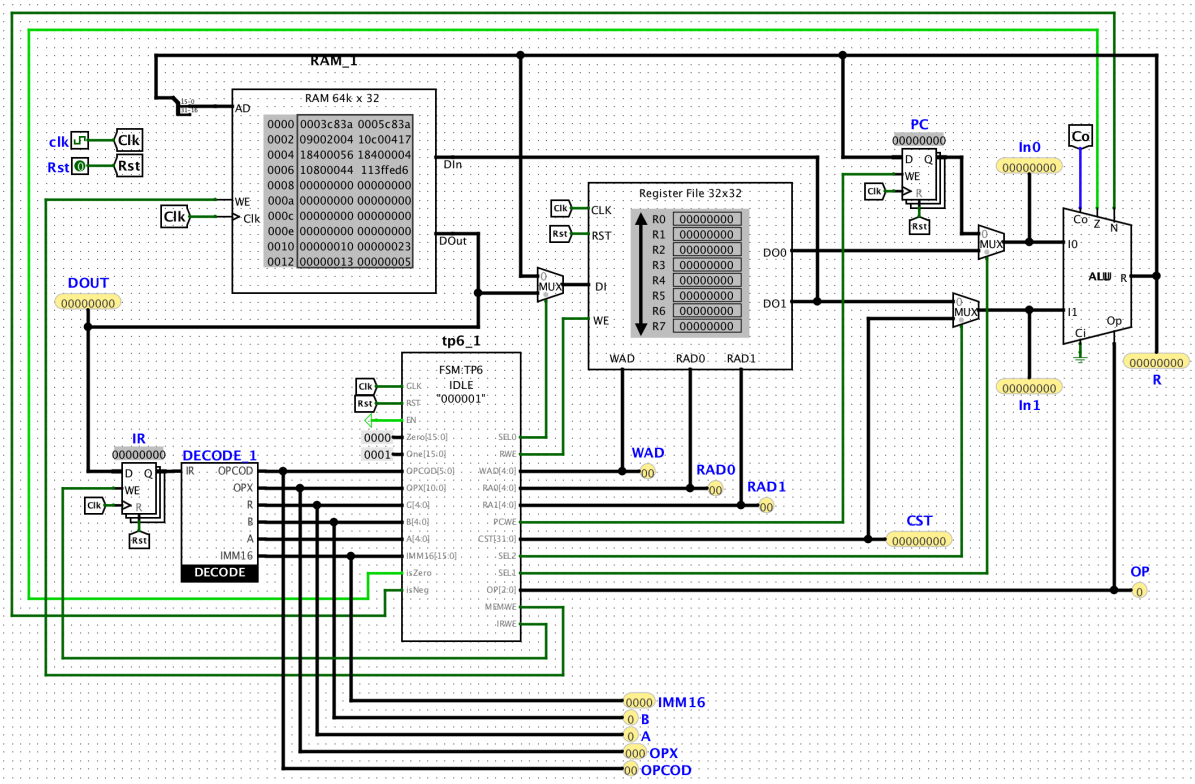
TP6 : processeurs à jeu d'instruction

Dans ce TP on s'intéresse à la mise en œuvre d'un processeur à jeu d'instructions similaire à celui étudié en CM et TD. La mise en œuvre matérielle de ce processeur qui servira de base à ce TP est représentée ci-dessous :



Celle-ci est structurée autour de trois sous-parties :

- Une Unité de traitement qui est chargée de réaliser les traitements et de contrôler les registres.
- Une mémoire RAM qui contient à la fois des données et les instructions du programme. Attention, dans l'exemple étudié en cours, les données et instructions utilisaient des composants mémoire distincts.
- Une Unité de Contrôle réalisée comme une machine à état, qui se charge (i) de piloter les accès à cette mémoire, puis (ii) de séquencer l'exécution des calculs sur l'unité de traitement.

Dans la mise en œuvre qui vous est fournie comme point de départ du TP (fichier TP6.circ), la machine à état de l'UC est incorrecte et incomplète. L'objectif de ce TP est de compléter/corriger le diagramme d'état afin d'obtenir un processeur capable d'exécuter les instructions du programme ci-dessous.

Partie I : analyse d'un programme an langage machine

Le programme exécuté par le processeur est une version légèrement modifiée de celui étudié au TD4. L'algorithme mis en œuvre et sa traduction sont rappelés dans le tableau ci-dessous.

Algorithme	Traduction en instructions machine
`R1=0;`	`R1:=R0-R0; PC:=PC+1;`
`R2=0;`	`R2:=R0-R0; PC:=PC+1;`
`R4=128;`	`R4:=R1+sext(0x80); PC:=PC+1;`
`do {`	
`  R3=MEM[R2+64];`	`R3:=MEM[R2+sext(0x40)]; PC:=PC+1;`
`  if(R3>=R1)`	`if (R3<R1) PC:=PC+1+sext(1); else PC:=PC+1;`
`    R1=R3;`	`R1:=R3+sext(0x0); PC:=PC+1;`
`  R2=R2+1;`	A compléter
`} while(R2<R4)`	`if (R2<R4) PC:=PC+1+sext(-5); else PC:=PC+1;`

Question 1 :

Que fait cet algorithme ?

