

# CONCEPTION LOGIQUE D'UN PROCESSEUR ARM V2.3

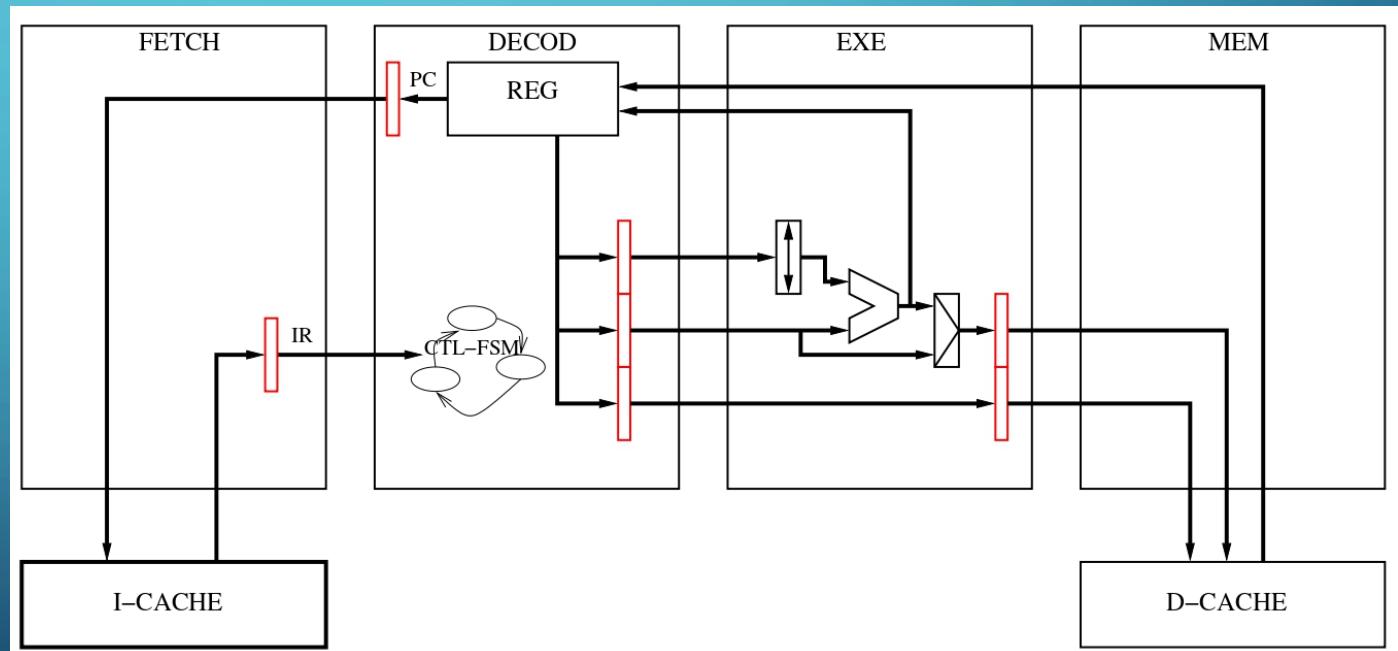
THAÏS MILLERET - GUILLAUME REGNAULT

SORBONNE UNIVERSITÉ MASTER INFORMATIQUE SEMESTRE 1 09/2024 - 01/2025

CONCEPTION DE CIRCUITS INTÉGRÉS NUMÉRIQUES, JEAN-LOU DESBARBIEUX

