Decode

ARO
3 - Decode

Résumé du document

Le décodage dans un processeur traduit les instructions binaires en signaux de contrôle pour les opérations nécessaires. Une banque de registres stocke des données temporaires, entourée d'un décodeur en entrée et d'un multiplexeur en sortie. Dans un processeur ARM, différents registres ont des objectifs spécifiques comme le Stack Pointer (SP) pour la gestion de la pile. Les instructions ARM telles que MOV et ADD permettent de manipuler les données et d'effectuer des opérations arithmétiques.

Table des matières

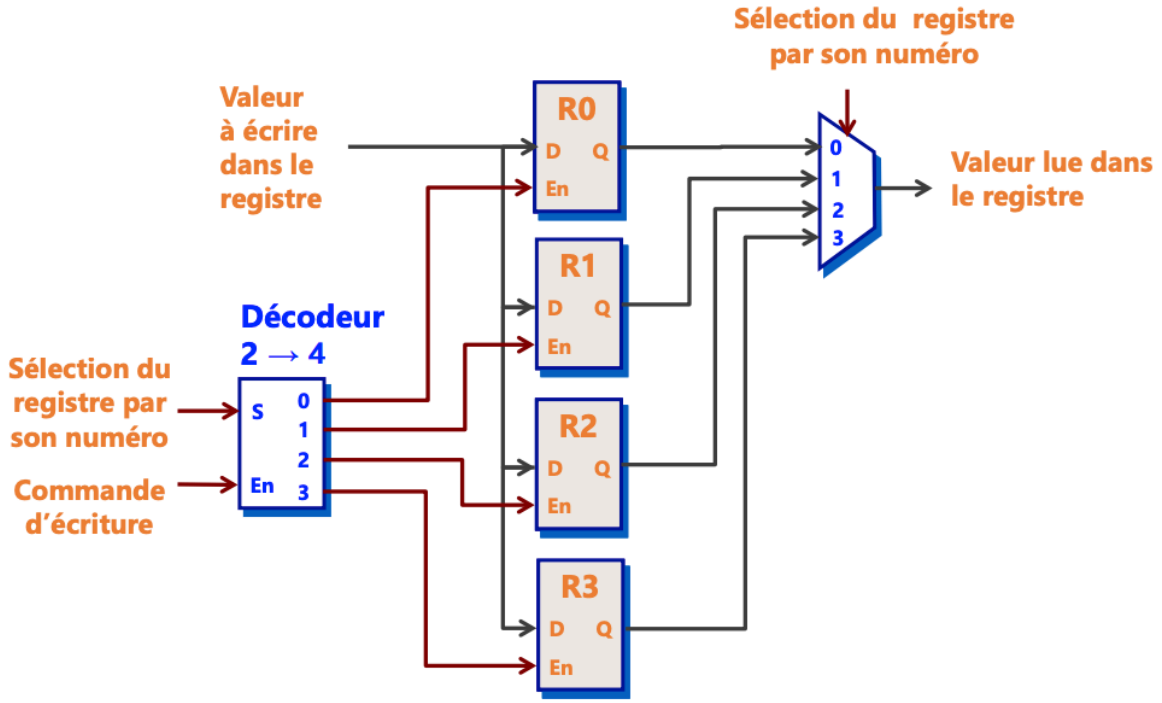
1. Opération DECODE

L’opération de décodage dans un processeur consiste à interpréter les instructions binaires reçues depuis la mémoire et à les traduire en signaux de contrôle. Ces signaux dirigent les différentes opérations nécessaires pour exécuter l’instruction, comme les accès aux registres, les opérations arithmétiques ou logiques, et la gestion des flux de données.

