

Laboratoire d'architecture des ordinateurs semestre printemps 2022

Microarchitecture DECODE

Informations générales

Le rendu pour ce laboratoire sera **par groupe de deux**, chaque groupe devra rendre son travail.

Un rapport intermédiaire vous sera également demandé à la fin du troisième laboratoire (EXECUTE) ainsi qu'un rapport final à la fin du dernier laboratoire (PIPELINE)

Le rendu du laboratoire ne sera pas évalué comme indiqué dans la planification des laboratoires, cependant il permet aux professeurs/assistants d'assurer un suivi de votre travail. Tout retard impactera l'évaluation du rapport intermédiaire ou final.



N'oubliez pas de sauvegarder et d'archiver votre projet à chaque séance de laboratoire

NOTE : Nous vous rappelons que si vous utilisez les machines de laboratoire situées au niveau A, il ne faut pas considérer les données qui sont dessus comme sauvegardées. Si les machines ont un problème nous les remettons dans leur état d'origine et toutes les données présentes sont effacées.

Objectifs du laboratoire

L'objectif principal de ce laboratoire est la réalisation de la partie DECODE d'un processeur simplifié. L'idée sera de développer un système de A à Z afin que vous puissiez faire chaque étape vous-mêmes et ainsi bien comprendre les concepts vus dans la théorie du cours afin de les appliquer dans un cas pratique.

Vous devez rendre les projets Logisim ainsi que les codes en assembleur de cette partie. Une documentation des étapes réalisées sera également présente dans le rapport qui est à rendre pour le processeur.

Outils

Pour ce labo, vous devez utiliser les outils disponibles sur les machines de laboratoire (A07/A09) ou votre ordinateur personnel avec la machine virtuelle fournie par le REDS.

⚠ L'installation de la machine virtuelle doit se faire en dehors des séances de laboratoire afin que vous puissiez profiter de poser des questions pendant le laboratoire. L'installation n'est pas comptée dans les périodes nécessaires à la réalisation de ce laboratoire.

Fichiers

Comme précisé dans le laboratoire précédent, l'objectif sera de reprendre votre bloc fetch afin de construire un processeur de A à Z. Vous pouvez donc continuer de travailler directement dans le même dossier que le laboratoire FETCH. Pensez cependant à créer une sauvegarde afin de pouvoir revenir au début de cette partie si quelque chose devient, selon vous, trop compliqué.

⚠ Le fichier Makefile ne doit pas être modifié!

Workspace

Vous devez télécharger le nouveau workspace fourni sur la page Cyberlearn du cours car quelques corrections ont été apportées à l'ancienne version du workspace. Le nouveau workspace contient aussi la correction du bloc FETCH. Le nouveau workspace devrait contenir :

- Mémoire d'instructions
- Mémoire de données
- Processeur_ARO2
- Contrôleur mémoire
- bloc FETCH

Ainsi que la plupart des entités que vous allez réaliser dans le cadre de ce cours. Certaines sont déjà complétées, ce sont soit des parties qui prennent trop de temps selon nous ou alors qui ne sont pas très constructives à réaliser. Cependant, lorsque vous allez en avoir besoin, nous vous expliquerons leur fonctionnement.

⚠ Respectez l'architecture hiérarchique présentée dans le cours.

Travail demandé

⚠ Le nom des registres diffèrent entre le manuel de référence et la donnée du labo.

Les différences sont données dans le tableau suivant :