# ARCHITECTURE DU PROCESSEUR

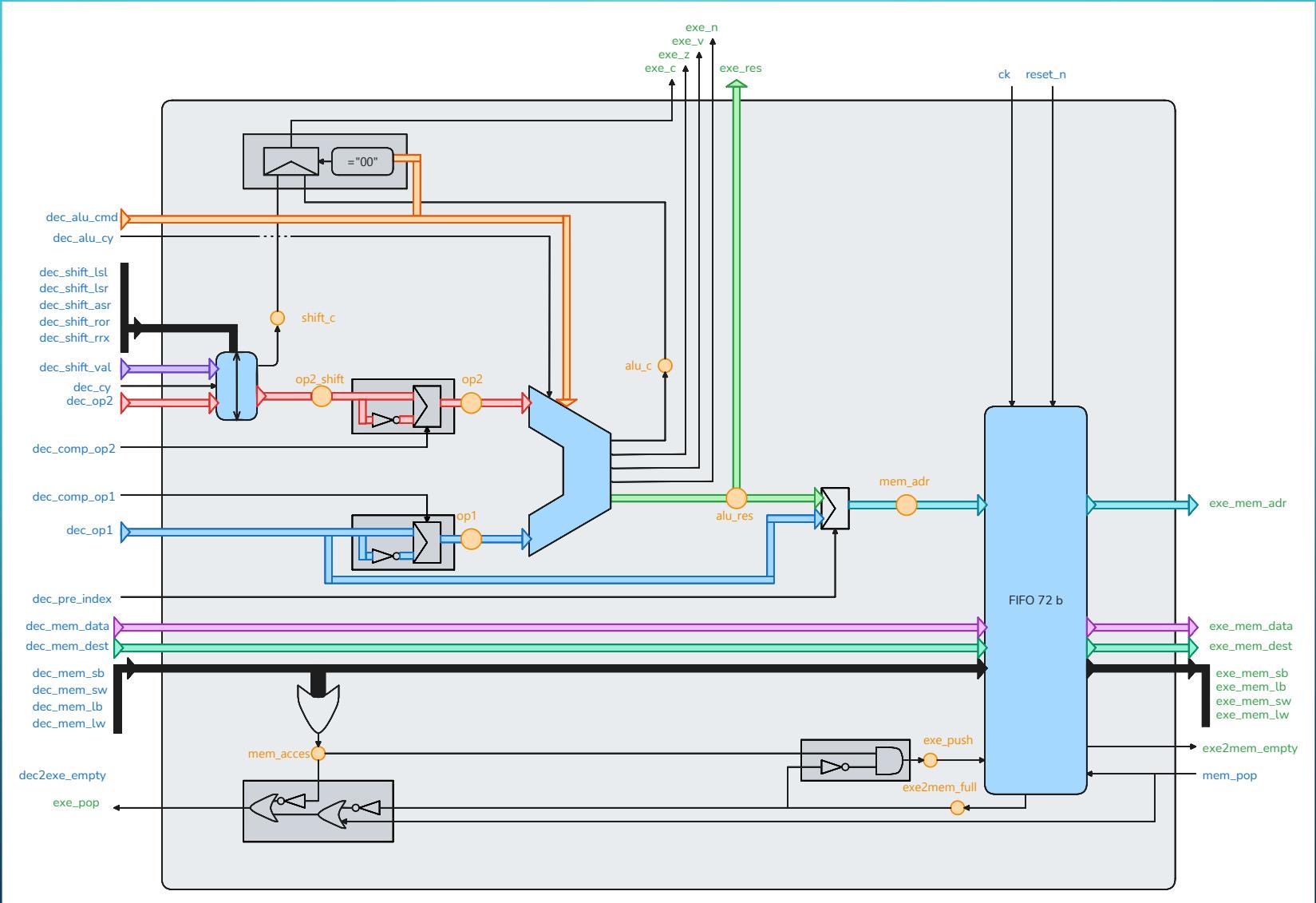
- Architecture asynchrone => étages séparés par des FIFO
- 16 registres dont SP, LR, PC et CPSR (Current Program Status Register)
- CPSR flags
  - N : ALU res < 0
  - Z : ALU res = 0
  - C : retenue
  - V : dépassement de capacité