Question 2 :

Dans l'algorithme de départ, l'instruction **`if`** teste la condition **`R3>=R1`**, et pourtant sa traduction en instruction machine teste la condition **`R3<R1`** qui est la négation logique de la condition de l'instruction **`if`**. Expliquez pourquoi.

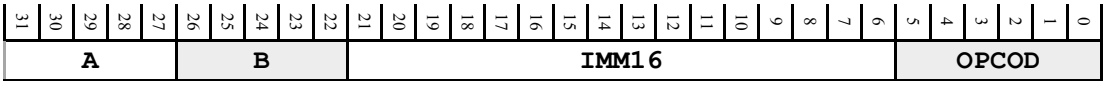
Question 3 :

Complétez le tableau avec l'instruction machine permettant de réaliser **`R2=R2+1`**.

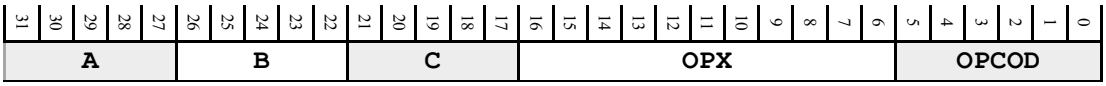
Partie II : représentation en mémoire des instructions machine

Les instructions machines sont représentées en mémoire à l'aide d'un codage sur 32 bits. Celles-ci sont organisées en trois familles (I, R, J) en fonction du nombre et du type d'opérandes, chaque famille disposant de son format propre illustré ci-dessous.

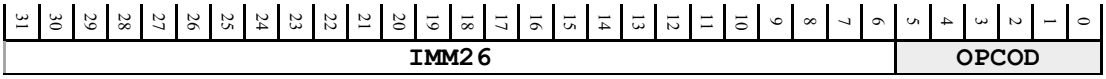
Instructions de format I



Instructions de format R



Instructions de format J



On s'intéresse dans la suite à un sous–ensemble du jeu d'instruction de ce processeur qui est donné ci-dessous.

Instructions de format I

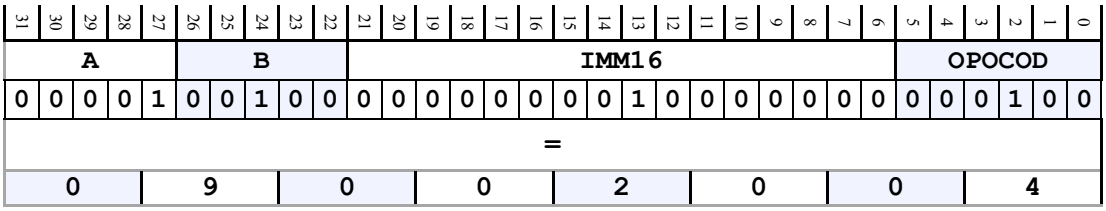
OPCOD[5:0]	Opération
17_h	`REG[B]:=MEM[REG[A]+sext(IMM16)]; PC:=PC+1;`
15_h	`MEM[REG[A]+sext(IMM16)]:=REG[B]; PC:=PC+1;`
04_h	`REG[B]:=REG[A]+sext(IMM16); PC:=PC+1;`
16_h	`if (REG[A]<REG[B]) PC:=PC+1+sext(IMM16); else PC:=PC+1;`

Instructions de format R

OPCOD[5:0]	OPX[10:5]	Opération
$3A_h$	39_h	`REG[C]:=REG[A]-REG[B]; PC:=PC+1;`
$3A_h$	31_h	`REG[C]:=REG[A]+REG[B]; PC:=PC+1;`

Exemple : le codage de l'instruction **`R4:=R1+sext(0x80)`**, est de format I (voir table plus haut). Les instructions de format I utilisent un format à 4 champs : **`A`**, **`B`**, **`IMM16`** et **`OPCOD`**.

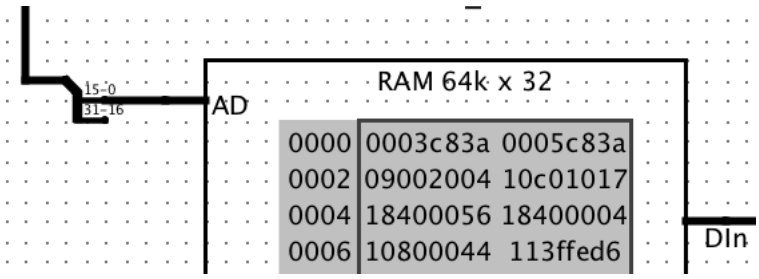
Dans le cas de l'instruction **`R4 :=R1+sext(0x80)`**, ces champs auront donc pour valeurs **`OPCOD=04h`**, **`A=1h`** (pour **`R1`**), **`B=4h`** (pour **`R4`**) et **`IMM16=80h`**. On obtient alors le code 32 bits **`09002004h`**, comme illustré ci-dessous.



