Institut d'Enseignement à Distance
Licence 1 - Architecture des ordinateurs
L'Ordinateur en papier - Chapitre 10
Mathilde Reveney - 19003734

Contents

1 L'Ordinateur en papier

1.1 Présentation du projet

Pour ce dernier chapitre, il nous est demandé de créer un émulateur (un "Ordinateur en papier") capable de réaliser des opérations à partir d'instructions données sous forme de codes opératoires représentés par des nombres en base hexadécimale. Ce projet fait intervenir les connaissances de plusieurs cours, principalement celles acquises dans ce cours ainsi que des compétences en programmation.

1.2 Choix du langage et difficultés rencontrées

J'ai choisi de coder mon ordinateur dans le langage C car c'est celui avec lequel je me sens le plus à l'aise. Il existe depuis de nombreuses années et bénéficie donc d'une grande communauté d'utilisateurs (merci Stackoverflow !). De plus la documentation est directement accessible depuis le terminal, ce qui permet de vérifier la syntaxe de certaines fonctions standards et structures de contrôle même sans accès internet (je vis à la campagne où seule la 4G est disponible, il m'arrive d'avoir des coupures...).
J'ai choisi de représenter la mémoire de l'ordinateur en papier sous forme d'un vecteur de type char pouvant contenir jusqu'à 256 nombres. J'avais commencé par créer une matrice 16X16 mais je rencontrais un problème lors de l'extraction des codes opératoires depuis le fichier. Arrivé à la fin de la ligne, mon programme chargeait le premier code de la ligne suivante à la place du dernier code de la ligne en cours. Je n'ai pas réussi à resoudre ce problème et j'ai donc choisi de "tricher" un peu en utilisant un vecteur pour stocker mes valeurs, et de l'afficher sous forme d'une matrice 16X16. L'affichage de la mémoire au début de l'exécution du programme est une façon de m'assurer que le fichier a bien été chargé et que les codes et valeurs correspondent à ceux que j'attends. C'est en quelque sorte un moyen de "debugger" mon programme.
Un autre probleme auquel j'ai dû faire face : Je n'arrivais pas à coder une boucle avec une condition d'arret convenable ; mon programme tournait indéfiniment, même après avoir produit le résultat attendu, et cela me déplaisait fortement. J'ai donc pris la liberté de créer une nouvelle instruction qui permet de mettre fin à la boucle d'évaluation des codes opératoires. Puisque nous en sommes à créer un émulateur, autant aller jusqu'au bout et lui créer un bouton "Arrêt". Cette instruction correspond au code opératoire 01.

1.3 Manuel de l'utilisateur

Pour utiliser mon Ordinateur en papier il suffit de lancer le programme compilé dans le terminal, suivi du nom d'un fichier contenant les codes opératoires et valeurs à traîter, sous format hexadécimal bien sur. Par exemple la commande "./ordi_papier 10.5.txt" lancera le programme avec le fichier contenant les instructions pour multiplier deux nombres.

1.4 Conclusion et remerciements

Ce dernier chapitre a été pour moi très enrichissant du point de vue de la programmation. Jusqu'ici j'avais toujours eu l'impression que les cours me tenaient la main tout au long du développement d'un projet en me disant quoi faire et comment y parvenir. Pour ce projet en particulier nous avions le choix non seulement du langage de programmation utilisé, mais également de l'architecture du programme et cela m'a beaucoup plu de réfléchir à cet aspect du travail, et surtout de parvenir à écrire un programme qui fonctionne.
Je tiens à remercier M. Philippe Kislin-Duval pour un cours très bien écrit, des retours précis sur les exercices de chaque chapitre et surtout une compréhension et des rapports humains qui font trop souvent défaut lorsque nous communiquons uniquement par écrans interposés.
Je dois également un grand merci au serveur Discord de la Licence informatique et tous les élèves qui participent activement à l'entraide sur les nombreux channels.

2 Les exercices

2.1 Exercice 10.1 (A Rendre)

Il manque les microcodes de la phase 2 pour le cas où OP contient E2. Complétez cette série de microcodes.

Le code opératoire E2 correspond à l'operation NAND avec adressage indirect (NAND *A). La suite des microcodes à effectuer est très similaire à celles des operations ADD *A et SUB *A puisque seule l'opération à effectuer sur les opérandes change.
Cette suite est (17) (1) (13) (6) (7) (13) (6) (7) (13) (12)

2.2 Exercice 10.2 (A Rendre)

Étant donné la logique qui existe entre les codes opératoires et le microcode... A quelle code pourrait correspondre l'instruction IN # ?