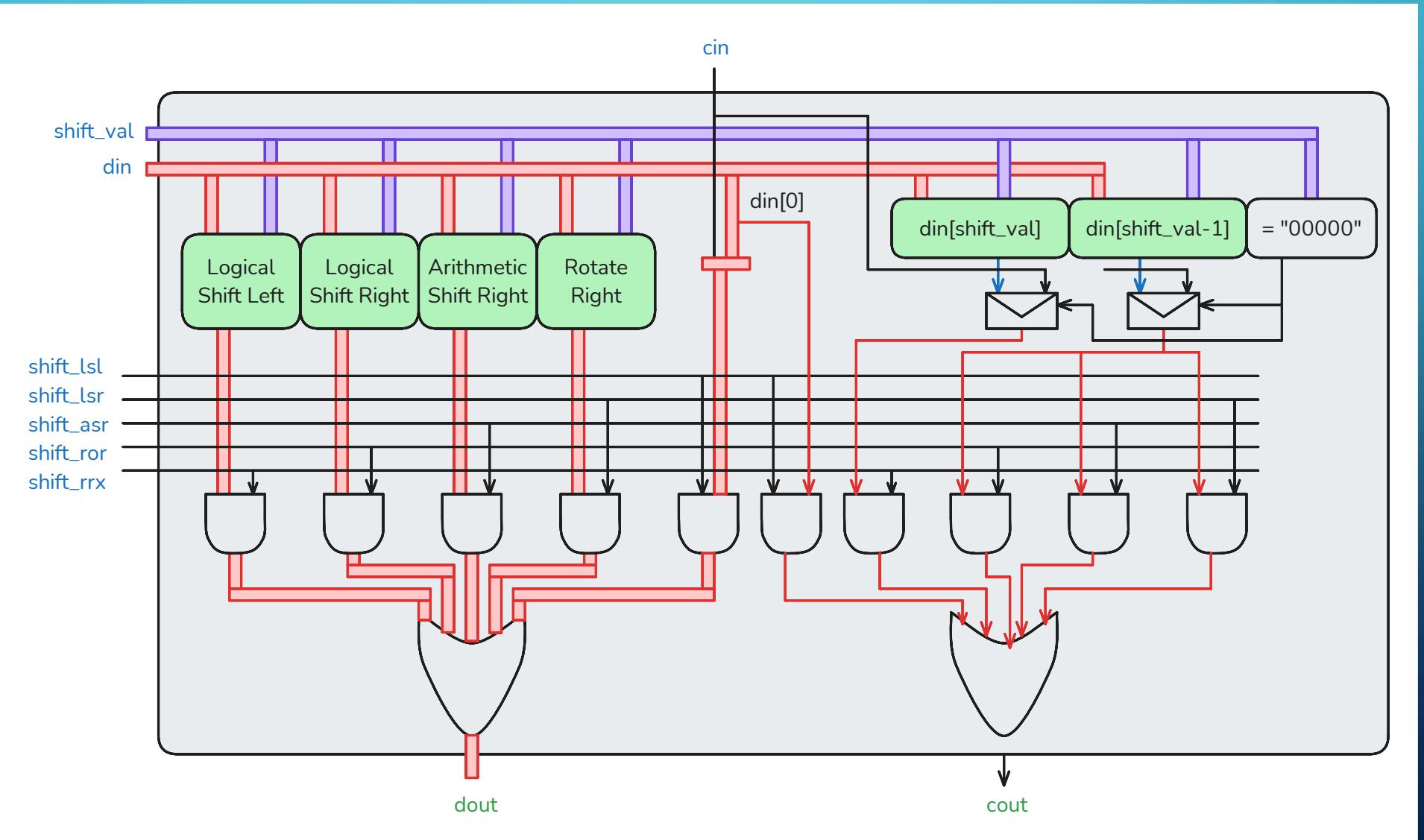
# TYPES D'INSTRUCTION À GÉRER

- Traitement de données
- Multiplications
- Branchements
- Accès mémoire simples
- Accès mémoire multiples

