Projet (à faire en binôme)

Avertissement

Ce projet est à faire en binôme et à rendre en 2 semaines. Toute tentative de fraude avérée sera sanctionnée par un 0 pour le binôme "copieur" et par une division de la note par 2 pour le binôme "copié". Dans le cas où il est impossible de déterminer qui a copié sur qui, les 2 binômes se verront attribuer un 0.

Introduction

Le circuit **projet.circ** est celui d'un microprocesseur qui a les caractéristiques suivantes :

- Instructions sur 32 bits
- Code opératoire sur 16 bits. Adresses ou données sur 16 bits
- Les données sortantes de l'ALU peuvent être stockées dans un des quatre registres 16 bits (A, B, SI ou DI) ou dans une RAM des données.
- L'entrée I0 de l'UAL peut prendre ses données soit à partir du Registre A, soit depuis le champ DATA ou Adresse du code de l'instruction et ceci en fonction de la valeur du bit MUX0.
- L'entrée I1 de l'UAL peut prendre ses données soit à partir du Registre B, soit à partir du Registre SI, soit à partir du registre DI, soit à partir de la RAM des données en fonction de la valeur des 2 bits de contrôle du MUX1 conformément à la table ci-dessous :

Bits de contrôle du MUX1		**Registre sélectionné**
Bit 1	**Bit 0**	
0	0	B
0	1	SI
1	0	DI
1	1	RAM des données

- Le circuit comporte un décodeur permettant de choisir le registre où sera écrit le résultat de l'UAL selon la table ci-dessous :

Bits de contrôle du décodeur			**Registre sélectionné**
Bit 2	**Bit 1**	**Bit 0**	
0	0	0	Aucun
0	0	1	A
0	1	0	B
0	1	1	SI
1	0	0	DI
1	0	1	RAM des données
1	1	0	Aucun
1	1	1	Aucun

- Ce circuit comporte 2 mémoires : une pour les instructions et une autre pour les données. Il est fait de telle sorte qu'il supporte un **adressage indirect** à partir des registres SI et DI, c'est-à-dire qu'il est possible d'utiliser une donnée (de la mémoire des données) dont l'adresse se trouve soit dans le registre SI soit dans le registre DI en fonction de la valeur du bit MUX2.

Question 1

Charger le fichier **projet.circ**. Analysez le circuit et aidez-vous des TPs précédents pour comprendre son fonctionnement. Il n'existe pas de fichiers **PROJET.MAD** et **PROJET.PAR**, c'est à vous de les créer en utilisant le logiciel **MAG**. Commencez par créer les instructions suivantes :

LOADA #valeur	Charge dans le registre A, la valeur immédiate qui suit
LOADSI #valeur	Charge dans le registre SI, la valeur immédiate qui suit
LOADADI	Copie dans le registre A, le contenu du registre DI
LOADAADRSI	Charge dans le registre A, la donnée de la RAM des données dont l'adresse se trouve dans le registre SI
LOADBADRDI	Charge dans le registre B, la donnée de la RAM des données dont l'adresse se trouve dans le registre DI
LOADDIADRSI	Charge dans le registre DI, la donnée de la RAM des données dont l'adresse se trouve dans le registre SI
INCSI	Incrémente SI de 1
DECDI	Décrémente DI de 1
CMPSIA	Compare SI et A et met à jour les indicateurs en conséquence (l'UAL fait en interne la soustraction SI-A pour mettre à jour les indicateurs)
CMPBA	Compare B et A et met à jour les indicateurs en conséquence (l'UAL fait en interne la soustraction B-A pour mettre à jour les indicateurs)
JMP <label>	Effectue un saut inconditionnel à l'instruction étiquetée par <label>
JMPNZ <label>	Effectue un saut à l'instruction étiquetée par <label> si le résultat de l'instruction précédente n'est pas nul
JMPPZ <label>	Effectue un saut à l'instruction étiquetée par <label> si le résultat de l'instruction précédente n'est pas négatif (positif ou nul)

Question 2

En utilisant **uniquement les instructions précédentes**, écrire un programme en assembleur, **palin.asm,** qui vérifie si la chaîne présente dans la mémoire des données est un palindrome. La première case de la RAM des données contiendra la longueur de la chaîne. Les cases suivantes contiendront les codes ASCII des caractères de la chaîne. Le programme devra mettre à la fin dans le **registre A** la **valeur 1 si le mot est un palindrome** et la **valeur -1 si ce n'est pas un palindrome**.

Votre programme devra obligatoirement se terminer par l'instruction suivante : **fin** **JMP fin**

Votre programme ne devra pas dépasser plus de **17 instructions**. Toute instruction supplémentaire entraînera un **malus de -0,5 pts**. Si votre programme fonctionne **parfaitement** (dans tous les cas) avec moins de 17 instructions vous recevrez un **bonus de +0,5 pts** par instruction économisée.

Vérifiez la bonne définition de vos instructions et le bon fonctionnement de votre programme dans LOGISIM. On veillera à avoir un **jeu d'essai le plus complet possible** (mot vide, mot d'1 seul lettre, mot avec un nombre pair de lettres, mot avec un nombre impair de lettres, …). On considérera **qu'un mot vide est un palindrome**.