Dans les codes de l'ordinateur en papier, il apparaît que le second chiffre de chaque code indique l'opération à effectuer tandis que le premier chiffre indique le mode d'adressage. On peut également spéculer que les instructions concernant les transferts et celles concernant les entrées/sorties sont "groupées" puisqu'elles utilisent les mêmes chiffres pour indiquer le mode d'adressage. En suivant cette logique, je peux déduire que le mode d'adressage immédiat est représenté par un 0 (zéro) et l'opération IN par un 9 (neuf). L'instruction IN# correspond donc au code opératoire 09.

2.3 Exercice 10.3 (A Rendre)

Étant donné la logique qui existe entre les codes opératoires et le microcode... à quelle opération pourrait correspondre le code 01 ?

Toujours en suivant la logique décrite dans l'exercice précèdent, le chiffre 1 (un) correspond à l'opération OUT. Nous avons déjà dit que le 0 (zéro) indique un adressage immédiat. Le code opératoire 01 correspond donc à l'opération OUT #.

2.4 Exercice 10.4 (A Rendre)

Dans le programme de bootstrap, les adresses 18 à 1E sont remplies de 00. Que se passe-t-il au cours du boot-strap quand l'ordinateur exécute ces instructions ?

L'adresse 16 contient le code opératoire 10, qui signifie JUMP. Lorsqu'il arrive à cette instruction, le programme retourne donc à l'adresse contenue dans la case d'adresse 17, qui est 06. Il reprend donc son cours à l'adresse 06. Il exécute donc une boucle. L'adresse 0E quant à elle contient le code opératoire 12 qui renvoie à l'adresse contenue dans la case d'adresse 0F lorsque A = 0 (donc lorsque le programme est complètement chargé). La case d'adresse 0F contient l'adresse 1F, qui se trouve après 1E, le programme ne passe donc jamais par les cases 18 à 1E.

