RAPPORT: Virtual core implementation

Pailhe Mathieu **Groupe A**

Partie 1: Virtual core

Explication du rendu:

Rendu en solitaire car mon partenaire ne m'a jamais partagé son code!
Je suis parti du principe qu'il avait fait le code du compiler, et j'ai fait l'autre partie. Je me suis un peu aidé du code de mes camarades pour faire mes tests comme si j'avais un compiler qui marchait à côté de moi, (mais bien sûr je ne l'ai pas ajouté au rendu).
(Teo Martin)

Sans parler d'avant car il était sur un autre projet sur lequel il s'est fait virer par son groupe.

Je ne dis pas que c'est une excuse pour avoir fait si peu, mais on peut dire que je l'attendais pour faire le projet, mais qu'il n'est jamais arrivé et je n'ai jamais eu la motivation de faire sa partie.

https://github.com/wincoverr/Virtual-core-implementation

Le Code:

Tout d'abord le projet ne répond pas à l'énoncé car je n'ai pas géré ceci :

BIN_NAME <CODE> <STATE> (VERBOSE)

J'ai toutefois créé un code que je n'ai pas pu tester pour récupérer et stocker ces arguments:

```c
// Verification
if (argc < 3 || argc > 4)
{
    fprintf(stderr, "BIN_NAME <CODE> <STATE> (VERBOSE)");
    return 1;
}
bool verbose = (argc == 4 && strcmp(argv[3], "VERBOSE") == 0);

// Ouvre le fichier de code spécifié dans le premier argument, vérifie que le fichier est ouvert avec succès et calcule sa taille.
// charge ensuite le contenu du fichier dans un tableau d'octets.
FILE *codeF = fopen(argv[1], "rb");
if (!codeF)
{
    fprintf(stderr, "erreur sur le fichier code");
    return 1;
}
fseek(codeF, 0, SEEK_END);
long code_size = ftell(codeF);
rewind(codeF);
uint8_t *code = (uint8_t *)malloc(code_size);
fread(code, 1, code_size, codeF);
fclose(codeF);

// Ouvre le fichier d'état initial spécifié dans le deuxième argument, vérifie que le fichier est ouvert avec succès.
FILE *stateF = fopen(argv[2], "r");
if (!stateF)
{
    fprintf(stderr, "erreur sur le fichier state");
    return 1;
}

// Je ne sais pas trop quoi faire après avec ce fichier
```

fonction decode:

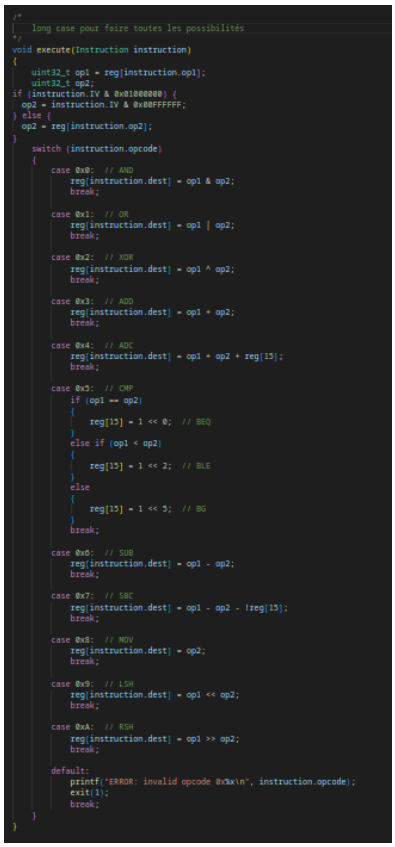
```c
/*
Décode l'instruction à partir d'une valeur de 32 bits.Les bits sont extraits en suivant l'énoncé et stockés dans une structure.
*/
Instruction decode(uint32_t *buff)
{
    Instruction instruction = {0};
    uint32_t mask = 0b00000000000000000000000011111111;

    uint32_t tmp = ntohl(*buff);

    instruction.IV = tmp & mask;
    mask = 0b00000000000000000000111100000000;
    instruction.dest = (tmp & mask) >> 8;
    mask = 0b00000000000000001111000000000000;
    instruction.op2 = (tmp & mask) >> 12;
    mask = 0b00000000000011110000000000000000;
    instruction.op1 = (tmp & mask) >> 16;
    mask = 0b00000000001111000000000000000000;
    instruction.opcode = (tmp & mask) >> 20;
    mask = 0b00000001000000000000000000000000;
    instruction.flag = (tmp & mask) >> 24;
    mask = 0b11110000000000000000000000000000;
    instruction.BBC = (tmp & mask) >> 28;
    return instruction;
}
```