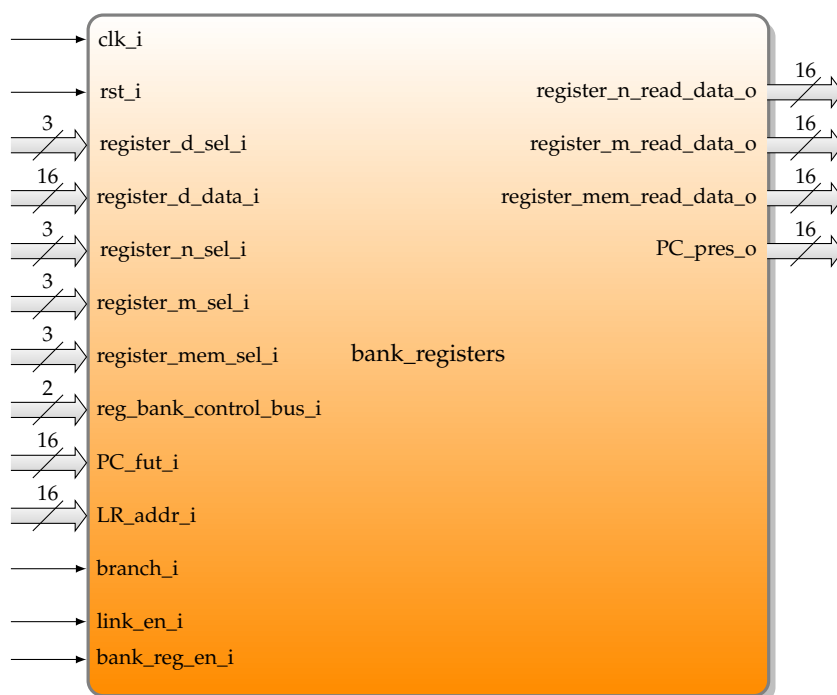
Manuel	Labo
Rd	Rd
Rs	Rn
Rn	Rm

Ce laboratoire est divisé en deux parties :

- La banque de registres
- Le bloc decode

1 Banque de registres

Entité du bloc BANK_REGISTERS



Nom I/O	Description
clk_i	Entrée d'horloge
rst_i	Entrée du reset asynchrone
register_d_sel_i	Sélecteur du registre destination
register_d_data_i	Valeur à mettre dans le registre destination
register_n_sel_i	Sélecteur du registre N
register_m_sel_i	Sélecteur du registre M
register_mem_sel_i	Sélecteur du registre MEM
reg_bank_control_bus_i	Bus de contrôle de la banque de registres
PC_fut_i	Valeur future du Program Counter
LR_addr_i	Valeur à mettre dans le LR pour un retour futur (BL)
branch_i	Flag qu'un saut doit être effectué
link_en_i	Permet le support de l'instruction pour un long saut
bank_reg_en_i	Signal enable de la banque de registre
register_n_read_data_o	Valeur du registre sélectionné par N
register_m_read_data_o	Valeur du registre sélectionné par M
register_mem_read_data_o	Valeur du registre sélectionné par MEM
PC_pres_o	Valeur présente du Program Counter

Construction du bus de contrôle de la banque de registres :

Position	Taille	Description
0	1	Pas utilisé
1	1	Ecriture dans la banque de registres (write enable)

1.1 Instanciation des registres dans la banque

Dans le bloc *bank_registers*, instancier 8 registres pour les données dont 3 registres pour les fonctions spécifiques. Ces registres **doivent avoir** :

- La même "**clock**"
- Le même "**reset**"

Ces registres seront adressés de la façon suivante :

Adress reg	Fonctionnalité
0	R0
1	R1
2	R2
3	R3
4	R4
5	SP (R5)
6	LR (R6)
7	PC (R7)

FIGURE 1 – Registres

Remarque : Les registres 5, 6 et 7 sont ici utilisés pour le Stack Pointer (SP), le Link Register (LR) et le Program Counter (PC) à la place des registres 13, 14 et 15 vu dans le cours. Ceci est dû à une simplification des instructions et l'accès aux registres sur 3 bits.

Pour simplifier le debug durant le développement, il faut nommer les registres et les rendre visibles dans l'onglet registers.

Pour faire cela : Sélectionner un registre et dans les propriétés, "show in registers tab" -> Yes.

Maintenant vous pouvez voir en tout temps l'état de ces registres. Ce qui pourra être utile pour la suite.

1.2 Ecriture dans la banque de registres

Ajouter un bus qui permettra d'écrire la donnée en entrée de la banque de registres dans le registre sélectionné par l'entrée *register_d_sel_i*.