# ÉTAGE EXE

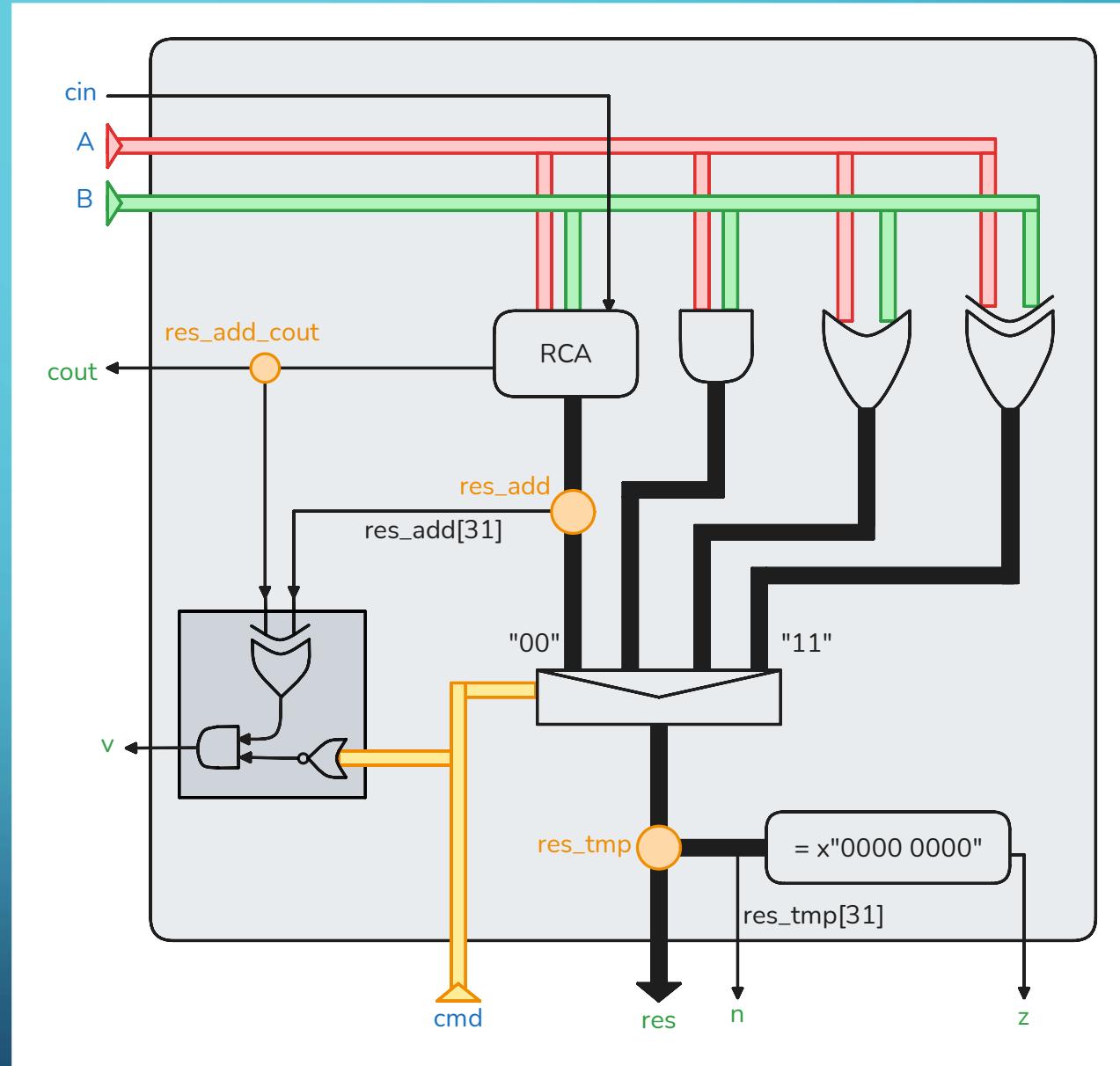


# SHIFTER



# ALU

- 4 opérations en fonction de cmd
  - 00 : ADD (avec cin)
  - 01 : AND
  - 10 : OR
  - 11 : XOR
- RCA = Ripple Carry Adder