fonction fetch:

```
/*

je n'ai pas ajouté le code à mon programme, donc ça ne va pas marcher!
Calcule la nouvelle valeur du pc
Gère les différents types de conditions de branchement
*/
int32_t fetch(uint32_t pc, uint32_t *code, Instruction instruction) {
    uint32_t new_pc;
    bool take_branch = false;

    new_pc = pc;

    if (instruction.BBC == 0x8) {
        return new_pc;
    }
    if (instruction.BBC == 0x9) {
        if (reg[instruction.dest] == reg[instruction.op2]) {
            take_branch = true;
        }
    } else if (instruction.BBC == 0xA) {
        if (reg[instruction.dest] != reg[instruction.op2]) {
            take_branch = true;
        }
    } else if (instruction.BBC == 0xB) {
        if (reg[instruction.dest] <= reg[instruction.op2]) {
            take_branch = true;
        }
    } else if (instruction.BBC == 0xC) {
        if (reg[instruction.dest] >= reg[instruction.op2]) {
            take_branch = true;
        }
    } else if (instruction.BBC == 0xD) {
        if (reg[instruction.dest] < reg[instruction.op2]) {
            take_branch = true;
        }
    } else if (instruction.BBC == 0xE) {
        if (reg[instruction.dest] > reg[instruction.op2]) {
            take_branch = true;
        }
    }

    if (take_branch) {
        return new_pc;
    } else {
        return pc + 4;
    }
```

la fonction devrait utiliser code et verbose, qui ne sont pas utilisé dans cet algorithme… mais je n'ai pas tout bien compris

enfin, la fonction decode:



```c
/*
   long case pour faire toutes les possibilités
*/
void execute(Instruction instruction)
{
    uint32_t op1 = reg[instruction.op1];
    uint32_t op2;
if (instruction.IV & 0x01000000) {
  op2 = instruction.IV & 0x00FFFFFF;
} else {
  op2 = reg[instruction.op2];
}
    switch (instruction.opcode)
    {
        case 0x0:  // AND
            reg[instruction.dest] = op1 & op2;
            break;

        case 0x1:  // OR
            reg[instruction.dest] = op1 | op2;
            break;

        case 0x2:  // XOR
            reg[instruction.dest] = op1 ^ op2;
            break;

        case 0x3:  // ADD
            reg[instruction.dest] = op1 + op2;
            break;

        case 0x4:  // ADC
            reg[instruction.dest] = op1 + op2 + reg[15];
            break;

        case 0x5:  // CMP
            if (op1 == op2)
            {
                reg[15] = 1 << 0;  // BEQ
            }
            else if (op1 < op2)
            {
                reg[15] = 1 << 2;  // BLE
            }
            else
            {
                reg[15] = 1 << 5;  // BG
            }
            break;

        case 0x6:  // SUB
            reg[instruction.dest] = op1 - op2;
            break;

        case 0x7:  // SBC
            reg[instruction.dest] = op1 - op2 - !reg[15];
            break;

        case 0x8:  // MOV
            reg[instruction.dest] = op2;
            break;

        case 0x9:  // LSH
            reg[instruction.dest] = op1 << op2;
            break;

        case 0xA:  // RSH
            reg[instruction.dest] = op1 >> op2;
            break;

        default:
            printf("ERROR: invalid opcode 0x%x\n", instruction.opcode);
            exit(1);
            break;
    }
}
```

Conclusion:

Le code ne marche pas forcément, car il me manque une partie pour mieux comprendre l'énoncé. Ce n'est clairement pas un travail de futur ingénieur, et ça me fait mal de rendre un travail comme celui-ci.

Partie 2: Questions

1 - Les parties d'un processeur 64 bits qui sont également de 64 bits de large sont le bus de données, le bus d'adresse, les registres généraux et le registre d'état

2 - Les instructions qui peuvent potentiellement créer une retenue sont les instructions arithmétiques qui font des sommes ou des soustractions. exemple: ADD et SUB

3 - L'instruction ADD with Carry (ADC) est utilisée pour ajouter deux opérandes ainsi que la retenue précédemment stockée dans le registre d'état

4 - Lors de l'exécution d'une instruction de branchement (branch instruction), il faut vérifier si la

5 - Il est possible de créer une pipeline en utilisant des techniques de virtualisation