L'adresse de destination sera **décodée** afin de pouvoir utiliser le "write enable" du registre désiré.

1.3 Lecture de 1 registre dans la banque de registres

Lire le registre sélectionné par *register_n_sel_i* et fournir la valeur du registre désiré sur la sortie *register_n_read_data_o*.

1.4 Lecture de 3 registres dans la banque de registres

Ajouter les deux autres bus afin que les sorties *register_m_read_data_o* et *register_mem_read_data_o* soient connectées et permettent de lire les valeurs des registres nécessaires pour les calculs.

1.5 Enable pour la banque de registres

L'enable de la banque de registre (*bank_reg_en_i*) gère l'autorisation de l'écriture de la banque de registres. Si l'enable n'est pas actif, alors aucun registre ne doit pouvoir être modifié.

1.6 Registre Program Counter (PC)

Le registre PC est un registre particulier, comme vu dans le laboratoire Fetch, qui doit être géré différemment. Ce registre PC doit répondre au tableau suivant :

write_en_PC	link_en	PC_reg_input+
0	0	PC_fut_i
0	1	PC_fut_i
1	0	register_d_data_i
1	1	PC_fut_i

Remarque : Il faut que les signaux en entrées de la table considèrent l'enable de la banque de registres.

1.7 Gestion du Link Register (LR)

Vous trouverez un bloc *LR_manager* qui est déjà réalisé. Par exemple, ce bloc permet de gérer l'écriture dans le LR lorsque nous voulons faire un long saut avec un lien et/ou que les 11 bits du saut inconditionnel ne suffisent pas. Il faut instancier ce bloc devant le registre LR et le connecter. Les noms des signaux devraient être assez parlants.

Si vous observez le contenu de ce bloc, vous remarquerez une grande simplicité. Ce bloc permet de choisir si nous stockons la valeur du PC dans le LR afin de préparer un long saut. Sinon, c'est un registre accessible en codant le registre R6.

