PARM-esan

Quentin ELLEON
Eliot MENORET

Matice MARILL
Gauthier MARTIN

Sommaire

01 PRÉSENTATION DU PROJET

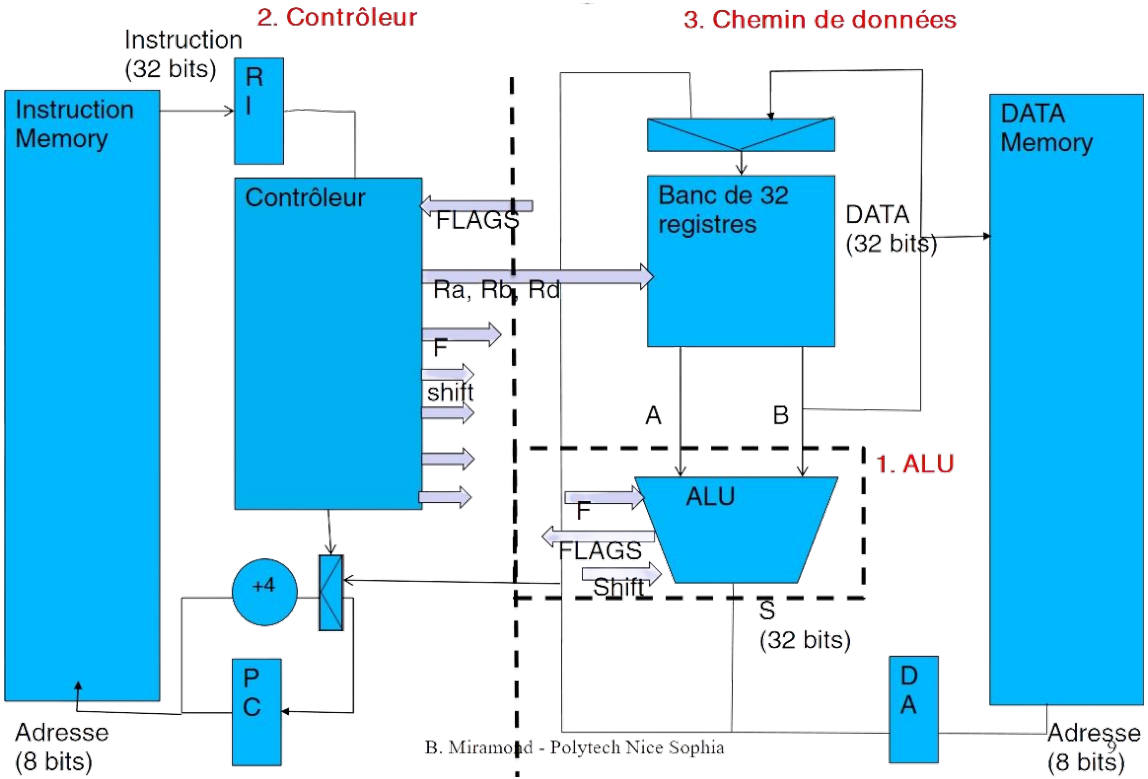
02 RÉPARTITION DES TÂCHES

03 QUALITÉ

04 DÉMONSTRATION

05 PROBLÈMES

Le PARM-esan

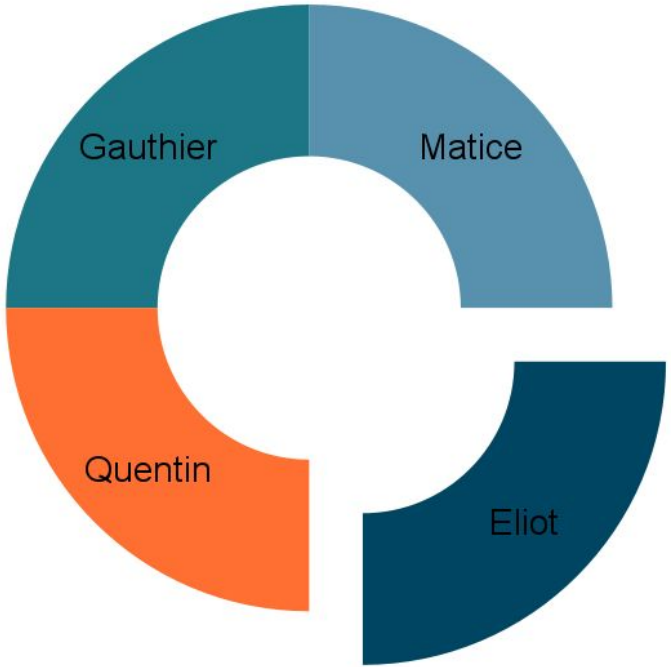


Un microprocesseur ARM

Répartition des tâches

- ALU
- Banc de registre
- Shift, add, sub, mov

Répartition des tâches



- CPU
- Data Processing
- Load Store

Répartition des tâches

- Code assembleur
- Code C
- Test des composants

Répartition des tâches

- Conditional
- Flags APSR
- Fix du sasm, ALU…

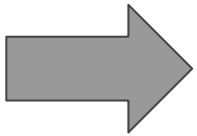
Qualité

Bonne qualité	Mauvaise qualité
Tests d'intégration ASM passent	Certains vecteurs de test ne passent pas
Possibilité de faire tourner des programmes complexes, avec gestion des entrées et sorties	Architecture du Shift Add Sub Move et de l'ALU trop complexe
Code C de l'assembleur clair, modulaire et facilement améliorable	Notre assembleur ne supporte pas les labels et l'indentation

