KINFB40U: Architecture – TP

Conception d'un microprocesseur simplifié en Logisim

II. Première ébauche de processeur

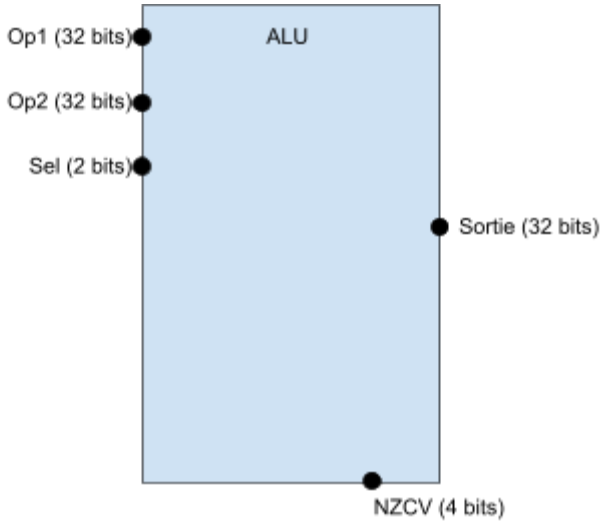
Dans cette partie, nous allons créer la base du processeur, en commençant par les parties nécessaires à l'exécution des opérations de base. Ce TP fonctionne de façon incrémentale : on va commencer par réaliser un circuit relativement simple, capable d'exécuter les fonctionnalités d'instructions de base (MOV, ADD, SUB, AND), puis on va y ajouter un décodeur d'instructions (qui prend en entrée une instruction sous forme binaire et fournit en sortie les valeurs aux signaux qui configurent le processeur), puis une mémoire contenant les instructions composant un programme. On ajoutera ensuite les éléments de gestion des conditions, les instructions de branchement, et enfin la gestion des données en mémoire (STR, LDR, etc.).

II.1. Composants de base

Le processeur est organisé autour de deux composants cruciaux que sont l'**unité arithmétique et logique** (ALU en anglais) et le **banc de registres** (register file en anglais). L'ALU effectue les calculs correspondants aux différentes instructions, et le banc de registre stocke les résultats intermédiaires dans ses registres.
Pour ce TP, ces deux composants sont fournis afin que vous n'ayez pas à les développer vous-même.

II.1.1. ALU

L'ALU fournie possède 3 ports d'entrée et 2 ports de sortie. Les deux premiers ports d'entrée attendent des signaux de taille 32 bits qui seront les opérandes des calculs. Le 3ème port d'entrée attend un signal de taille 2 bits permettant de configurer l'opération à réaliser. Le 1er port de sortie contient le résultat de l'opération en cours d'exécution et le second recueille les valeurs N, Z, C et V (dans cet ordre) pour l'opération en cours.



Port	Taille	E/S	Rôle
Op1	32 bits	E	Recueille le premier opérande de l'opération à réaliser
Op2	32 bits	E	Recueille le second opérande de l'opération à réaliser
Sel	2 bits	E	Sélectionne l'opération à réaliser
Sortie	32 bits	S	Contient le résultat de l'opération
NZCV	4 bits	S	Contient les valeurs N, Z, C et V relatives à l'opération en cours

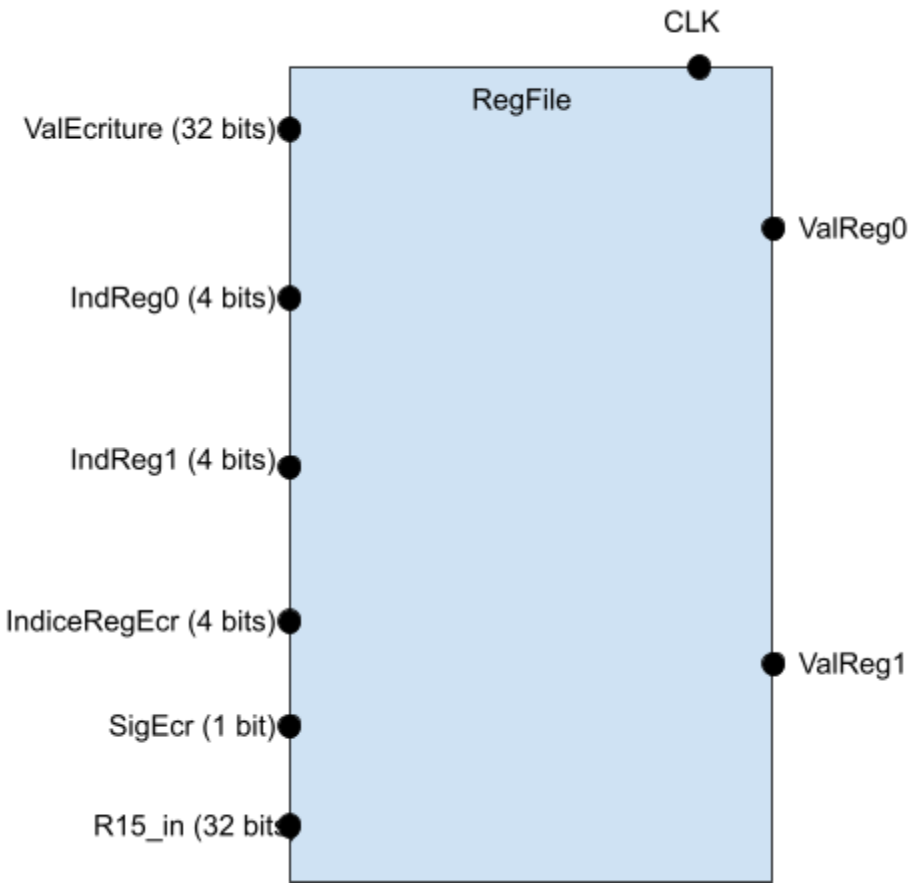
Cette ALU permet d'effectuer **4 opérations** : l'**addition**, la **soustraction**, le **ET bit à bit** et le **transfert direct du second opérande vers la sortie** (utile pour l'instruction MOV).