2.5 Le code de l'Ordinateur en papier

```
#include <stdio.h>

//  ******************************************************
//  Nom ......... : ordi_papier
//  Auteur ...... : Mathilde Reveney
//  Version ..... : V0.1 du 04/08/2022
//  Licence ..... : Cours AdO de L1
//  Compilation : gcc -Wall ordi_papier.c
//  Usage : Pour executer : ./ordi_papier 10.5.txt (par exemple)
//  ******************************************************

typedef char *str;     // type string

//les variables
struct contexte {unsigned char RS; unsigned char RM; unsigned char
    PC; unsigned char OP; unsigned char AD; char A; unsigned char
   entree; enum UAL {add, sub, NAND} cet_UAL; unsigned char
   memoire[256];} ordi_papier;//la memoire etant representee par
   des variables unsigned char, on ne peut traiter que des nombres
    allant de 0 a 255. seul A peut avoir une valeur negative
   puisque nous avons besoin de tester si A < 0 dans l'OP code 11,
    cela dit elle n'est pas ecrite en memoire lorsque c'est le cas

//les microcodes
void mc_1(struct contexte *ce_contexte) { ce_contexte->RS =
   ce_contexte->PC; }
void mc_2(struct contexte *ce_contexte) { ce_contexte->PC =
   ce_contexte->RM; } //ne pas faire la phase 3
void mc_3(struct contexte *ce_contexte) { ce_contexte->A =
   ce_contexte->RM; }
void mc_4(struct contexte *ce_contexte) { ce_contexte->RM =
   ce_contexte->A;  }
void mc_5(struct contexte *ce_contexte) { ce_contexte->OP =
   ce_contexte->RM; }
void mc_6(struct contexte *ce_contexte) { ce_contexte->AD =
   ce_contexte->RM; }
void mc_7(struct contexte *ce_contexte) { ce_contexte->RS =
   ce_contexte->AD; }
void mc_8(struct contexte *ce_contexte) { ce_contexte->RM =
   ce_contexte->entree; }
void mc_9(struct contexte *ce_contexte) { printf("Sortie:%x \n",
   ce_contexte->RM); }
void mc_10(struct contexte *ce_contexte) { ce_contexte->cet_UAL =
   add; }
```

```
void mc_11(struct contexte *ce_contexte) { ce_contexte->cet_UAL =
   sub; }
void mc_12(struct contexte *ce_contexte) {
    if (ce_contexte->cet_UAL == add) { ce_contexte->A =
       ce_contexte->A + ce_contexte->RM; }
    else if (ce_contexte->cet_UAL == sub) { ce_contexte->A =
       ce_contexte->A - ce_contexte->RM; }
    else if (ce_contexte->cet_UAL == NAND) { ce_contexte->A = !(
       ce_contexte->A && ce_contexte->RM); }
    }
void mc_13(struct contexte *ce_contexte) { ce_contexte->RM =
   ce_contexte->memoire[(int)ce_contexte->RS]; }
void mc_14(struct contexte *ce_contexte) { ce_contexte->memoire[(
   int)ce_contexte->RS] = ce_contexte->RM; }
void mc_15(struct contexte *ce_contexte) { ce_contexte->PC++; }
void mc_16(struct contexte *ce_contexte) { printf("Entrez une
   valeur: "); int buffer; scanf("%x", &buffer); ce_contexte->
   entree = (char)buffer; getchar(); } //GETCHAR() pour se
   debarasser de "\n" et avoir un tampon vide au prochain appel de
    la fonction printf("\n");
void mc_17(struct contexte *ce_contexte) { ce_contexte->cet_UAL =
   NAND; }

//les opcodes
void opcode_00(struct contexte *ce_contexte){mc_1(ce_contexte);
   mc_13(ce_contexte);mc_3(ce_contexte);}
void opcode_10(struct contexte *ce_contexte){mc_1(ce_contexte);
   mc_13(ce_contexte);mc_2(ce_contexte);}
    //passer ensuite a la phase 1
void opcode_11(struct contexte *ce_contexte){ if (ce_contexte->A <
    0){mc_1(ce_contexte);mc_13(ce_contexte);mc_2(ce_contexte);}}
   //passer ensuite a la phase 1
void opcode_12(struct contexte *ce_contexte){ if (ce_contexte->A
   == 0){mc_1(ce_contexte);mc_13(ce_contexte);mc_2(ce_contexte);}}
    //passer ensuite a la phase 1

void opcode_20(struct contexte *ce_contexte){mc_10(ce_contexte);
   mc_1(ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_21(struct contexte *ce_contexte){mc_11(ce_contexte);
   mc_1(ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_22(struct contexte *ce_contexte){mc_17(ce_contexte);
   mc_1(ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_40(struct contexte *ce_contexte){mc_1(ce_contexte);
   mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
   ce_contexte);mc_3(ce_contexte);}
void opcode_41(struct contexte *ce_contexte){mc_1(ce_contexte);
   mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
```

```
    ce_contexte);mc_9(ce_contexte);}
void opcode_48(struct contexte *ce_contexte){mc_1(ce_contexte);
    mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_4(
    ce_contexte);mc_14(ce_contexte);}
void opcode_49(struct contexte *ce_contexte){mc_1(ce_contexte);
    mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_16(
    ce_contexte);mc_8(ce_contexte);mc_14(ce_contexte);}
void opcode_60(struct contexte *ce_contexte){mc_10(ce_contexte);
    mc_1(ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_61(struct contexte *ce_contexte){mc_11(ce_contexte);