2. Banque de registres

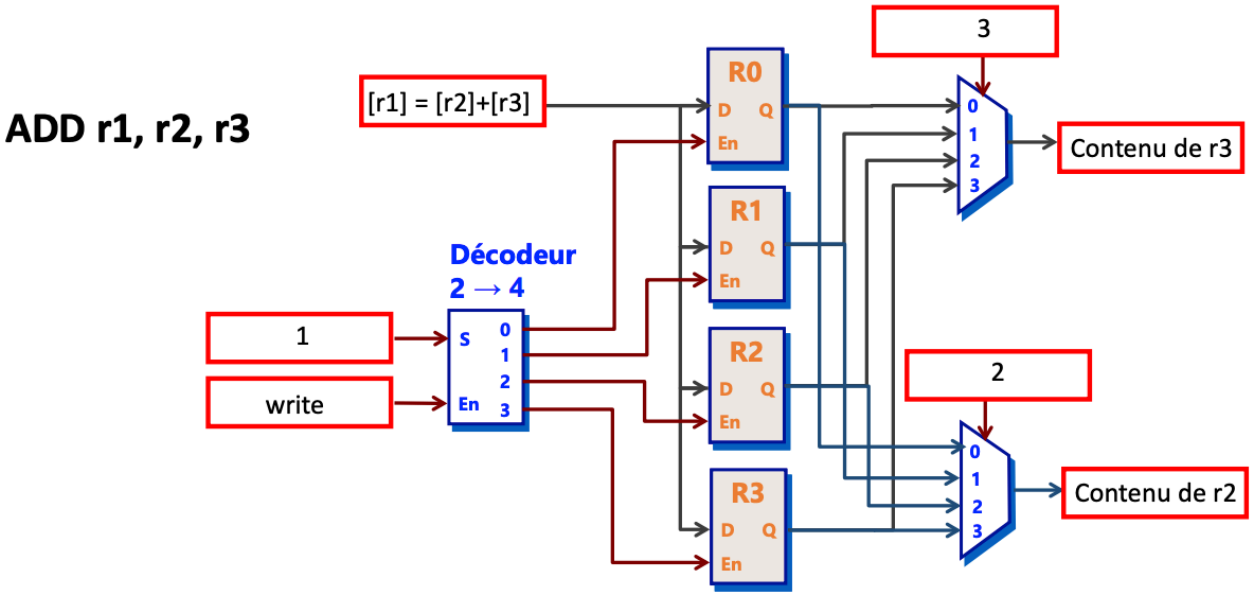
Une banque de registre s'articule autour de plusieurs registres, chacun stockant une valeur. Ces registres sont utilisés pour stocker des données temporaires, des adresses mémoire, des résultats intermédiaires, ou des valeurs de contrôle.

Ces registres sont entourés d'un décodeur en entrée et d'un multiplexeur en sortie. Le décodeur permet de définir dans quel registre nous allons écrire tandis que le multiplexeur permet de choisir quel registre nous allons lire.



2.1. Traitement de plusieurs valeur en parallèle

Dans un processeur, les registres sont souvent utilisés pour traiter plusieurs valeurs en parallèle. Pour cela il est nécessaire de pouvoir lire au même coup d'horloge des valeurs de plusieurs registres. Pour cela on utilise un multiplexeur supplémentaire qui permet de lire dans un registre de plus.



3. Fonctionnement ARM

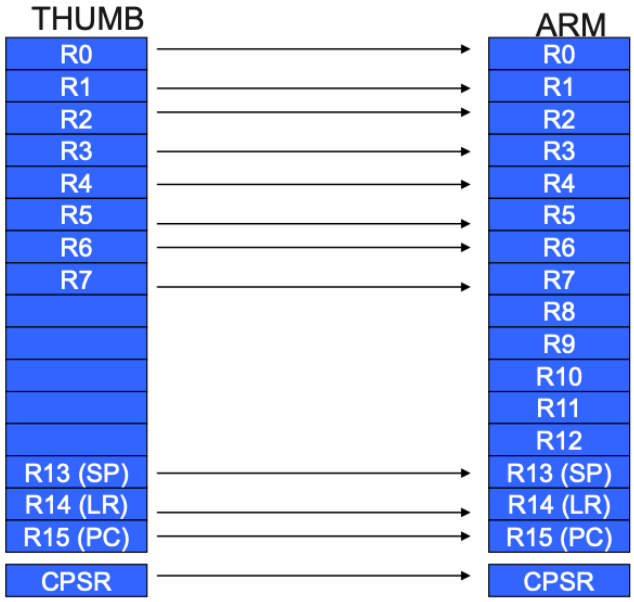
Dans un processeur ARM nous avons un total de 37 registres physiques. Il y en a 16 (+2) qui sont utilisé en fonction du mode.

Registre	Objectif
R13	Stack Pointer (SP) → Stocke la position dans la pile de stockage (interruptions chaînées)
R14	Link Register (LR) →Garde l'adresse de retour (appel de fct, interruption)
R15	Program Counter (PC) →Stocke l'adresse de la prochaine instruction

3.1. Différence ARM / THUMB

- ARM : C'est l'ensemble d'instructions original, utilisant des instructions de 32 bits pour des performances maximales. Il est adapté aux tâches nécessitant une puissance de calcul élevée, comme les smartphones haut de gamme ou les serveurs.
- Thumb : C'est une version compacte de l'ensemble d'instructions ARM, avec des instructions de 16 bits. Il est utilisé pour économiser de l'espace mémoire et de l'énergie, adapté aux appareils embarqués ou mobiles à basse puissance.

En résumé, ARM est performant mais utilise plus d'espace mémoire, tandis que Thumb est plus compact et économe en énergie, adapté aux appareils avec des contraintes de mémoire ou de puissance.