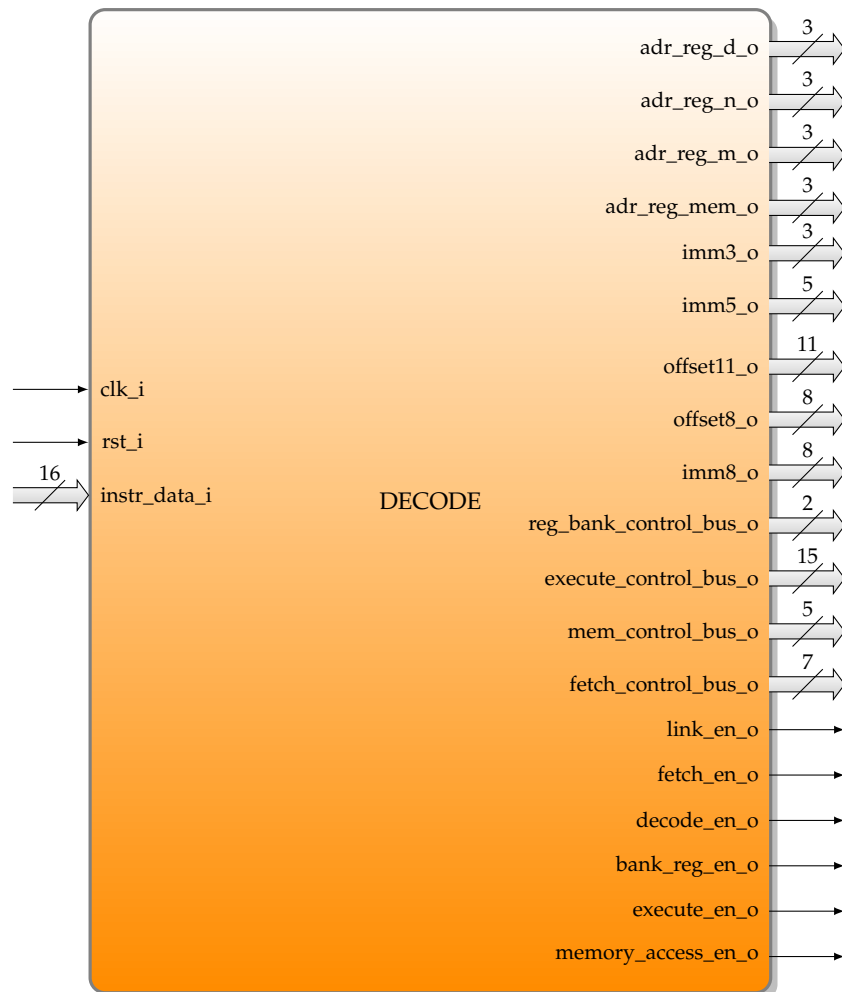
2 Bloc DECODE

⚠ Pour le décodage des instructions, il sera noté :

- Instruction_r_imm : instruction reg, immediate
- Instruction_r_r : instruction reg, reg
- etc ...

Remarque : Vous trouverez sur Cyberlearn la documentation des instructions Thumb correspondantes (ARM7-TDMI-manual-pt3).

Entité du bloc DECODE



Nom I/O	Description
clk_i	Entrée d'horloge
instr_data_i	Instruction à décoder, donnée par le fetch
adr_reg_d_o	Adresse registre destination
adr_reg_n_o	Adresse registre source principal
adr_reg_m_o	Adresse registre source secondaire
adr_reg_mem_o	Adresse registre utilisé pour memory access
imm3_o	Valeur immédiate 3 bits (pour instruction correspondante)
imm5_o	Valeur immédiate 5 bits (pour instruction correspondante)
offset11_o	Valeur d'offset 11 bits (pour instruction correspondante)
offset8_o	Valeur d'offset 8 bits (pour instruction correspondante)
imm8_o	Valeur immédiate 8 bits (pour instruction correspondante)
reg_bank_control_bus_o	Bus de contrôle du bloc registers bank
execute_control_bus_o	Bus de contrôle du bloc execute
mem_control_bus_o	Bus de contrôle du bloc memory access
fetch_control_bus_o	Bus de contrôle du bloc fetch
link_en_o	Autorisation d'écrire dans le LR
fetch_en_o	Enable du bloc fetch
decode_en_o	Enable du bloc decode
bank_reg_en_o	Enable de la banque de registres
execute_en_o	Enable du bloc execute
memory_access_en_o	Enable du bloc memory access

Construction du bus de contrôle du bloc fetch :

Position	Taille	Description
0	1	Sélection PC (équivalent de saut)
1	1	Flag qui indique une instruction conditionnelle
5-2	4	Sélection de la condition à tester
6	1	Pas utilisé

Construction du bus de contrôle du bloc decode (qui ne sort pas du bloc decode) :

Position	Taille	Description
1-0	2	Sélection Rn (source)
2	1	Sélection Rm (source)
3	1	Link enable (Utilisé pour long saut)
4	1	Sélection Rd (destination)
5	1	Pas utilisé

Construction du bus de contrôle du bloc bank_registers :

Position	Taille	Description
0	1	Pas utilisé
1	1	Ecriture dans la banque de registres (write enable)

Construction du bus de contrôle du bloc execute :

Position	Taille	Description
2-0	3	Sélection de l'opération du shifter
5-3	3	Sélection de l'opération de l'unité arithmétique
7-6	2	Sélection opérande 2
8	1	Sélection opérande 1
9	1	Sélection valeur de shift
10	1	Flag qui indique que l'instruction met à jour le CPSR
13-11	3	Sélection valeur du muliplication x 2
14	1	Pas utilisé

Construction du bus de contrôle du bloc memory access :

Position	Taille	Description
0	1	Sélection des données pour memory access
1	1	Flag qui indique qu'on veut sauver une donnée en mémoire
2	1	Flag qui indique qu'on veut lire une donnée en mémoire
3	1	Flag qui indique que l'instruction accède en byte
4	1	Pas utilisé