    mc_1(ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_62(struct contexte *ce_contexte){mc_17(ce_contexte);
    mc_1(ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_C0(struct contexte *ce_contexte){mc_1(ce_contexte);
    mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
    ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
    ce_contexte);mc_3(ce_contexte);}
void opcode_C1(struct contexte *ce_contexte){mc_1(ce_contexte);
    mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
    ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
    ce_contexte);mc_9(ce_contexte);}
void opcode_C8(struct contexte *ce_contexte){mc_1(ce_contexte);
    mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
    ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_4(
    ce_contexte);mc_14(ce_contexte);}
void opcode_C9(struct contexte *ce_contexte){mc_1(ce_contexte);
    mc_13(ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_13(
    ce_contexte);mc_6(ce_contexte);mc_7(ce_contexte);mc_16(
    ce_contexte);mc_8(ce_contexte);mc_14(ce_contexte);}
void opcode_E0(struct contexte *ce_contexte){mc_10(ce_contexte);
    mc_1(ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_E1(struct contexte *ce_contexte){mc_11(ce_contexte);
    mc_1(ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_E2(struct contexte *ce_contexte){mc_17(ce_contexte);
    mc_1(ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_6(ce_contexte);mc_7(
    ce_contexte);mc_13(ce_contexte);mc_12(ce_contexte);}
void opcode_01(struct contexte *ce_contexte) {printf("Fin␣du␣
    programme␣\n");}//ajout d'une instruction qui permet de sortir
    de la boucle et mettre fin au programme
```

```
int main(int k, const str argv[]) {

    enum bool {false, true};
    enum bool opcode_inconnu = false;
    int lu, cycles = 0;
    FILE * zap = fopen(argv[1], "r") ;
    if (!zap) { printf("fichier pas lisible"); }
    int buffer;
    for (int i=0; i<256 && lu != EOF; i++) { lu = fscanf(zap, "%x"
       , &buffer); ordi_papier.memoire[i] = (char)buffer; } //
       chaque valeur est chargee dans un tampon sous forme d'un
       int puis convertie en char avant d'etre enregistree dans la
        memoire
    for (int i=0; i<256; i++) { printf("%x ", ordi_papier.memoire[
       i]);
        if ((i+1)%16==0) { printf("\n"); } }//pour afficher sous
           forme de matrice 16*16

    do {
        //phase 1
        mc_1(&ordi_papier);
        mc_13(&ordi_papier);
        mc_5(&ordi_papier);
        mc_15(&ordi_papier);

        //phase 2
        if (ordi_papier.OP == 00){opcode_00(&ordi_papier);}
        else if (ordi_papier.OP == 0x10){opcode_10(&ordi_papier);}
        else if (ordi_papier.OP == 0x11){opcode_11(&ordi_papier);}
        else if (ordi_papier.OP == 0x12){opcode_12(&ordi_papier);}
        else if (ordi_papier.OP == 0x20){opcode_20(&ordi_papier);}
        else if (ordi_papier.OP == 0x21){opcode_21(&ordi_papier);}
        else if (ordi_papier.OP == 0x22){opcode_22(&ordi_papier);}
        else if (ordi_papier.OP == 0x40){opcode_40(&ordi_papier);}
        else if (ordi_papier.OP == 0x41){opcode_41(&ordi_papier);}
        else if (ordi_papier.OP == 0x48){opcode_48(&ordi_papier);}
        else if (ordi_papier.OP == 0x49){opcode_49(&ordi_papier);}
        else if (ordi_papier.OP == 0x60){opcode_60(&ordi_papier);}
        else if (ordi_papier.OP == 0x61){opcode_61(&ordi_papier);}
        else if (ordi_papier.OP == 0x62){opcode_62(&ordi_papier);}
        else if (ordi_papier.OP == 0xC0){opcode_C0(&ordi_papier);}
        else if (ordi_papier.OP == 0xC1){opcode_C1(&ordi_papier);}
        else if (ordi_papier.OP == 0xC8){opcode_C8(&ordi_papier);}
        else if (ordi_papier.OP == 0xC9){opcode_C9(&ordi_papier);}
        else if (ordi_papier.OP == 0xE0){opcode_E0(&ordi_papier);}
        else if (ordi_papier.OP == 0xE1){opcode_E1(&ordi_papier);}
```

```
            else if (ordi_papier.OP == 0xE2){opcode_E2(&ordi_papier);}
            else if (ordi_papier.OP == 0x01){opcode_01(&ordi_papier);
               return 0;}
            else {printf("OP code inconnu \n"); opcode_inconnu = true
               ;}

            cycles += 1;
            //phase 3
            if (ordi_papier.OP == 0x10) { } //pas de phase 3, on
               retourne au debut du cycle
            else if (ordi_papier.OP == 0x11 && ordi_papier.A < 0x0) {
               } //pas de phase 3, on retourne au debut du cycle
            else if (ordi_papier.OP == 0x12 && ordi_papier.A == 0x0) {
                } //pas de phase 3, on retourne au debut du cycle
            else { mc_15(&ordi_papier); }

        } while(!opcode_inconnu);//fin de la boucle en cas d'erreur d'
           OP code
        fclose(zap);

        return 0;
}
```