Opération	Valeur du sélecteur
Addition	10
Soustraction	11
ET	00
Transfert opérande 2	01

II.1.2. Banc de registres

Le banc de registres fourni possède 6 ports d'entrée et 2 ports de sortie. Le premier port d'entrée attend un signal de taille 32 bits contenant la valeur à écrire dans un registre. Les 2ème et 3ème ports attendent des signaux de taille 4 bits contenant l'indice des registres à lire et à transférer sur les ports de sortie du banc. Le 4ème port attend un signal de taille 4 bits contenant l'indice du registre dans lequel écrire la valeur transmise au 1er port. Le 5ème port attend un signal 1 bit indiquant s'il faut ou non écrire un registre et le 6ème port contient le signal d'horloge du processeur (les écritures s'effectuent sur front montant).

Le banc de registre est composé de **16 registres de 32 bits chacun**, comme dans les processeurs ARM vus en cours.



Port	Taille	E/S	Rôle
ValEcriture	32 bits	E	Valeur à écrire dans un registre
IndReg0	4 bits	E	Indice du 1er registre à lire
IndReg1	4 bits	E	Indice du second registre à lire
IndiceRegEcr	4 bits	E	Indice du registre à écrire
SigEcr	1 bit	E	Ecriture ou non
CLK	1 bit	E	Horloge
R15_in	32 bits	E	Nouvelle valeur du PC
ValReg0	32 bits	S	Valeur du registre d'indice IndiceReg0
ValReg1	32 bits	S	Valeur du registre d'indice IndiceReg1

II.1.3. Import de l'ALU et du banc de registres dans Logisim

L'ALU et le banc de registres que nous allons utiliser sont présents dans la librairie Composants.circ disponible sur la page moodle du cours. Afin de charger ces composants, vous devez cliquer dans le menu de logisim sur Project > Load Library >Logisim-evolution library.
L'ALU et le banc de registres sont alors disponibles dans le menu de gauche, à l'intérieur du dossier composants.

II.2. Instruction MOV

II.2.1. MOV registre/immédiat

Dans un premier temps, nous allons concevoir un circuit permettant de réaliser la fonction de l'instruction MOV d'un immédiat vers un registre (par exemple mov r2, #1).
Pour cela on va instancier une ALU et un banc de registres, et relier la sortie de l'ALU à l'entrée **ValEcriture** du banc de registres.

Pour tester ce circuit, nous allons instancier un certain nombre de ports ("pins") d'entrée qui vont nous permettre de fournir les signaux de configuration :

- Un port d'entrée contenant une valeur sur 32 bits, représentant un immédiat à faire passer au banc de registre. On connectera ce port à l'entrée **Op2** de l'ALU.
- Un port d'entrée contenant une valeur sur 2 bits, représentant la configuration de l'opération à effectuer dans l'ALU. Pour tester notre circuit, on mettra ce port à la valeur **01** qui correspond à l'opération mov dans l'ALU

- Un port d'entrée sur 4 bits représentant l'indice du registre dans lequel écrire la valeur immédiate
- Un port d'entrée 1 bit permettant d'activer l'écriture dans le banc de registres
- Un port d'entrée 1 bit servant d'horloge.

Afin de tester la bonne mise à jour des registres, on déclarera également une sonde 32 bits branchée sur la sortie **ValReg1** du banc de registres qui nous permettra de lire en direct la valeur des registres. Pour choisir lequel des registres on souhaite observer, on va également ajouter un port d'entrée de 4 bits connecté à l'entrée **IndReg1** du banc de registres. Notez que **l'utilisation de sondes branchées sur les différents signaux d'un circuit permettent d'afficher les valeurs courantes de ces signaux, et aident à débugger le circuit**.

Certains ports de l'ALU et du banc de registres ne sont pour le moment connectés à rien. Ils le seront par la suite, à mesure que nous ajouterons des fonctionnalités au processeur.