2.1 Circuit Decode

Dans ce circuit, vous allez devoir implémenter la sélection des adresses **adr_reg_x** en fonction du bus de contrôle du DECODE :

Table de vérité de l'adresse du registre N :

sel_rn	adr_reg_n	Remarque
0	rn_5_3_s	Valeur immédiate depuis le splitter d'instructions
1	adr_reg_d	Copie de la valeur mise sur adr_reg_d_o
2	7	Utilisé lorsque BL msB
3	6	Utilisé lorsque BL lsB


Remarque : L'instruction BL est séparée en deux sous-instructions lors de la compilation et va devoir accéder au registre PC ainsi qu'au registre LR (donc 6 et 7 dans notre cas). Ces valeurs sont donc des constantes car elles sont imposées par notre architecture.

Table de vérité de l'adresse du registre D :

sel_rn	sel_rd	adr_reg_d	Remarque
0	0	rd_2_0_s	Partie de l'instruction à considérer
0	1	rd_10_8_s	Partie de l'instruction à considérer
1	0	rd_2_0_s	Partie de l'instruction à considérer
1	1	rd_10_8_s	Partie de l'instruction à considérer
2	0	6	Utilisé lorsque BL msB
2	1	6	Utilisé lorsque BL msB
3	0	7	Utilisé lorsque BL lsB
3	1	7	Utilisé lorsque BL lsB

Table de vérité de l'adresse du registre M :

sel_rm	adr_reg_m	Remarque
0	rm_8_6_s	Partie de l'instruction à considérer
1	rn_5_3_s	Partie de l'instruction à considérer

 Connecter la sortie *link_en_o* au bus de contrôle du DECODE (bit 3).

2.2 Circuit decode_instr_splitter

Remarque : Analyser et tester.

Cette partie n'a pas besoin d'être modifiée car elle a déjà réalisée mais il est important de comprendre qu'on prépare les diverses informations nécessaires dont nous aurons besoin pour exécuter les instructions que nous voulons supporter.

2.3 Circuit main_control_unit

Remarque : Analyser et tester.

Cette partie n'a pas besoin d'être réalisée car elle a déjà été réalisée. Cette partie sert à fournir **tous** les flags des instructions détectées et ainsi construire les informations qui construiront les bus de contrôle.

2.4 Circuit opcode_supported_unit

Cette partie est partiellement réalisée et vous montre comment les instructions sont détectées.

Ajouter le décodage des instructions suivantes :

- **strb_r_r_r** : Store Byte using 1 data reg, 1 address reg and 1 offset reg
- **ldrb_r_r_r** : Load Byte using 1 data reg, 1 address reg and 1 offset reg
- **strh_r_r_r** : Store Halfword using 1 data reg, 1 address reg and 1 offset reg
- **ldrh_r_r_r** : Load Halfword using 1 data reg, 1 address reg and 1 offset reg
- **strb_r_r_imm** : Store Byte using 1 data reg, 1 address reg and 1 immediate offset
- **ldrb_r_r_imm** : Load Byte using 1 data reg, 1 address reg and 1 immediate offset
- **strh_r_r_imm** : Store Halfword using 1 data reg, 1 address reg and 1 immediate offset
- **ldrh_r_r_imm** : Load Halfword using 1 data reg, 1 address reg and 1 immediate offset

Référez-vous au manuel **ARM7-TDMI-manual-pt3** que vous trouverez sur le site Cyberlearn afin de pouvoir identifier l'opcode des instructions.

