UE VLSI
cours 5: Architecture générale Proc ARM, détail étage EXEC

Jean-Lou Desbarbieux
UPMC 2017

Sommaire

Objectifs du pipeline

ARM n'a pas détaillé son architecture interne, nous allons essayer de l'imaginer en intégrant quelques contraintes :

- Exécuter les instructions conformément à leur spécification ;
- Assurer autant que possible un flot d'une instruction par cycle d'horloge ;
- Limiter le matériel nécessaire ;
- Supporter des interfaces mémoire (ICache et Dcache) qui ne répondent pas nécessairement dans le cycle ;
- Minimiser le temps de cycle (maximiser la fréquence de fonctionnement).

Satisfaire à la première contrainte est prioritaire sur toutes les autres.

Une architecture asynchrone

Le jeu d'instruction *ARM* comprend des instructions de complexité très variable. Certaines instructions seront sans difficulté exécutées en 1 cycle, d'autres telles que les transferts multiples peuvent prendre plusieurs cycles.
Les accès à la mémoire peuvent être longs, si il est possible poursuivre l'exécution pourquoi ne pas le faire ?
Nous avons fait le choix de concevoir un `pipeline` asynchrone, on évite le gel de tous les étages.
Implique une gestion fine des dépendances à chaque registre et *flag* est associé un bit de validité.
Une instruction n'est lancée que si toutes ses opérandes sources sont valides.
Le registre correspondant au résultat de l'instruction et marqué comme non valide du lancement de l'instruction jusqu'à la production effective du résultat.
Une instruction dont la destination n'est pas valide n'est pas lancée pour éviter que ses différentes affectations soient réalisées dans le désordre.

4 étages !

Nous avons fait le choix d'une architecture à 4 étages :

- **IFETCH**
- **DECOD**
- **EXEC**
- **MEM**

Interfaces, fonctionnalités et structure

EXEC reçoit ses instructions de l'étage DECOD.
EXEC transmet le instructions mémoire vers MEM (Adresse donnée).
EXEC envoie ses résultats à DECOD qui contient le banc de registre.
EXEC calcule le résultats des instructions arithmétiques et logiques, EXEC calcule les adresses des transferts mémoires, EXEC calcule les adresses des branchements.
EXEC est principalement constitué d'un décaleur et d'une ALU.

Instructions arithmétique set logiques

0000 - **AND** : $Rd <= Rn\ AND\ Op2$
0001 - **EOR** : $Rd <= Rn\ XOR\ Op2$
0010 - **SUB** : $Rd <= Rn - Op2$
0011 - **RSB** : $Rd <= Op2 - Rn$
0100 - **ADD** : $Rd <= Rn + Op2$
0101 - **ADC** : $Rd <= Rn + Op2 + C$
0110 - **SBC** : $Rd <= Rn - Op2 + C - 1$
0111 - **RSC** : $Rd <= Op2 - Rn + C - 1$
1000 - **TST** : Positionne les *flags* pour $Rn\ AND\ Op2$
1001 - **TEQ** : Positionne les *flags* pour $Rn\ XOR\ Op2$
1010 - **CMP** : Positionne les *flags* pour $Rn - Op2$
1011 - **CMN** : Positionne les *flags* pour $Rn + Op2$
1100 - **ORR** : $Rd <= Rn\ OR\ Op2$
1101 - **MOV** : $Rd <= Op2$
1110 - **BIC** : $Rd <= Rn\ AND\ NOT\ Op2$
1111 - **MVN** : $Rd <= NOT\ Op2$

Interface détaillée 1

```
-- Decode interface synchro
dec2exe_empty : in Std_logic;
exe_pop : out Std_logic;

-- Decode interface operands
dec_op1 : in Std_Logic_Vector(31 downto 0);
dec_op2 : in Std_Logic_Vector(31 downto 0);
dec_exe_dest : in Std_Logic_Vector(3 downto 0);
dec_exe_wb : in Std_Logic;
dec_flag_wb : in Std_Logic;
```

Interface détaillée 2

```
-- Shifter command
dec_shift_lsl : in Std_Logic;
dec_shift_lsr : in Std_Logic;
dec_shift_asr : in Std_Logic;
dec_shift_ror : in Std_Logic;
dec_shift_rrx : in Std_Logic;
dec_shift_val : in Std_Logic_Vector(4 downto 0);
dec_cy : in Std_Logic;
```

Interface détaillée 3

```
-- Alu operand selection
dec_comp_op1 : in Std_Logic;
dec_comp_op2 : in Std_Logic;
dec_alu_cy   : in Std_Logic;

-- Alu command
dec_alu_cmd : in Std_Logic_Vector(1 downto 0);
```

Interface détaillée 4

```
-- Exe bypass to decod
exe_res : out Std_Logic_Vector(31 downto 0);

exe_c : out Std_Logic;
exe_v : out Std_Logic;
exe_n : out Std_Logic;
exe_z : out Std_Logic;

exe_dest : out Std_Logic_Vector(3 downto 0); -- Rd des
exe_wb : out Std_Logic; -- Rd destination write back
exe_flag_wb : out Std_Logic; -- CSPR modifiy
```

Interface détaillée 5

```
-- Decode to mem interface
dec_mem_data : in Std_Logic_Vector(31 downto 0); -- da
dec_mem_dest : in Std_Logic_Vector(3 downto 0); -- De
dec_pre_index : in Std_logic;

dec_mem_lw : in Std_Logic;
dec_mem_lb : in Std_Logic;
dec_mem_sw : in Std_Logic;
dec_mem_sb : in Std_Logic;
```

Interface détaillée 5

```
-- Mem interface
exe_mem_adr : out Std_Logic_Vector(31 downto 0); -- Al
exe_mem_data : out Std_Logic_Vector(31 downto 0);
exe_mem_dest : out Std_Logic_Vector(3 downto 0);

exe_mem_lw : out Std_Logic;
exe_mem_lb : out Std_Logic;
exe_mem_sw : out Std_Logic;
exe_mem_sb : out Std_Logic;

exe2mem_empty : out Std_logic;
mem_pop : in Std_logic;
```