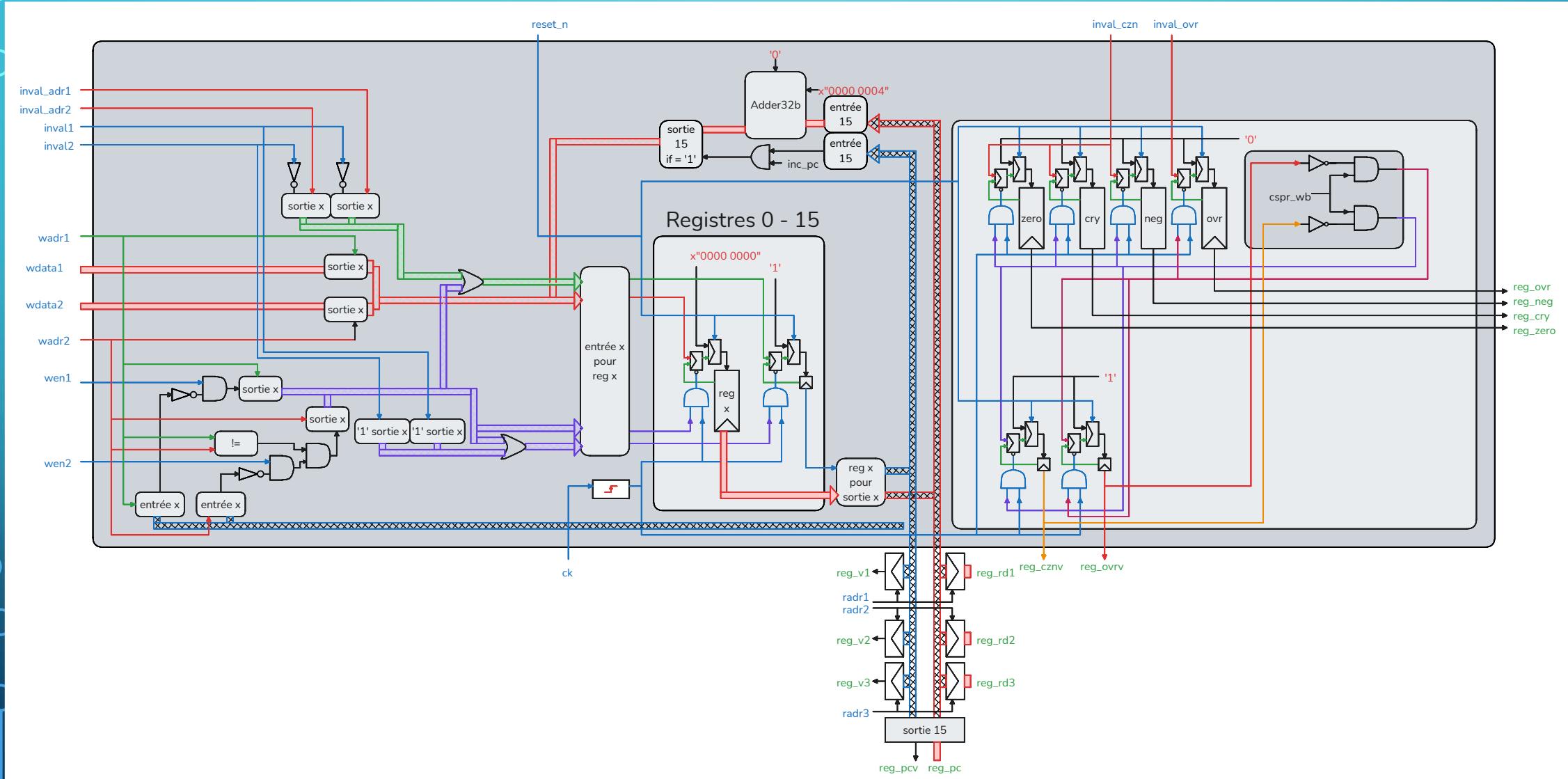
# TEST BENCH DE EXE

- Shifter : tests pour les 5 opérations (LSL, LSR, ASR, ROR et RRX) pour des valeurs aléatoires
- ALU : tests pour les 4 opérations (ADD, AND, OR et XOR) pour des valeurs aléatoires et des cas extrêmes + flags
- EXE : envoi d'instructions décodées à traiter, vérification de la sortie de EXE vers REG ou MEM et état de la FIFO EXE-MEM