2.5 Circuit fetch_control_unit

Compléter les valeurs manquantes du tableau en vous aidant du manuel **ARM7-TDMI-manual-pt3**, puis implémenter cette partie :

Instruction	sel_cpsr	link_en	sel_PC	cond_en
lsl_r_r_imm_s	1	0	0	0
lsr_r_r_imm_s	1	0	0	0
asr_r_r_imm_s	1	0	0	0
add_r_r_r_s	?	?	?	?
add_r_r_imm_s	?	?	?	?
sub_r_r_r_s	1	0	0	0
sub_r_r_imm_s	1	0	0	0
mov_r_imm_s	1	0	0	0
add_r_imm_s	1	0	0	0
sub_r_imm_s	1	0	0	0
and_r_r_s	1	0	0	0
eor_r_r_s	1	0	0	0
lsl_r_r_s	1	0	0	0
lsr_r_r_s	1	0	0	0
asr_r_r_s	1	0	0	0
ror_r_r_s	1	0	0	0
neg_r_r_s	1	0	0	0
orr_r_r_s	1	0	0	0
mvn_r_r_s	1	0	0	0
strb_r_r_r_s	0	0	0	0
ldrb_r_r_r_s	?	?	?	?
strh_r_r_r_s	0	0	0	0
ldrh_r_r_r_s	0	0	0	0
strb_r_r_imm_s	0	0	0	0
ldrb_r_r_imm_s	0	0	0	0
strh_r_r_imm_s	0	0	0	0
ldrh_r_r_imm_s	0	0	0	0
b_cond_s	0	0	1	1
b_incond_s	0	0	1	0
bl_msb_s	0	1	0	0
bl_lsb_s	0	0	1	0

Explications des signaux qui sortent de ce bloc :

- **sel_cpsr_o** : Flag qui met à jour le CPSR avec l’instruction courante
- **link_en_o** : Flag qui permet de sauver la valeur de PC dans le LR pour un long saut
- **sel_PC_o** : Flag qui indique qu’on va écrire dans le PC
- **cond_en_o** : Flag qui indique l’utilisation de la condition

2.6 Circuit decode_control_unit

Compléter les valeurs manquantes du tableau en vous aidant du manuel **ARM7-TDMI-manual-pt3**, puis implémenter cette partie :

Instruction	sel_rn(0)	sel_rn(1)	sel_rm	sel_rd
lsl_r_r_imm_s	0	0	0	0
lsr_r_r_imm_s	0	0	0	0
asr_r_r_imm_s	0	0	0	0
add_r_r_r_s	0	0	0	0
add_r_r_imm_s	?	?	?	?
sub_r_r_r_s	0	0	0	0
sub_r_r_imm_s	0	0	0	0
mov_r_imm_s	?	?	?	?
add_r_imm_s	1	0	0	1
sub_r_imm_s	1	0	0	1
and_r_r_s	1	0	1	0
eor_r_r_s	1	0	1	0
lsl_r_r_s	1	0	1	0
lsr_r_r_s	1	0	1	0
asr_r_r_s	1	0	1	0
ror_r_r_s	?	?	?	?
neg_r_r_s	0	0	0	0
orr_r_r_s	1	0	1	0
mvn_r_r_s	0	0	0	0
strb_r_r_r_s	0	0	0	0
ldrb_r_r_r_s	0	0	0	0
strh_r_r_r_s	0	0	0	0
ldrh_r_r_r_s	0	0	0	0
strb_r_r_imm_s	0	0	0	0
ldrb_r_r_imm_s	0	0	0	0
strh_r_r_imm_s	0	0	0	0
ldrh_r_r_imm_s	0	0	0	0
b_cond_s	0	1	0	0
b_incond_s	0	1	0	0
bl_msb_s	0	1	0	0
bl_lsb_s	1	1	0	0

Explications des signaux qui sortent de ce bloc :

- **sel_rn_o** : bus 2 bits pour choisir Rn
- **sel_rm_o** : sélection Rm
- **sel_rd_o** : sélection Rd

2.7 Circuit reg_bank_control_unit

Compléter les valeurs manquantes du tableau en vous aidant du manuel **ARM7-TDMI-manual-pt3**, puis implémenter cette partie :