Question 4 : quelle est la représentation hexadécimale de l'instruction **`R8:=R4+R5`** ?

Question 5 :

Dans la suite du TP, le processeur va exécuter une série d'instructions stockées en mémoire à partir de l'adresse 0, comme illustré ci-dessus.



Cette séquence d'instructions implémente le programme présenté dans la partie I, selon la correspondance ci-dessous :

Traduction en instructions machine	Codage
`R1:=R0-R0; PC:=PC+1;`	**0003C83A**
`R2:=R0-R0; PC:=PC+1;`	**0005C83A**
`R4:=R1+sext(0x80); PC:=PC+1;`	**09002004**
`R3:=MEM[R2+sext(0x40)]; PC:=PC+1;`	**10C01017**
`if (R3<R1) PC:=PC+1+sext(1); else PC:=PC+1;`	**18400056**
`R1:=R3+sext(0x0); PC:=PC+1;`	**18400004**
?	**10800044**
`if (R2<R4) PC:=PC+1+sext(-5); else PC:= PC+1;`	**113FFED6**

Question 5 : comment pouvez-vous utiliser cette information pour vérifier votre réponse à la question 3 ?

Partie III : fonctionnement du processeur

Le fonctionnement d'un processeur à jeu d'instruction peut se résumer par la séquence d'opérations suivante, qui est répétée à l'infini.

1. Chargement de l'instruction à exécuter : **`IR:=Mem[PC]`**
2. Décodage et exécution des traitements associés à l'instruction
3. Mise à jour du registre **`PC`** (en général **`PC:=PC+1`**)

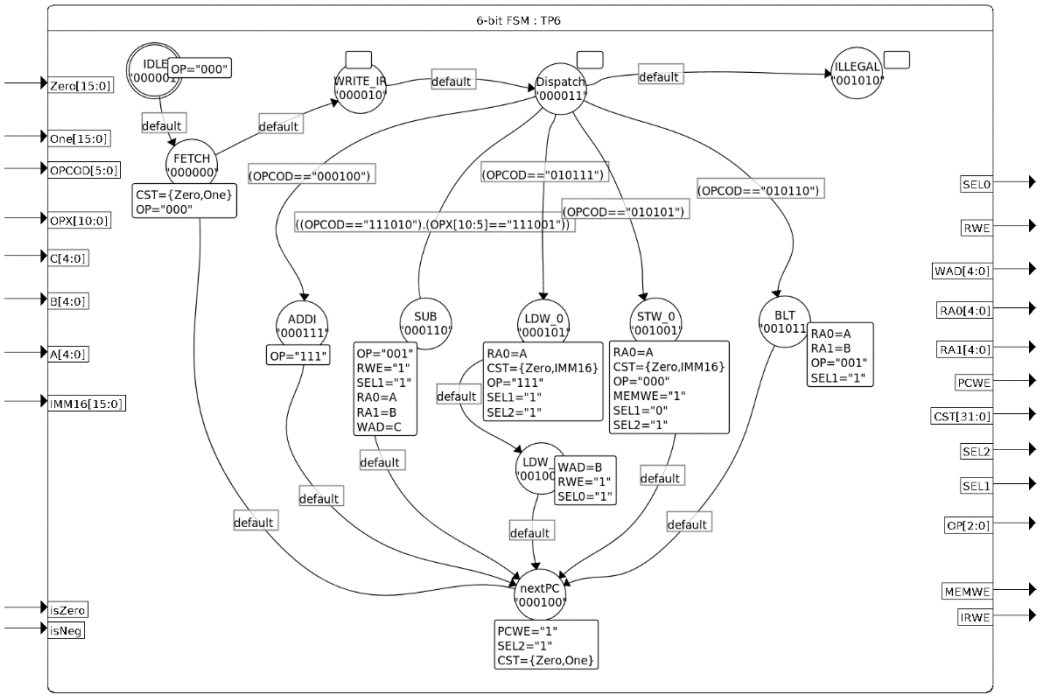
Les questions qui suivent ont pour but de vous faire observer (et comprendre) le fonctionnement du processeur lorsqu'il exécute le programme stocké en mémoire.

Question 6 :

Simulez l'exécution du processeur sur 2 cycles horloges et observez les valeurs des registres **`PC`** et **`IR`**. Que pouvez-vous en conclure concernant le fonctionnement du processeur (indice : le registre **`IR`** contient-il la/les bonne(s) valeur(s) ?)

Question 7 :

Ouvrez dans l'éditeur la machine à état de l'UC du processeur (représentée ci-dessous).



Modifiez le diagramme d'états afin de corriger le problème (indice : commencez par identifiez l'état qui se charge d'écrire dans le registre **`IR`**).

Question 8 :

Continuer à simuler l'exécution du processeur jusqu'au chargement de l'instruction machine **`09002004`** à **`PC=2`** dans le registre **`IR`**. Qu'est supposé faire cette instruction ? Que se passe-t-il lors de son exécution ? Complétez le diagramme d'état de manière à permettre le bon fonctionnement de l'instruction.

Question 8 :

Complétez l'ensemble de la machine à état afin de permettre l'exécution complète du programme.