2.6 Exercice 10.5 (A Rendre)

Écrire un programme pour l'ordinateur en papier qui lit deux nombres et affiche leur produit.

Adresse	code	mnémonique	commentaire
00	49 30	IN 30	L'utilisateur entre une valeur, enregistrée adresse 30
02	49 31	IN 31	Une seconde valeur, enregistrée adresse 31
04	40 32	LOAD 32	On charge la valeur qui se trouve à l'adresse 32
06	60 30	ADD 30	On ajoute la valeur qui se trouve à l'adresse 30
08	48 32	STORE 32	On enregistre la nouvelle valeur à l'adresse 32
0A	40 31	LOAD 31	On charge la valeur à l'adresse 31
0C	21 1	SUB 1	On soustrait 1
0E	48 31	STORE 31	On enregistre la nouvelle valeur à l'adresse 31
10	12 14	BRZ 14	Si A = 0 on passe à l'instruction qui se trouve à l'adresse 14
12	10 04	JUMP 04	On retourne à l'instruction qui se trouve à l'adresse 04 (on boucle)
14	41 32	OUT 32	On affiche la valeur qui se trouve à l'adresse 32
16	01 00	END PROGRAM	On met fin à l'execution du programme

J'ai pris la liberté d'ajouter une instruction qui permet de mettre fin au programme lorsque l'opération à effectuer est terminée. Elle correspond à l'OP code 01.