On va commencer par tester le mov reg/immédiat en exécutant l'instruction **mov r2, #1** :

1. Fixez le signal correspondant à la valeur immédiate à la valeur 1, grâce au port d'entrée correspondant
2. Fixez le signal correspondant au registre **à lire** à la valeur 2. On constate que la sortie vaut 0 car le registre r2 a été initialisé à cette valeur (cette étape n'est pas nécessaire pour exécuter le mov, on ne la fait que pour constater l'écriture sur la sortie)
3. Fixez le signal correspondant au registre **à écrire** à la valeur 2. On constate que la sortie vaut toujours 0 : l'écriture dans le registre n'a pas encore été prise en compte. Il faut pour cela jouer sur les signaux de configuration du banc de registres
4. Fixez le signal d'écriture dans le banc à la valeur 1 pour autoriser l'écriture
5. Faites passer le signal d'horloge de la valeur 0 à la valeur 1 pour générer un front montant d'horloge. A ce moment, la sortie doit passer à 1, car le registre 2 a été écrit avec la valeur immédiate fournie.

II.2.2. MOV registre/registre

On veut également être en mesure d'exécuter des mov de registre à registre (par exemple mov r2, r1).
Pour cela on doit être capable de lire la valeur d'un registre et de la passer comme second opérande à l'ALU. Puisque le second opérande de l'ALU doit aussi pouvoir recevoir la valeur provenant d'un immédiat (ce que l'on a fait à la section précédente), on va ajouter un multiplexeur. Ce multiplexeur va permettre de choisir entre un signal correspondant à un immédiat et un signal correspondant au port **ValReg1** du banc de registres (selon que le mov est reg/immédiat ou reg/reg). Sa sortie est connectée au port **Op2** de l'ALU. On branchera le signal contenant un immédiat sur l'entrée 1 du multiplexeur, et le signal provenant du banc de registres sur son entrée 0.

Afin de tester ce circuit, on ajoutera un port d'entrée 1 bit permettant de contrôler le multiplexeur et de sélectionner l'une de ses entrées pour la passer à l'ALU.

On peut maintenant tester le mov reg/reg en exécutant l'instruction **mov r2, r1** :

1. Fixez le signal correspondant au registre **à lire** à la valeur 1. La sortie affiche désormais 0 car le registre r1 a été initialisé à 0
2. Laissez le signal correspondant au registre **à écrire** à la valeur 2 pour écrire dans r2
3. Fixez le signal correspondant au sélecteur du multiplexeur à 0 pour laisser passer la valeur de r2 vers l'ALU
4. Générez un front montant d'horloge. L'écriture a été effectuée. On va procéder à une ultime étape pour vérifier que la valeur de r2 est désormais 0
5. Fixez le signal correspondant au registre **à lire** à la valeur 2. La sortie doit afficher 0 (c'est-à-dire la valeur que nous venons d'écrire dans r2).

II.3. Instruction ADD

Nous allons maintenant compléter ce circuit pour permettre l'exécution de l'instruction add. La principale nouveauté est que l'opération add prend 2 opérandes et non une seule comme mov. Pour ce faire, nous allons relier la sortie **ValReg0** du banc de registres à l'entrée **op1** de l'ALU. Pour tester le circuit, on va ajouter un port d'entrée de 4 bits que nous connecterons à l'entrée **IndReg0** du banc de registres afin de spécifier l'indice du premier opérande de l'instruction.

Notez que l'addition peut prendre un registre ou un immédiat comme second opérande grâce au circuit déjà réalisé pour l'instruction mov.

Nous allons maintenant tester l'instruction add :

1. (préparation) Commencez par exécuter les instructions mov r1, #5 et mov r2, #3
2. Changez la valeur du signal de sélection de l'opération à exécuter à **10** pour effectuer une addition
3. (add r3, r1, #2) Fixez le signal d'entrée correspondant à l'indice du premier opérande à 1 pour lire r1
4. (add r3, r1, #2) Fixez le signal de l'immédiat à la valeur 2 et le signal de contrôle du multiplexeur à la valeur 1 pour laisser passer cette valeur comme second opérande
5. (add r3, r1, #2) Fixez le signal correspondant à l'indice du registre **à écrire** à la valeur 3 pour écrire dans r3
6. (add r3, r1, #2) Générez un front montant avec le signal d'horloge
7. (vérification) Fixez le signal correspondant à l'indice du deuxième registre à lire à la valeur 3 pour afficher la valeur de r3 sur le port de sortie. Cette valeur doit valoir 7
8. (add r0, r2, r3) Fixez les signaux des indices des registres **à lire** à 2 et 3 respectivement, et le signal de l'indice du registre **à écrire** à 0
9. (add r0, r2, r3) Fixez le signal de contrôle du multiplexeur à 0 pour laisser passer la valeur de r3 vers l'ALU
10. (add r0, r2, r3) générez un front montant d'horloge