# ÉTAGE DECOD

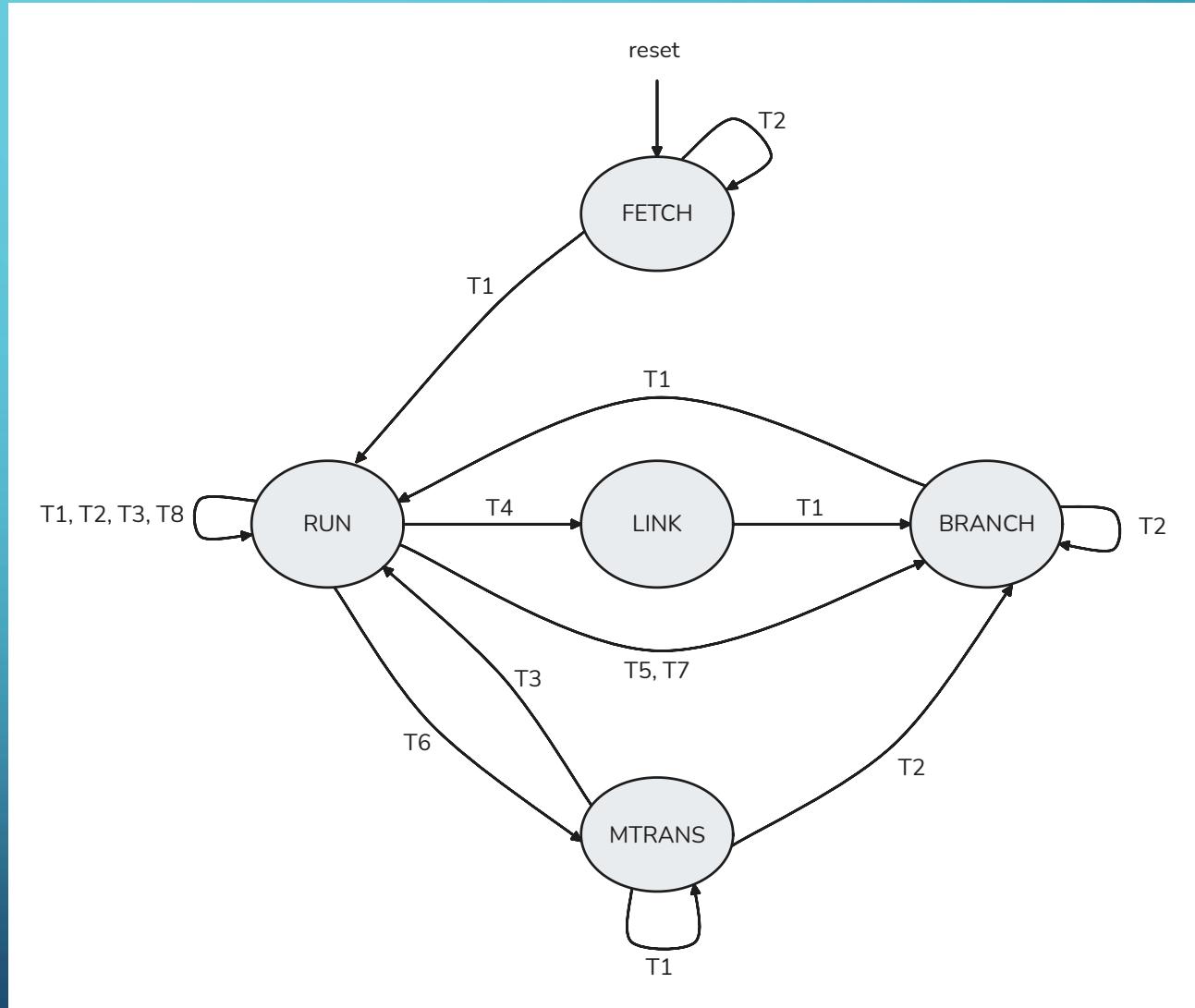
- DECOD = le banc de registre REG + une machine à états
- 2 fonctionnalités
  - décoder les instructions pour permettre leur exécution par les étages EXE et MEM
  - assurer le séquencement du pipeline, en particulier pour les instructions multi-cycles
- 1 instruction = mot de 32 bits => mot de 127 bits interprétable par EXE et MEM
- Instruction lancée si flags, registres sources et destination valides

# BANC DE REGISTRES REG



# MACHINE À ÉTATS DE DECODAGE

1. envoyer l'adresse d'une instruction dans la FIFO dec2if
2. lire l'instruction chargée par IFETCH et stockée dans la FIFO if2dec
3. décoder l'instruction chargée
4. écrire le résultat du décodage dans la FIFO dec2exec à destination de EXE



# IDÉES POUR LES TRANSFERTS MULTIPLES

- Transfert multiple => état MTRANS
- Boucler dans MTRANS tant qu'il reste des registres à traiter
- Pop le registre une fois traité

## TEST BENCH DE DECOD

- REG : vérifier que les registres sont mis à jour au bon moment et renvoient la bonne valeur lorsqu'on les lit
- DECOD testé via la plateforme globale