Question 3

Ajoutez à vos fichiers PROJET.MAD et PROJET.PAR les instructions suivantes :

LOADASI	Copie dans le registre A, le contenu du registre SI
LOADBADRSI	Charge dans le registre B, la donnée de la RAM des données dont l'adresse se trouve dans le registre SI
LOADADRSIB	Copie le contenu du registre B, dans la RAM des données à l'adresse indiquée par le registre SI
CMPDIA	Compare DI et A et met à jour les indicateurs en conséquence (l'UAL fait en interne la soustraction DI-A pour mettre à jour les indicateurs)

CMPB #valeur	Compare B et la valeur immédiate qui suit et met à jour les indicateurs en conséquence (l'UAL fait en interne la soustraction B-valeur pour mettre à jour les indicateurs)
SUBB #valeur	Effectue la soustraction B – valeur et stocke le résultat dans B
JMPN <label>	Effectue un saut à l'instruction étiquetée par <label> si le résultat de l'instruction précédente est négatif

Question 4

En utilisant **uniquement les instructions ci-dessous :**

LOADSI #valeur	LOADBADRSI	CMPB #valeur	JMP <label>
LOADASI	LOADADRSIB	SUBB #valeur	JMPN <label>
LOADDIADRSI	CMPDIA	INCSI	JMPPZ <label>

écrire un programme en assembleur, **maj.asm,** qui modifie les lettres minuscules de la chaîne présente dans la mémoire des données en lettres majuscules. La première case de la RAM des données contiendra la longueur de la chaîne. Les cases suivantes contiendront les codes ASCII des caractères de la chaîne (voir table des caractères à la fin du document).

Votre programme devra obligatoirement se terminer par l'instruction suivante : **fin** **JMP fin**

Votre programme ne devra pas dépasser plus de **16 instructions**. Toute instruction supplémentaire entraînera un **malus de -0,5 pts**. Si votre programme fonctionne **parfaitement** (dans tous les cas) avec moins de 16 instructions vous recevrez un **bonus de +0,5 pts** par instruction économisée.

Vérifiez la bonne définition de vos instructions et le bon fonctionnement de votre programme dans LOGISIM. On veillera à avoir un **jeu d'essai le plus complet possible** (chaîne mixant des lettres minuscules, majuscules, chiffres, caractères spéciaux, …).

Travail à rendre

La date limite pour la remise de votre projet est le : **Lundi 25 Mars 2024 à 23h59**.

Vous devez envoyer par mail à votre enseignant chargé de TP un fichier .zip comprenant **uniquement** les fichiers suivants (**ne pas mettre de fichier OBJ ou BIN ou EXE** sinon le mail risque de ne pas passer) :

1. Les fichiers **PROJET.MAD** et **PROJET.PAR** contenant toutes les instructions que vous avez créées (avec les définitions des champs).
2. Les fichiers **palin.asm** et **maj.asm** contenant les programmes réalisés.
3. Un fichier PDF contenant :
    a. La description détaillée des champs que vous avez créés.
    b. La description détaillée des instructions que vous avez créées (avec les valeurs des champs).
    c. Une explication sur les programmes réalisés.

Il sera tenu compte dans la notation de la qualité du dossier (rédaction, présentation, etc.).

Pénalités en cas de retard

Tout retard dans la remise du projet sera sanctionné par le **retrait d'un point dans la note finale par jour de retard**. **Attention** : il vous sera comptabilisé 1 jour de retard même si vous avez rendu avec seulement 1 heure de retard. Par exemple, si vous rendez votre projet le Mardi 26 Mars à 1h00 du matin, il vous sera comptabilisé 1 jour de retard.

Table des codes ASCII

ASCII Déc.	ASCII Hex	Char	ASCII Déc.	ASCII Hex	Char	ASCII Déc.	ASCII Hex	Char	ASCII Déc.	ASCII Hex	Char
0	00	■	32	20		64	40	@	96	60	`
1	01	■	33	21	!	65	41	A	97	61	a
2	02	■	34	22	"	66	42	B	98	62	b
3	03	■	35	23	#	67	43	C	99	63	c
4	04	■	36	24	$	68	44	D	100	64	d
5	05	■	37	25	%	69	45	E	101	65	e
6	06	■	38	26	&	70	46	F	102	66	f
7	07	■	39	27	‘	71	47	G	103	67	g
8	08	■	40	28	(	72	48	H	104	68	h
9	09	■	41	29	)	73	49	I	105	69	i
10	0A	■	42	2A	*	74	4A	J	106	6A	j
11	0B	■	43	2B	+	75	4B	K	107	6B	k
12	0C	■	44	2C	,	76	4C	L	108	6C	l
13	0D	■	45	2D	-	77	4D	M	109	6D	m
14	0E	■	46	2E	.	78	4E	N	110	6E	n
15	0F	■	47	2F	/	79	4F	O	111	6F	o
16	10	■	48	30	0	80	50	P	112	70	p
17	11	■	49	31	1	81	51	Q	113	71	q
18	12	■	50	32	2	82	52	R	114	72	r
19	13	■	51	33	3	83	53	S	115	73	s
20	14	■	52	34	4	84	54	T	116	74	t
21	15	■	53	35	5	85	55	U	117	75	u
22	16	■	54	36	6	86	56	V	118	76	v
23	17	■	55	37	7	87	57	W	119	77	w
24	18	■	56	38	8	88	58	X	120	78	x
25	19	■	57	39	9	89	59	Y	121	79	y
26	1A	■	58	3A	:	90	5A	Z	122	7A	z
27	1B	■	59	3B	;	91	5B	[	123	7B	{
28	1C	■	60	3C	<	92	5C	\	124	7C	|
29	1D	■	61	3D	=	93	5D	]	125	7D	}
30	1E	■	62	3E	>	94	5E	^	126	7E	~
31	1F	■	63	3F	?	95	5F	_	127	7F	■