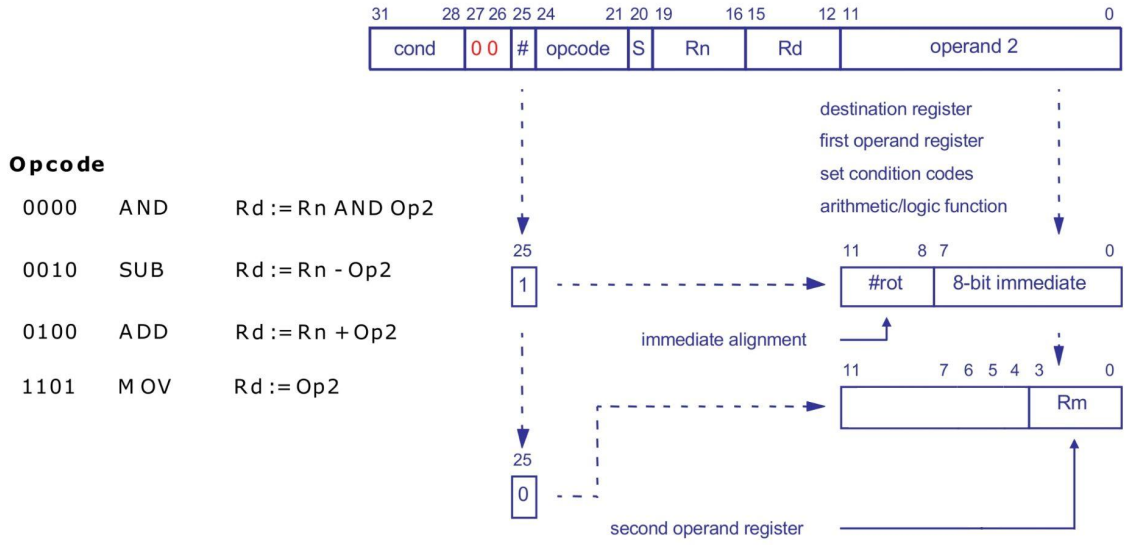
11. (vérification) Fixez le signal correspondant à l'indice du deuxième registre à lire à la valeur 0 pour afficher la valeur de r0 sur le port de sortie. Cette valeur doit valoir 10.

Avec ce circuit, on peut en fait d'ores et déjà exécuter des soustractions et des ET bit à bit en sélectionnant le bon code pour contrôler l'opération de l'ALU. Qui plus est on peut, pour toutes les opérations, sélectionner le deuxième opérande entre un immédiat et un registre en utilisant l'entrée correspondante.

III. Décodeur d'instructions

Nous allons désormais réaliser un composant permettant de décoder les instructions déjà supportées par notre processeur. L'objectif du décodeur est de générer, à partir d'une instruction 32 bits en entrée, tous les signaux de configuration du processeur que nous avons jusqu'ici gérés à la main : indices des registres sources, indice du registre destination, opération à réaliser, choix entre immédiat ou registre pour l'opérande 2.

Afin de simplifier notre circuit, nous n'allons pas réaliser la totalité des instructions du jeu ARM, ni la totalité des options (décalage du second opérande, etc.) de ces instructions. Pour les instructions de traitement de données, on se limitera au schéma suivant :



A partir d'un registre 32 bits en entrée, le décodeur doit produire plusieurs sorties :

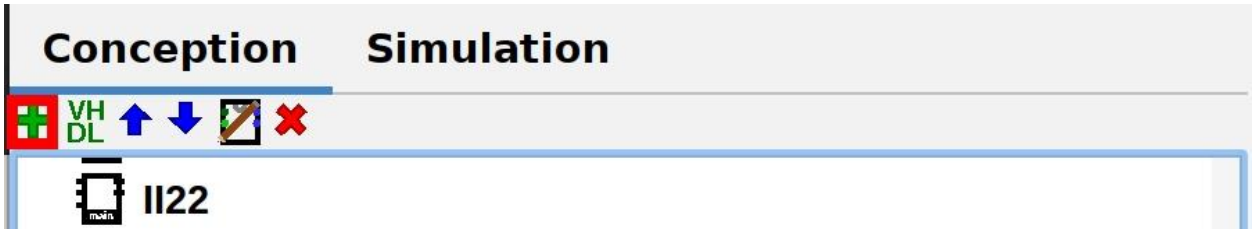
- Un signal 2 bits de sélection de l'opération dans l'ALU. Ce signal est généré en combinant les 4 bits 24 à 21 qui forment l'opcode dans l'instruction (cf partie III.4). On nommera cette sortie **Op**.
- Un signal 4 bits contenant l'indice du premier registre opérande. Ce signal est obtenu en regroupant les bits 19 à 16 de l'instruction. On nommera cette sortie **Rs1**.
- Un signal 4 bits contenant l'indice du registre destination. Ce signal est obtenu en regroupant les bits 15 à 12 de l'instruction. On nommera cette sortie **Rd**.
- Un signal 8 bits contenant le motif de l'immédiat (avant rotation). Ce signal est obtenu en regroupant les bits 7 à 0 de l'instruction. On nommera cette sortie **Imm**.
- Un signal 4 bits contenant la valeur de rotation. Ce signal est obtenu en regroupant les bits 11 à 8 de l'instruction. On nommera cette sortie **Rot**.
- Un signal 4 bits contenant l'indice du second registre opérande. Ce signal est obtenu en regroupant les bits 3 à 0 de l'instruction. On nommera cette sortie **Rs2**.
- Un signal 1 bit permettant de sélectionner si le second opérande souhaité est un registre ou un immédiat. On nommera cette sortie **Imm_reg**.