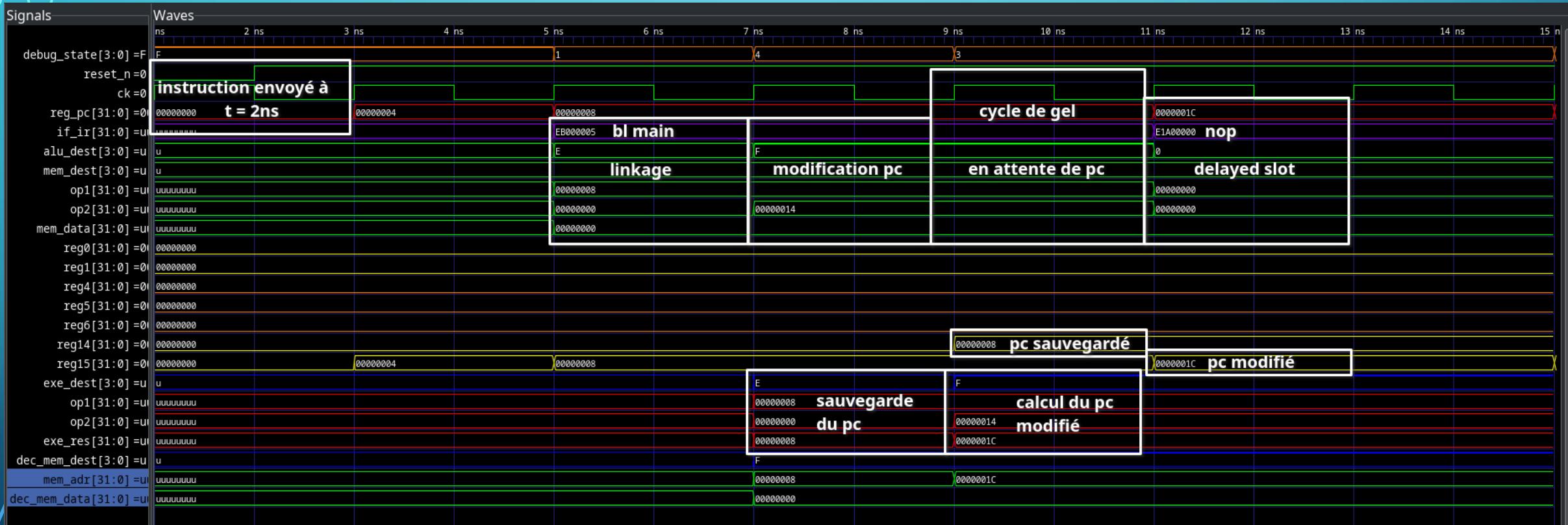
# PLATEFORME GLOBALE

- Instancie et connecte les 4 étages
- Processus qui envoie les instructions à tester à IFETCH
- Processus qui vérifie les valeurs en sortie de DECOD
- Vérification des valeurs des signaux (dont les registres de REG) via GTKWave
- Affichages dans le terminal pour suivre l'exécution (trace)

# TEST DE PROGRAMME : SOMME D'UN VECTEUR

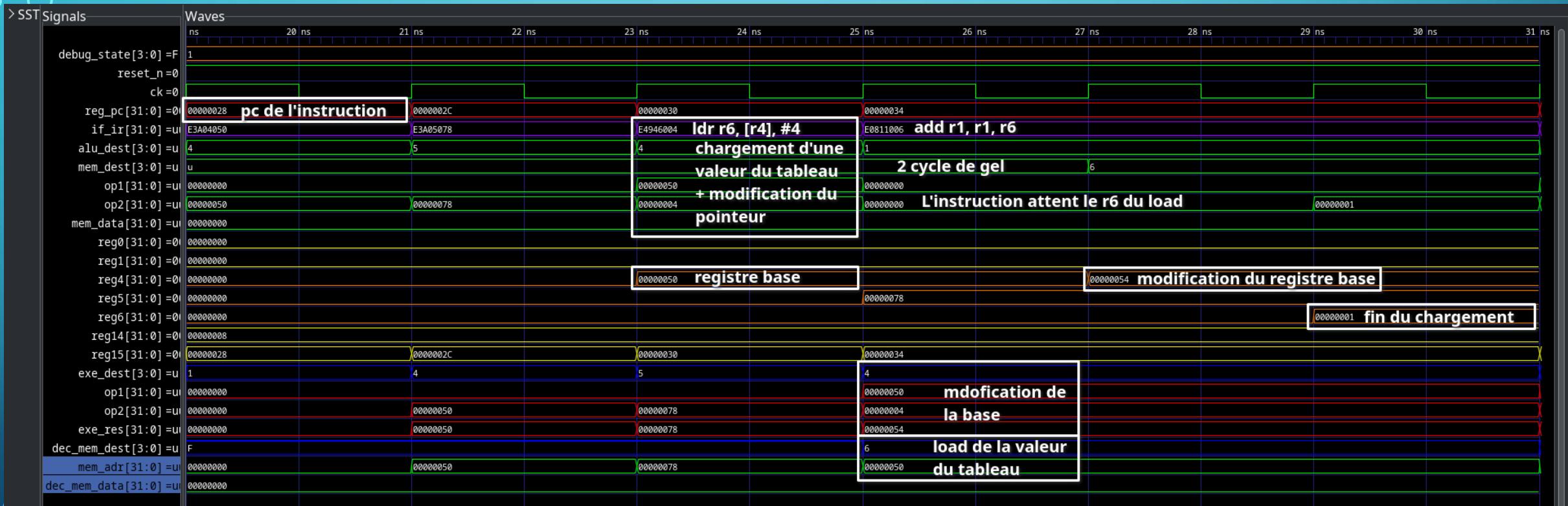
```
int tab[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
  
int main() {  
    int sum = 0;  
    int * ptr;  
    for (ptr = tab; ptr < &tab[10]; ptr++) {  
        sum = sum + (*ptr);  
    }  
    return sum;  
}
```