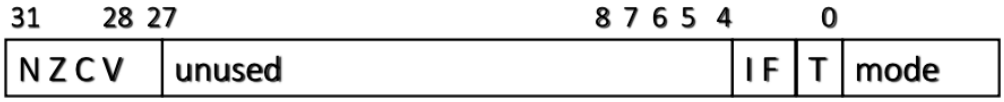


4. CSPR (Current Program Status Register)

Le registre CSPR est un registre de 32 bits qui contient des informations sur l'état courant du processeur. Il contient des informations sur les drapeaux de statut, les modes d'exécution, et les niveaux de priorité des interruptions.

Il est composé des bits suivants:

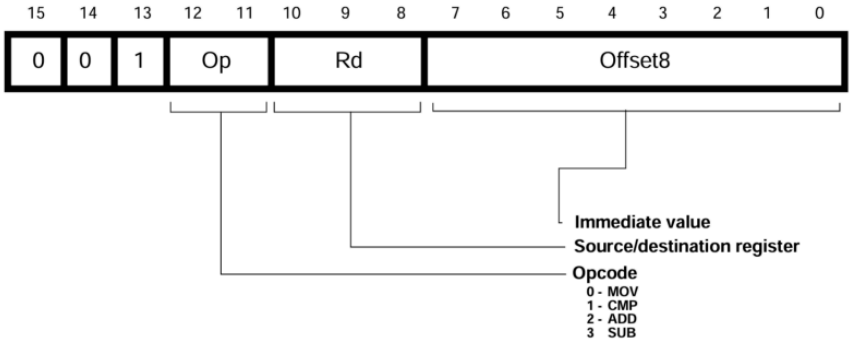
- Bits de condition (N Z C V)
- Bits de masquage des interruptions (I F)
- Jeu d'instructions (T)
- Mode du processeur (mode)



5. Instruction ARM

5.1. Move

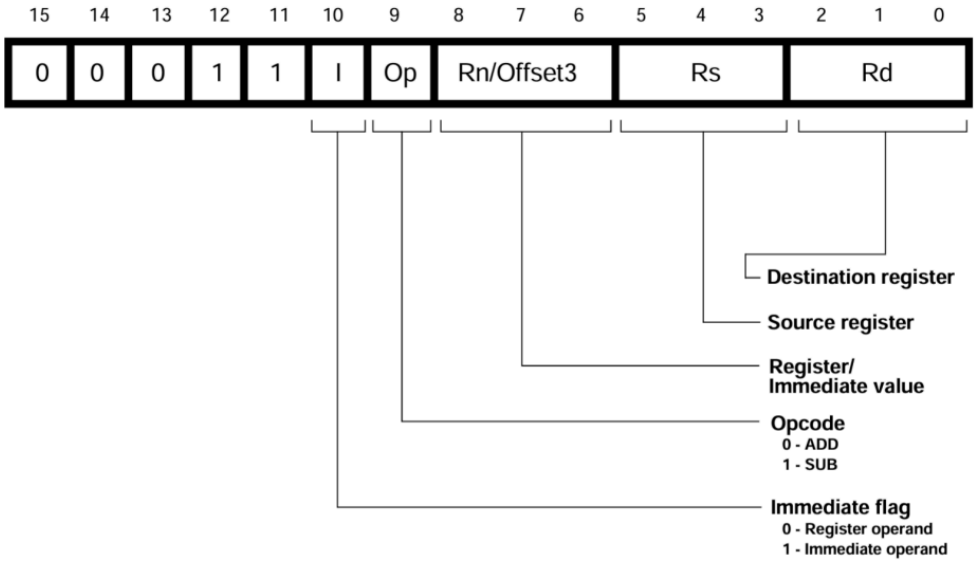
L'instruction `MOV` permet de déplacer une valeur d'un registre à un autre. Elle prend en entrée un registre source et un registre destination, et copie la valeur du registre source dans le registre destination.



- syntaxe : `MOV <Rd>, <Rs>`
- charge la valeur de Rs dans Rd
- flag = 1, Opcode = 0
- offset3 = 000

5.2. Add

L'instruction `ADD` permet d'additionner deux valeurs et de stocker le résultat dans un registre. Elle prend en entrée deux registres source et un registre destination, et ajoute les valeurs des deux registres source pour stocker le résultat dans le registre destination.



- syntaxe : `ADD <Rd>, <Rs>, #<immed_3>`
- ajoute la valeur immédiate (3 bits non-signé) à la valeur du registre Rs et écrit le résultat dans Rd

OU

- syntaxe : `ADD <Rd>, <Rs>, <Rn>`
- ajoute la valeur de Rs à celle de Rn et écrit le résultat dans Rd