Le tableau suivant récapitule les ports d'entrées et de sorties du composant décodeur :

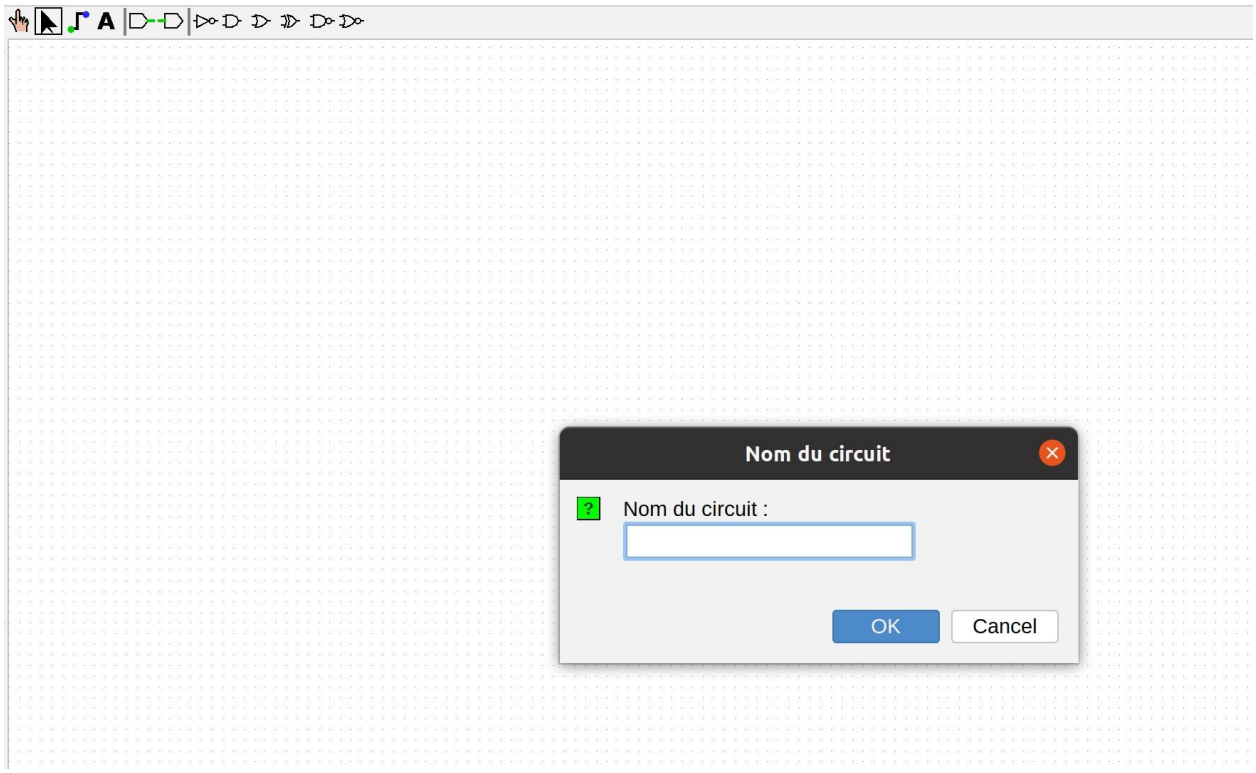
Port	Taille	E/S	Rôle
Instr	32 bits	E	Instruction à décoder
Op	2 bits	S	Code de sélection de l'opération dans l'ALU
Rd	4 bits	S	Indice du registre destination
Rs1	4 bits	S	Indice du registre opérande 1
Rs2	4 bits	S	Indice du registre opérande 2
Imm	8 bits	S	Motif de l'immédiat (avant rotation)
Rot	4 bits	S	Valeur de rotation
Imm_reg	1 bit	S	Sélection immédiat ou registre pour opérande 2

III.1. Déclarer le composant décodeur

Pour déclarer un nouveau composant, on clique sur le + vert dans le menu en haut à gauche :



Dans la fenêtre qui s'ouvre, on tape le nom du composant que l'on souhaite déclarer. Dans notre cas : **decodeur.**



III.2. Ports d'entrées et de sorties

A l'intérieur d'un composant, les ports d'entrée et de sortie sont matérialisés par les composants **broches**, que l'on règle sur entrée ou sortie dans le menu en bas à gauche.

Créez et configurez les broches correspondant aux ports d'entrées et de sorties du composant décodeur, en vous aidant du tableau du début de la section III. Veillez à orienter le port d'entrée d'entrée vers l'Est, et les ports de sortie vers l'Ouest. Pour chacun des ports, remplissez le

champ "**Étiquette**" avec le nom du port (Instr, Op, etc.). Pour des raisons de lisibilité du circuit du processeur, il est conseillé de placer les ports de sortie dans l'ordre suivant de haut en bas dans votre schéma : Rs1, Rs2, Rd, Imm, Rot, Imm_reg, Op.

