Micro architecture de l'ARM v2A

Laniel Francis
francis.laniel@etu.upmc.fr

10 janvier 2016

Résumé

Rapport présentant la micro architure de l'ARM v2A

Table des matières

1 Introduction

1.1 L'UE VLSI

Dans le cadre du cours d'*initiation à la conception* ***Very Large Scale Integration (VLSI)*** il m'a été demandé de réaliser une architecture simplifiée d'un processeur basée sur celle de l'ARM v2A.

Pour la modélisation j'ai utilisé le langage **Very High Speed Integrated Circuit Hardware Description Language (VHDL)** ainsi que différents outils dont voici la liste :

ghdl : un compilateur vhdl libre basé sur gnat

gtkwave : un outil libre de visualisation de simulation

Alliance CAD tools : une suite d'outil libre pour la conception assistée par ordinateur de design VLSI

1.2 L'ARM v2A

Le processeur étudié est un processeur **Reduced Instruction Set Computer** (RISC) 32 bits comportant un pipeline découpé en 5 étages (IFETCH, DECOD, EXE, MEM, WBK). C'est un processeur ARM par conséquent son jeu d'instructions s'appuie sur une gestion élégante des conditions qui sont symbolisées par 4 registres d'un bit appelés "flags" :

N : ce flag est positionné si une instruction a produit un résultat négatif

Z : ce flag sera positionné par une instruction ayant produit un résultat nul

C : ce flag sera levé lorsqu'une opération non signée produit un dépassement de capacité

V : le flag V agit identiquement au flag C mais dans le cas d'opérations signées

Grâce à ces flags il est possible de conditionner chaque opération, voici un petit aperçu de la puissance de ce langage d'assemblage face à celui de l'architecture MIPS :

```
#code C
int i;
for(i = 0; i < size; i++){
  if(tab[i] < val)
    tab[i] += val;
}
#R4 est l'adresse de notre itérateur
#R6 est l'adresse de fin du tableau
#R7 est la valeur à comparer et à potentiellement ajouter
```

```
#MIPS
  _loop :
LW R5, 0(R6)
SLT R10, R5, R7
BEQ R10, R0, _endif
NOP
ADD R5, R5, R7
_endif :
ADDIU R4, R4, 4
BNE R4, R6, _loop
NOP
```

```
#ARM
_loop :
LDR R5, 0(R6)
CMP R7, R5
ADDLT R5, R5, R7
ADD R4, R4, #4
BNE R4, R6, _loop
```

Pour cet **exemple** il est clair que même en optimisant le code de l'assembleur MIPS le code ARM sera meilleur en terme de cycles par instruction. Bien entendu il est impossible d'affirmer qu'en **général** un langage d'assemblage est meilleur qu'un autre surtout que le nombre de cycles n'est pas la seule variable à prendre en compte.

La modélisation de ce processeur aurait du m'amener à obtenir le dessin des masques en utilisant les outils Alliance sur le code VHDL écrit. Malheureusement la simulation ne s'est pas déroulée comme prévu et je n'ai pas pu obtenir ces dessins...

Dans ce rapport je présenterai d'abord les différents étages de notre processeur puis je concluerai sur mon travail.

2 Les étages du processeur

2.1 IFECTH

Cet étage a pour principale tâche d'aller lire dans le cache d'instructions la prochaine instruction à éxécuter. Une fois ceci fait il enverra à l'étage DECOD l'instruction lue.

C'est aussi cet étage qui s'occupe de la gestion du registre **PC** (Program Counter) et qui répercute les cycles de gel sur la suite du pipeline notamment grâce à cette ligne :

```
if_ir_valid <= not ic_stall;
```

2.2 DECOD

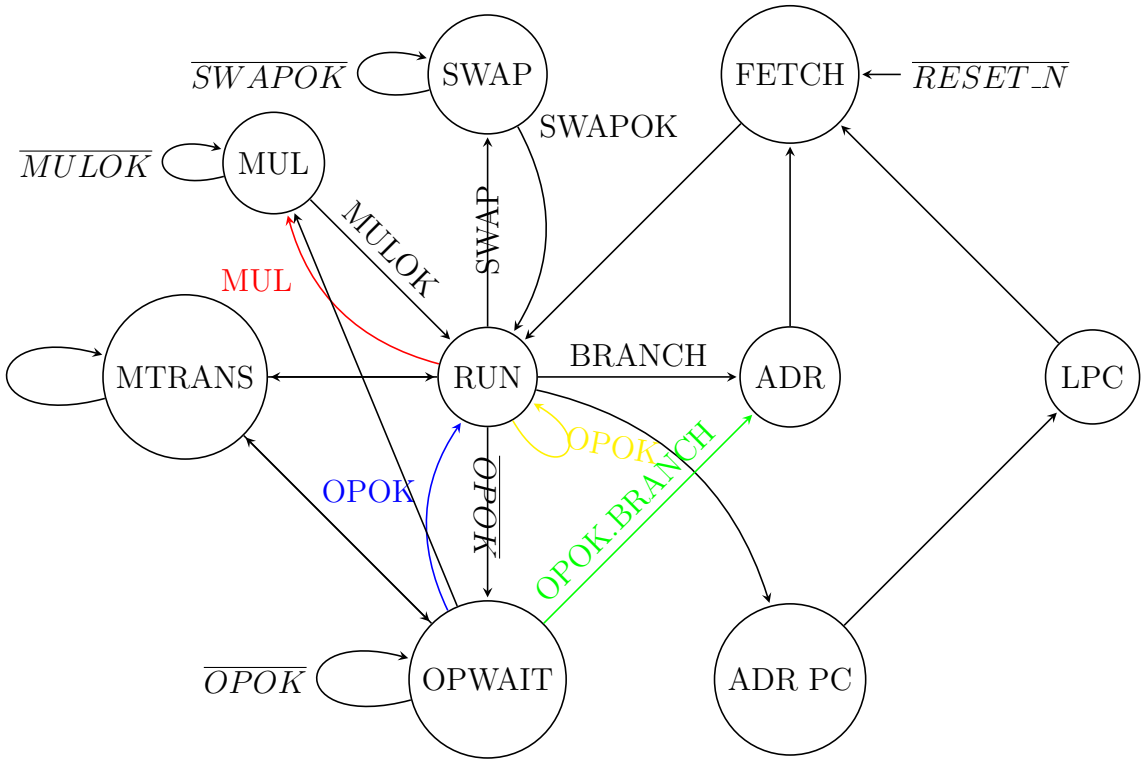

FIGURE 1 – Machine à état de DECOD (les couleurs servent uniquement à clarifier le dessin)

2.3 EXE

2.4 MEM

3 Conclusion