Instruction	reg_bank_wr
lsl_r_r_imm_s	?
lsr_r_r_imm_s	1
asr_r_r_imm_s	1
add_r_r_r_s	1
add_r_r_imm_s	1
sub_r_r_r_s	1
sub_r_r_imm_s	1
mov_r_imm_s	1
add_r_imm_s	1
sub_r_imm_s	1
and_r_r_s	1
eor_r_r_s	?
lsl_r_r_s	1
lsr_r_r_s	1
asr_r_r_s	1
ror_r_r_s	1
neg_r_r_s	1
orr_r_r_s	1
mvn_r_r_s	1
strb_r_r_r_s	0
ldrb_r_r_r_s	1
strh_r_r_r_s	?
ldrh_r_r_r_s	1
strb_r_r_imm_s	0
ldrb_r_r_imm_s	1
strh_r_r_imm_s	0
ldrh_r_r_imm_s	1
b_cond_s	0
b_incond_s	0
bl_msb_s	1
bl_lsb_s	1

Explications des signaux qui sortent de ce bloc :

- **reg_bank_wr_o** : Flag qui indique une écriture dans un registre

2.8 Circuit execute_control_unit

Implémenter cette partie en correspondant au tableau suivant :

Instruction	sel_operand_1	sel_operand_2	sel_op_shift	sel_op_alu	sel_mult_2
lsl_r_r_imm_s	0	0	2	0	0
lsr_r_r_imm_s	0	0	3	0	0
asr_r_r_imm_s	0	0	1	0	0
add_r_r_r_s	0	0	0	1	0
add_r_r_imm_s	0	1	0	1	0
sub_r_r_r_s	0	0	0	2	0
sub_r_r_imm_s	0	1	0	2	0
mov_r_imm_s	1	3	0	1	0
add_r_imm_s	0	3	0	1	0
sub_r_imm_s	0	3	0	2	0
and_r_r_s	0	0	0	3	0
eor_r_r_s	0	0	0	7	0
lsl_r_r_s	0	0	2	0	0
lsr_r_r_s	0	0	3	0	0
asr_r_r_s	0	0	1	0	0
ror_r_r_s	0	0	4	0	0
neg_r_r_s	0	0	0	6	0
orr_r_r_s	0	0	0	4	0
mvn_r_r_s	0	0	0	5	0
strb_r_r_r_s	0	0	0	1	4
ldrb_r_r_r_s	0	0	0	1	4
strh_r_r_r_s	0	0	0	1	0
ldrh_r_r_r_s	0	0	0	1	0
strb_r_r_imm_s	0	2	0	1	4
ldrb_r_r_imm_s	0	2	0	1	4
strh_r_r_imm_s	0	2	0	1	0
ldrh_r_r_imm_s	0	2	0	1	0
b_cond_s	0	2	0	1	2
b_incond_s	0	2	0	1	1
bl_msb_s	0	2	0	1	5
bl_lsb_s	0	2	0	1	3

Explications des signaux qui sortent de ce bloc :

- **sel_operand_1_o** : Sélection opérand 1 de l'execute
- **sel_operand_2_o** : Sélection opérand 2 de l'execute
- **sel_op_shift_o** : Sélection opération du shifter
- **sel_op_alu_o** : Sélection opération de l'ALU
- **sel_mult_2_o** : Sélection pour multiplicateur de l'execute
- **sel_shift_o** : Sélection de la valeur de shift

Pour *sel_shift_o*, on obtient :

$$sel_shift_o = lsl_r_r_s + lsr_r_r_s + asr_r_r_s + ror_r_r_s$$

Cette partie doit être analysée par vos soins. Les valeurs sont données en **décimales**. A vous de séparer correctement les équations pour pouvoir mettre la logique désirée dans le bloc.

2.9 Circuit memory_access_control_unit

Compléter les valeurs manquantes du tableau en vous aidant du manuel **ARM7-TDMI-manual-pt3**, puis implémenter cette partie :