III.3. Séparer les bits de l'instruction

Afin de pouvoir manipuler les bits de l'instruction séparément les uns des autres, on va utiliser un **répartiteur** (dans l'onglet câblage). Branchez l'entrée du répartiteur sur la broche d'entrée de votre composant. Il vous faut alors configurer le répartiteur :

- La largeur du faisceau est **32 bits**, car l'instruction en entrée fait 32 bits
- Le nombre de terminaisons est **32**, car on souhaite pouvoir accéder à chaque bit séparément

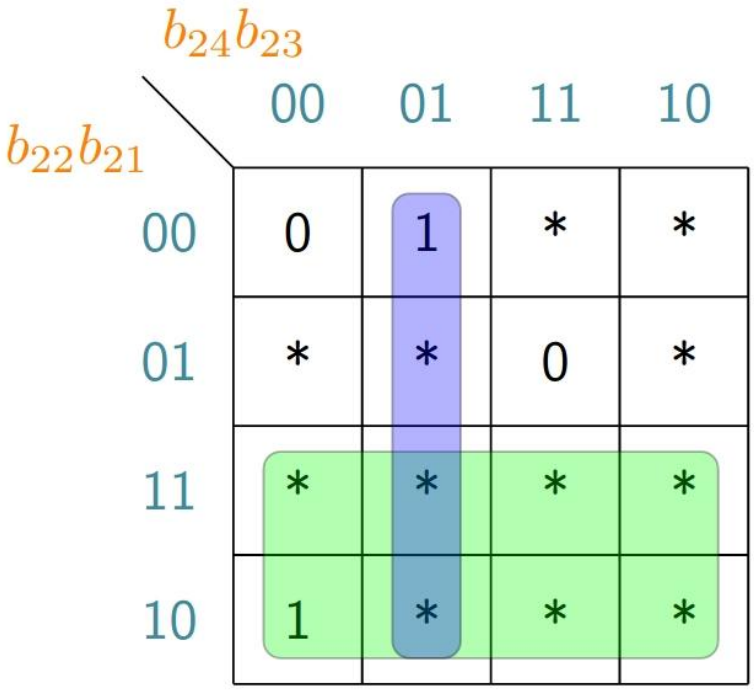
III.4. Générer le signal Op

Les instructions supportées par notre processeur sont ADD (opcode 0100), SUB (opcode 0010), AND (opcode 0000) et MOV (opcode 1101).
Dans l'ALU, le signal de sélection permet de réaliser l'addition (code 10), la soustraction (code 11), le ET logique (code 00) et de passer l'opérande 2 directement (code 01).
La sortie Op du décodeur va être reliée directement à l'entrée de sélection de l'ALU. Nous devons donc réaliser un morceau de circuit transformant les opcodes des opérations dans les codes correspondants reconnus par l'ALU :

Instruction	Opcode ($b_{24}b_{23}b_{22}b_{21}$)	Op (Op_1Op_0)
ADD	0100	10
SUB	0010	11
AND	0000	00
MOV	1101	01

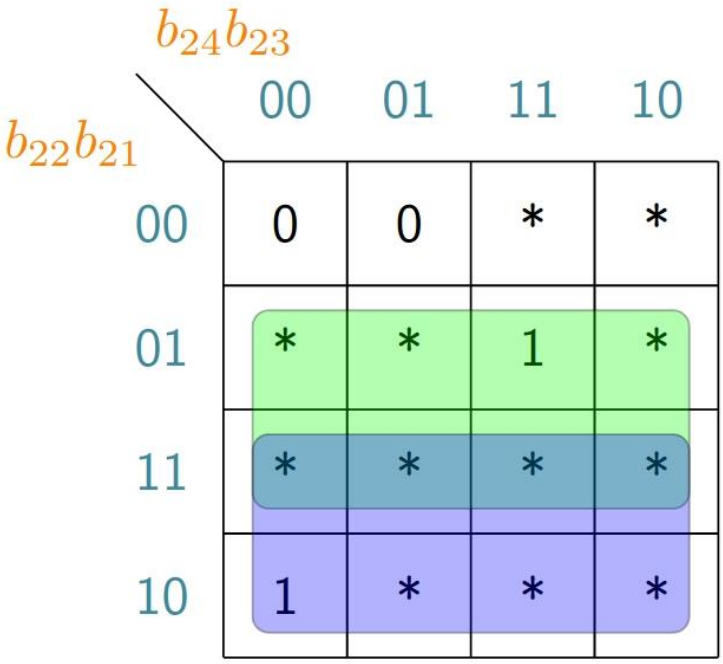
Pour déterminer les équations des bits Op_1 et Op_0 du signal Op, on peut réaliser des tables de Karnaugh.