Démo: les diviseurs

Code c

```c
//calculateur de diviseurs pour un nombre saisi sur clavier
void run()
{
    BEGIN();
    int a;
    PUTCHAR('A','=');
    a = READINT();
    PUTCHAR('\n');
    PUTCHAR('l','i','s','t','e',' ','d','e','s',' ','d','i','v','i','s','e','u','r','s',':');
    for(int k=1; k<=(a+1)/2; k++){
        if(MOD(a,k) == 0){
            RES = k;
            PRINTRES_SIGN();
            PUTCHAR(',');
        }
    }
    PUTCHAR('\n','s','t','o','p');
    WAITKEY();
    RESET();
    END();
}
```



conversion via Clang 8

code assembleur

```
.LBB0_3:
    b    .LBB0_4
.LBB0_4:
    movs    r0, #61
    str    r0, [sp, #48]
    b    .LBB0_5
.LBB0_5:
    b    .LBB0_6
.LBB0_6:
    movs    r0, #0
    str    r0, [sp, #20]
    b    .LBB0_7
.LBB0_7:
    b    .LBB0_8
.LBB0_8:
    b    .LBB0_9
.LBB0_9:
    ldr    r0, [sp, #72]
    cmp    r0, #0
    bne    .LBB0_11
    b    .LBB0_10
.LBB0_10:
    b    .LBB0_9
.LBB0_11:
    b    .LBB0_12
.LBB0_12:
    ldr    r0, [sp, #76]
    str    r0, [sp, #12]
    ldr    r0, [sp, #12]
    str    r0, [sp, #16]
    ldr    r0, [sp, #16]
    cmp    r0, #10
    bne    .LBB0_14
    b    .LBB0_13
```

Un commentaire sur le code C

Création d'un dictionnaire faisant office de base de données

Code dynamique et rigoureux

Permet le rajout de nouvelles instructions facilement.

```
void initDictionary(struct Dictionary *dict) {
    dict->size = 1;
    addToDictionary(dict, "movs", 2, "00100", 3, 8, 0, 0,1); //imm
    addToDictionary(dict, "ands", 2, "0100000000", 3, 3, 0, 1,0);
    addToDictionary(dict,"eors",2,"0100000001",3,3,0,1,0);
    addToDictionary(dict,"lsls",2,"0100000010",3,3,0,1,0); //reg
    addToDictionary(dict,"lsls",3,"00000",5,3,3,1,1); //imm
    addToDictionary(dict,"lsrs",2,"0100000011",3,3,0,1,0); //reg
    addToDictionary(dict,"lsrs",3,"00001",5,3,3,1,1); //imm
    addToDictionary(dict,"rsbs",3,"0100001001",0,3,3,1,1); //imm
    addToDictionary(dict,"asrs",2,"0100000100",3,3,0,1,0); //reg
    addToDictionary(dict,"asrs",3,"00010",5,3,3,1,1); //imm
    addToDictionary(dict,"adcs",2,"0100000101",3,3,0,1,0);
    addToDictionary(dict,"rors",2,"0100000111",3,3,0,1,0);
    addToDictionary(dict,"sbcs",2,"0100000110",3,3,0,1,0);
    addToDictionary(dict,"tst",2,"0100001000",3,3,0,1,0);
    addToDictionary(dict,"cmn",2,"0100001011",3,3,0,1,0);
```

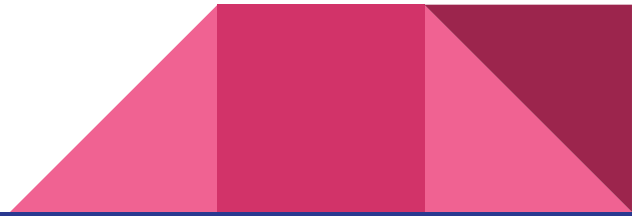
instruction, nombre d'arguments, codop, taille arg1, arg2, arg3, si il ya besoin d'inverser ou non certains champs et si c'est un immédiat ou non

Démo: les diviseurs

Sous vos grands yeux ébahis

Problèmes rencontrés

- Gestion des flags C (*carry*) et V (*overflow*)
- Difficultés sur le SASM
- Inversion des registres Rm et Rn pour certaines opérations du *Data Processing*
- Dans l’assembleur: gestion des différences entre opérations sur registres et opérations sur immédiats



Conclusion

Un projet complexe, abordant :

- Le fonctionnement interne d’un processeur
- Des notions d’électronique
- Des connaissances en assembleur

Merci pour votre attention !