Instruction	sel_mem	str_data	ldr_data	byte
lsl_r_r_imm_s	0	0	0	0
lsr_r_r_imm_s	0	0	0	0
asr_r_r_imm_s	0	0	0	0
add_r_r_r_s	0	0	0	0
add_r_r_imm_s	0	0	0	0
sub_r_r_r_s	0	0	0	0
sub_r_r_imm_s	0	0	0	0
mov_r_r_imm_s	0	0	0	0
add_r_imm_s	0	0	0	0
sub_r_imm_s	0	0	0	0
and_r_r_s	?	?	?	?
eor_r_r_s	0	0	0	0
lsl_r_r_s	0	0	0	0
lsr_r_r_s	0	0	0	0
asr_r_r_s	0	0	0	0
ror_r_r_s	0	0	0	0
neg_r_r_s	0	0	0	0
orr_r_r_s	0	0	0	0
mvn_r_r_s	0	0	0	0
strb_r_r_r_s	1	1	0	1
ldrb_r_r_r_s	1	0	1	1
strh_r_r_r_s	1	1	0	0
ldrh_r_r_r_s	?	?	?	?
strb_r_r_imm_s	?	?	?	?
ldrb_r_r_imm_s	1	0	1	1
strh_r_r_imm_s	1	1	0	0
ldrh_r_r_imm_s	1	0	1	0
b_cond_s	0	0	0	0
b_incond_s	0	0	0	0
bl_msb_s	0	0	0	0
bl_lsb_s	0	0	0	0

Explications des signaux qui sortent de ce bloc :

- **sel_mem_o** : Flag qui indique que le bloc memory access est utilisé
- **str_data_o** : Flag qui indique qu'on veut sauver une donnée en mémoire
- **ldr_data_o** : Flag qui indique qu'on veut lire une donnée en mémoire
- **byte_o** : Flag qui indique un accès par Byte

2.10 Circuit bus_constructor

Remarque : Analyser et tester.

Cette partie sert à construire les différents bus en suivant les tables fournies dans l'énoncé.

2.11 Programme qui teste les instructions supportées

Réaliser le code assembleur qui utilise toutes les instructions supportées.

⚠ Observer que les valeurs sélectionnées pour le registre ainsi que les informations données en sorties du decode soient cohérentes.

Voici la liste des instructions supportées :

- lsl_r_r_imm_s
- lsr_r_r_imm_s
- asr_r_r_imm_s
- add_r_r_r_s
- add_r_r_imm_s
- sub_r_r_r_s
- sub_r_r_imm_s
- mov_r_imm_s
- add_r_imm_s
- sub_r_imm_s
- and_r_r_s
- eor_r_r_s
- lsl_r_r_s
- lsr_r_r_s
- asr_r_r_s
- ror_r_r_s
- neg_r_r_s
- orr_r_r_s
- mvn_r_r_s
- strb_r_r_r_s
- ldrb_r_r_r_s
- strh_r_r_r_s
- ldrrh_r_r_r_s
- strb_r_r_imm_s
- ldrb_r_r_imm_s
- strh_r_r_imm_s
- ldrrh_r_r_imm_s
- b_cond_s
- b_incond_s
- bl_msb_s
- bl_lsb_s

Vous n'êtes pas obligés de faire toutes les instructions mais au moins de prendre des instructions avec diverses fonctionnalités. Par exemple, des accès mémoire, des sauts, et quelques instructions qui utilisent des registres, des valeurs immédiates.

2.12 Phase de test

Charger votre programme dans la mémoire d’instruction comme vu lors du laboratoire fetch.
Exécuter et observer les sorties et vérifier vos résultats.

Par exemple que les registres de destination et sources ont les bonnes valeurs par rapport aux instructions choisies.

Exemple :

```
add R0, R1, R2
```

- `adr_reg_d = 0`
- `adr_reg_n = 1`
- `adr_reg_m = 2`
- etc ...

Rendu

Pour ce laboratoire, vous devez rendre :

- votre fichier *.circ*
- votre programme *main.S* contenant le code de test que vous avez utilisé pour tester votre circuit.

Votre rendu ne sera pas évalué comme indiqué dans la planification des laboratoires, cependant il permet aux professeurs/assistants d’assurer un suivi de votre travail. Tout retard impactera l’évaluation du rapport intermédiaire ou final.

CONSEIL : Faire une petite documentation sur cette partie vous fera directement un résumé pour l’examen.