Pour le bit Op_1 :



On en déduit l'équation : $Op_1 = not(b_{24}).b_{23}+b_{22}$

Pour le bit Op_0 :



On en déduit l'équation : $Op_0 = b_{22}+b_{21}$

En utilisant des portes logiques, dessinez dans le composant décodeur les portions de circuit réalisant ces équations. Joignez ensuite ces deux signaux 1 bit en utilisant un répartiteur à 2 terminaisons (une pour chaque signal 1 bit) et avec un faisceau de largeur 2 bits. Branchez la sortie 2 bits de ce répartiteur sur la broche de sortie Op. Attention à ne pas inverser les 2 signaux lors de leur raccordement au répartiteur.

Testez cette partie du signal en faisant varier les bits 24 à 21 de l'instruction en entrée : faites leur successivement prendre les valeurs de l'addition, de la soustraction, du ET et du MOV, et vérifiez à chaque fois que le signal Op produit correspond bien à ce que l'on attend.

III.5. Générer les signaux des indices registres, rotation et immédiat

A l'aide de répartiteurs, et en vous basant sur le schéma de décodage des instructions de traitement de données, regroupez les bits de l'instruction pour former les signaux attendus en sortie. **Certains bits sont présents dans plusieurs regroupements**.

Testez votre circuit en variant ces différents champs dans l'instruction en entrée et en vérifiant que les sorties générées correspondent bien.

III.6. Générer le signal Imm_reg

Le signal Imm_reg va permettre de sélectionner le second opérande de l'ALU entre un registre et un immédiat. Dans notre processeur, cette sélection s'opère au niveau d'un multiplexeur (cf. Section II.2.2). Ce multiplexeur fait passer le signal provenant du banc de registres lorsque son signal de sélection vaut 0, et un immédiat lorsque son signal de sélection vaut 1. En se référant au schéma de décodage des instructions de traitement des données, on constate que cela correspond aux valeurs du bit 25 de l'instruction (0 -> registre, 1 -> immédiat). Il nous suffit donc de transmettre la valeur du bit 25 de l'instruction directement à la broche Imm_reg.

IV. Exécuter une instruction

IV.1. Instancier un décodeur dans le processeur

Sélectionnez le décodeur dans le menu des composants en haut à gauche et ajoutez le dans le schéma de votre processeur. Procédez ensuite au câblage entre les sorties Rs1, Rs2 et Rd du décodeur et les entrées correspondantes du banc de registre. Branchez ensuite Imm_reg au multiplexeur correspondant et Op à l'ALU. Supprimez au fur et à mesure les broches qui ne sont plus nécessaires.

IV.2. Calcul des immédiats

Les sorties Imm et Rot du décodeur vont nous permettre de calculer la valeur de l'immédiat à passer à l'ALU. On va commencer par étendre le signal Imm qui sort du décodeur d'instructions de 8 à 32 bits. Pour cela on va utiliser un répartiteur à 2 broches: la première broche recevra les 8 bits de poids faible du signal étendu, et la seconde broche recevra les 24 bits de poids fort. On branchera la première broche à la sortie Imm du décodeur d'instructions, et la constante 0 sur 24 bits à la seconde broche.
On instanciera ensuite une unité de rotation (dans le menu Arithmetic/shifter), paramétrée en rotate right, sur 32 bits. Le signal Imm étendu sera branché à sa première entrée. La seconde entrée attend la valeur de rotation. Celle-ci vaut 2 fois la valeur de la sortie rot du décodeur d'instructions. On produira donc le signal correspondant en insérant un bit 0 à droite du signal rot.

IV.3. Test

Pour tester votre circuit, ajoutez une broche 32 bits en entrée du décodeur. On va spécifier le code d'instructions dans cette broche, et simuler leur exécution en générant des fronts montants sur la broche correspondant au signal d'horloge (qui est toujours connectée au banc de registres).
En vous servant du schéma d'encodage de la section III, encodez et exécutez (en générant un front montant d'horloge) successivement les instructions :

- mov r0, #1
- mov r1, #2
- add r2, r0, #3
- add r3, r2, r1
- mov r3, r3

La dernière instruction mov permet d'afficher la valeur de r3 sur la sortie ValReg0 du banc de registres (sur laquelle vous avez normalement déjà attaché une broche de sortie 32 bits pour afficher la valeur du premier registre lu). Vous pouvez tester l'instruction sub et and de la même manière.

V. Mémoire d'instructions

Nous allons désormais rajouter une mémoire (RAM) qui servira à contenir les instructions composant le programme à exécuter. Ces instructions seront directement passées de la mémoire au décodeur d'instructions.

