Circuits & Architecture

Diagrammes temporels d’exécution

«LC-3»

JACQUETTE Pierrick & STASYSZYN Romain
21305551 & 21305734

Table des matières

Diagrammes temporels d’exécution....3
Création des tests des instructions....3
Modifications et composants ajoutés....3
1.ALU....3
2.DecodeIR....3
3.GetAddr....3
4.JSR/JSRR....4
5.Load/Store....4
6.LDI/STI....4
7.NZP....4
8.Offset11....4
9.RamCtrl....4
10.RegPC....4
Programme désassemblé....5

Ce projet est réalisé au sein de l'enseignement Circuits et Architecture. Le but de notre projet est de construire avec logisim un processeur qui implémente certaines instructions du LC3. Tous les câblages effectués ont été réalisés grâce aux codages des instructions et à leurs fonctions.

Les instructions que notre processeur traitent : NOT, ADD, AND, SETB, RSTB, LEA, LD, ST, LDR, STR, BR, JMP, JSR, JSRR, LDI et STI.

Diagrammes temporels d’exécution

Nous avons commencer le projet par les diagrammes temporels d’exécution de chaque instruction du processeur, ce qui nous a permis d’avoir une représentation claire et structurée de notre projet. Mais aussi de la fonction de chaque sous-circuit et de quand il devait fournir une sortie, à quel temps et en fonction de quelle(s) entrée(s).

Les diagrammes sont dans l'autre pdf.

Création des tests des instructions

Nous avons ensuite crée des tests pour tester notre processeur au fur et à mesure de sa création. Pour cela des fichiers assemblés ont été réalisé puis transformer en langage machine pour qu'il soit exécuté par notre processeur. En même temps on crée les fichiers .ref pour le script de test automatique.

Modifications et composants ajoutés

1. ALU

Modification du à l'ajout des instructions SETB et RSTB qui permettent de mettre un bit n à 1 ou 0 respectivement. Le principe étant de décaler les n bits vers la droite, de modifier le bit 0 (à 0 ou 1) puis de re-décaler dans l'autre sens. Pour la suite il suffit de rajouter les n bits compris entre 0 et n qui ont était mis à 0 lors des décalages.

2. DecodeIR

Modification permettant de connaître le type de l'instruction courante. Cette information est connue par le 12 et 13ième bit, cela permet aussi de définir si on va écrire ou pas dans les registres. Pour la suite nous avons modifié ce circuit pour intégrer les extensions. En regardant en plus le 14 et 15ième bit.

3. GetAddr

Modification pour effectuer le calcul d'adresse. Il ajoute le pc quand cela est

nécessaire, idem pour l'offset ou BaseR. Tout dépend de l'instruction courante pour déterminer, quel addition l'on doit effectuer.

4. JSR/JSRR

Ajout de ce circuit permettant de donner l'info que le registre où l'on doit stocker est R7.

5. Load/Store

Ajout de ce circuit permettant de faire la distinction entre les instructions LEA, LD, LDR et LDI, idem pour les instructions de Store. Physiquement on fait la distinction entre le choix de la valeur ou de l'adresse du label.

6. LDI/STI

Ajout de ce circuit permettant de savoir si l'on doit faire un deuxième appel mémoire.

7. NZP

Modification permettant de mettre à jour les flags nzp mais aussi d'en connaître la valeur quand on exécute une instruction BR.

8. Offset11

Ajout de ce circuit permettant d'avoir un offset de 11 bits pour JSR. Il contient juste un bit extender signé.

9. RamCtrl

Modification permettant le deuxième appel mémoire nécessaire pour les instructions LDI/STI en fonction du temps d’exécution et du type d'instruction.

10. RegPC

Modification permettant de calculer le pc. Il fait toujours pc+1 sauf pour les instructions BR, JMP, JSR et JSRR. Pour les instructions de branchement, le pc pour l'instruction suivante peut-être différent de +1.

Programme désassemblé

```
int size(char *s){
	int n=0;
	while(*s!='\0'){
		s++;
		n++;
	}
	return n;
}

int main(){
	char* chaine = "Hello World";
	int res = size(chaine); //R0
	int var1 = 11;  //R1
	if(var1 == res){
		int var2 = *var1; //R2
		int *var3 = var1;//R3
		char c = chaine[var1]; //R4
	}
	else{
		res=0;
	}
	res = var1 | (1 << 4);
	res = var1 & (1 << 3);
  return 0;
}
```