# BRANCH AND LINK SUR GTKWAVE



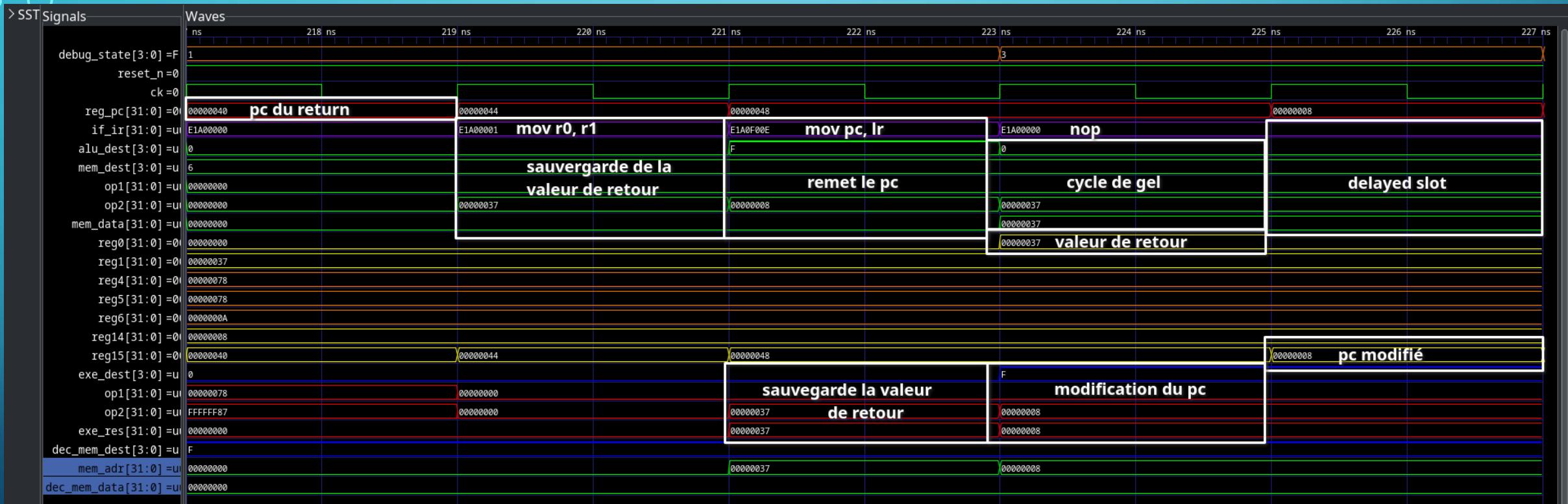
Saut de \_start vers main

# LOAD AVEC POST INDEXATION SUR GTKWAVE



Le load est dans la boucle for. Capture d'écran de l'itération 0.

# RETURN SUR GTKWAVE



Return du main

# TRACE DANS LE TERMINAL

Chargement du segment .text adr = 0x0

Chargement du segment pile adr = 0x7ffff000 Taille = 0x1000

Symbol \_good found at adr=84

Symbol \_bad found at adr=80

main\_tb.vhdl:231:17:@241ns:(report note): TTY out : 0x00000037

main\_tb.vhdl:218:9:@246ns:(assertion note): GOOD!!!

main\_tb.vhdl:220:9:@246ns:(assertion note): end of test

# CONCLUSION