V.1. Instancier la mémoire

Dans le menu des composants, onglet mémoire, sélectionnez et instanciez une RAM. Dans la configuration, modifiez les champs suivants :

- Largeur des données : **32 bits** (on veut lire des instructions codées sur 32 bits)
- Largeur d'adresse : **24 bits** (c'est le maximum dans cette version de logisim. Autrement on aurait opté pour 32 bits)
- Lecture asynchrone : **oui** (on veut pouvoir lire les instructions en permanence)

La RAM dispose de 5 ports d'entrée et d'un port de sortie. Les ports d'entrée comprennent (de haut en bas, dans l'ordre) :

- Le port d'adresse sur 24 bits (on laisse de côté les 2 bits de poids faible de l'adresse 32 bits, puisque les instructions sont toutes alignées sur des adresses multiples de 4, ainsi que les 6 bits de poids fort, puisque logisim ne permet pas d'utiliser des adresses de plus de 24 bits)
- Un port permettant d'activer l'écriture. Nous n'utiliserons pas ce port car nous ne souhaitons pas que le processeur écrive dans cette mémoire
- Un port permettant d'activer la lecture. Nous devons relier ce port à une broche 1 bit **dont la valeur vaudra toujours 1**
- Un port pour recevoir un signal d'horloge, pour réaliser des écritures synchrones. Nous n'utiliserons pas ce port
- Un port 32 bits pour recevoir des valeurs à écrire en mémoire. Nous n'utiliserons pas ce port

Le port de sortie contient en permanence les 32 bits situés en mémoire à l'adresse passée sur le port d'adresse de la RAM (l'instruction présente à cette adresse). On va donc relier ce port à l'entrée du décodeur.

V.2. Écrire un programme en mémoire

Pour écrire un programme en mémoire, il faut faire un clic-droit sur la mémoire et choisir “Modifier le contenu”. Un tableau s’ouvre alors dans lequel on peut écrire les instructions composant un programme, sous forme **hexadécimale**. Il est également possible de charger un programme écrit dans un fichier externe en cliquant en bas de la fenêtre sur le bouton "ouvrir". Un tel fichier doit contenir les instructions composant le programme, sous forme hexadécimale et séparées par des espaces. Après avoir sélectionné le fichier contenant votre programme, cliquez sur "v2.0 raw" pour que le programme soit interprété correctement.

V.3. Gestion simple du PC

C'est le **program counter** (pc / r15) qui contient l'adresse de l'instruction à exécuter au cycle courant. Au sein d'un cycle, le pc doit donc servir à adresser la mémoire d'instructions pour désigner l'instruction à récupérer et à passer au décodeur, puis être incrémenté de 4 pour passer à l'instruction suivante. Cependant, sa valeur ne doit être mise à jour qu'au début du cycle suivant, afin d'assurer que la bonne instruction est exécutée à chaque cycle. Pour ce faire, la solution la plus pratique est d'utiliser un registre 32 bits **en dehors du banc de registres** dans lequel on stocke le pc. Ce registre est mis à jour à chaque front montant d'horloge avec la valeur d'un signal contenant la sortie du registre incrémentée de 4. Ce signal s'obtient avec le composant **Add4** qui prend un signal 32 bits en entrée et fournit en sortie la valeur d'entrée incrémentée de 4, toujours sur 32 bits.
Afin d'assurer la cohérence de valeur entre ce registre et le registre r15 du banc de registres, on branchera la sortie du registre sur l'entrée R15_in du banc de registre.
Enfin, à l'aide d'un répartiteur, on récupèrera dans un signal 24 bits les bits <25...2> du signal de sortie du registre, et on branchera ce signal 24 bits à l'entrée d'adresse de la mémoire d'instructions.

Pour tester, écrivez dans un fichier texte le programme suivant (sous forme hexadécimale) :

```
mov r0, #1
mov r1, #2
add r2, r0, #3
add r3, r2, r1
mov r3, r3
```

Lors de l'exécution de la dernière instruction vous pouvez normalement constater sur la sonde de sortie du banc de registres que la valeur de r3 est correcte (c'est le même programme qu'à la section IV.3).

VI. Gestion des conditions d'exécution

Comme vu en cours, la gestion des conditions d'exécution nécessite quatre éléments :

- Récupérer et décoder la condition de l'instruction en cours d'exécution
- Comparer la condition à la valeur courante du CPSR pour déterminer si l'instruction doit être exécutée
- Gérer la mise à jour ou non du CPSR
- Gérer la mise à jour ou non du banc de registres

Nous allons commencer par créer un composant pour la gestion des conditions. Ce composant a 3 ports d'entrée:

- Un port cond 4 bits contenant la valeur de la condition courante
- Un port NZCV 4 bits contenant la valeur des bits NZCV calculés par l'ALU
- Un port MaJ 1 bit qui détermine si le CPSR doit être mis à jour

Ce composant a un port de sortie 1 bit exec. Exec est à 1 si l'instruction en cours doit être exécutée et à 0 sinon.
En utilisant un registre 4 bits et un multiplexeur, réalisez ce composant (en suivant l'exemple du cours), puis instanciez-le dans le processeur, reliez les entrées sorties et testez l'exécution conditionnelle. Attention, pour le moment nous n'avons pas de support pour l'instruction cmp.

VII. Ajout de l'instruction CMP

En suivant les conseils du cours, ajoutez l'instruction CMP à la liste d'instructions supportées par votre processeur. Rappels : l'instruction CMP réalise la soustraction de ses deux opérandes (l'opérande 1 étant toujours un registre et l'opérande 2 pouvant être un registre ou un immédiat), sauvegarde le CPSR (le bit S est forcément à 1 pour un CMP) et ne sauvegarde pas le résultat de la soustraction.