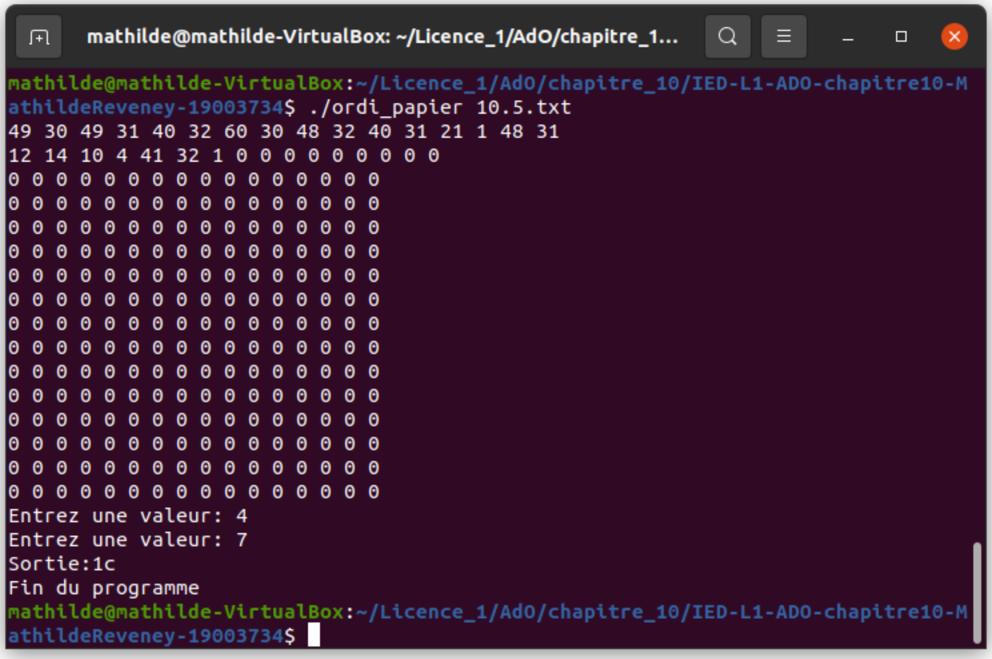

Figure 1: Le programme de l'exercice 10.5 - 04 X 07 = 1C

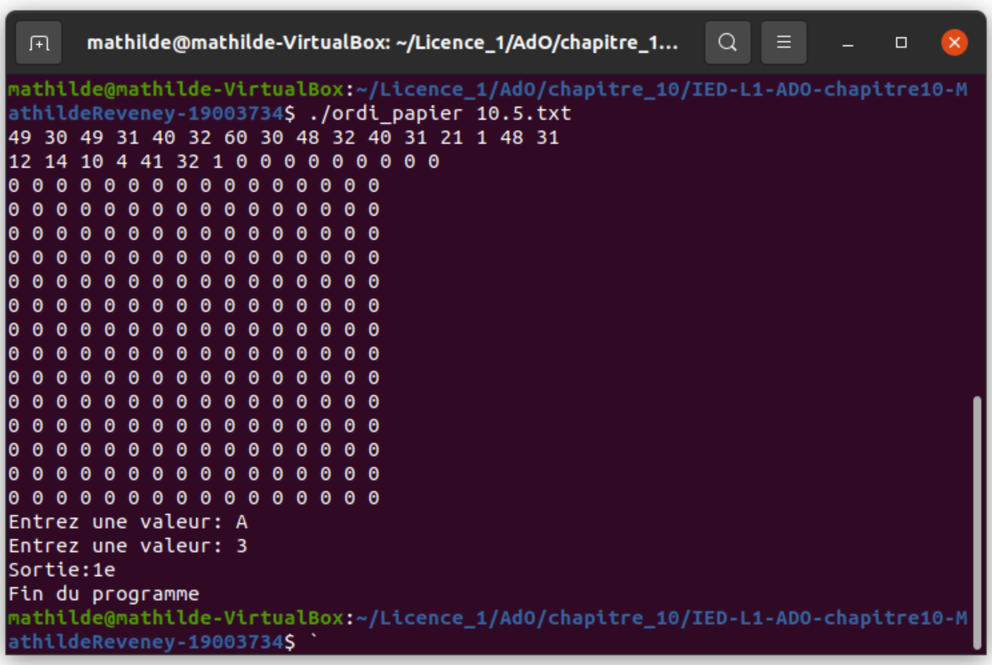

Figure 2: Le programme de l'exercice 10.5 - 0A X 03 = 1E

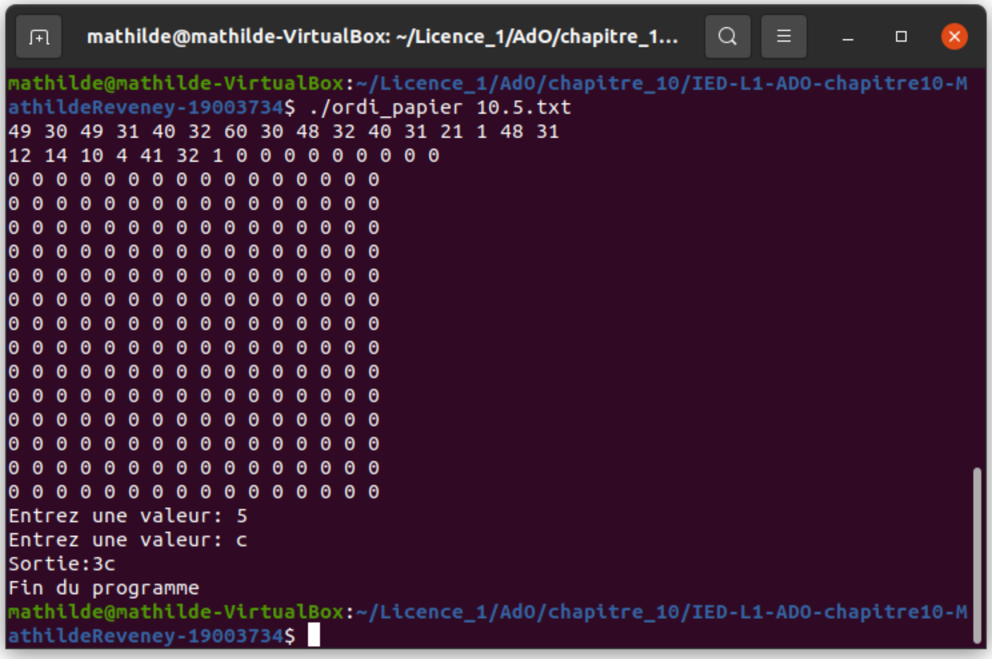

Figure 3: Le programme de l'exercice 10.5 - 05 X 0C = 3C

2.7 Exercice 10.6 (A Rendre)

En vous inspirant de l'Exercice 10.5... Écrire un programme pour l'ordinateur en papier qui lit deux nombres et affiche leur quotient "entier".

Adresse	code	mnémonique	commentaire
00	49 30	IN 30	L'utilisateur entre une valeur, enregistrée adresse 30
02	49 31	IN 31	Une seconde valeur, enregistrée adresse 31
04	40 31	LOAD 31	On charge la valeur qui se trouve a l'adresse 31
06	12 00	BRZ 00	Si A = 0 on retourne au début du programme
08	40 30	LOAD 30	On charge la valeur qui se trouve à l'adresse 30
0A	61 31	SUB 31	On soustrait la valeur qui se trouve à l'adresse 31
0C	11 18	BRN 18	Si A ¡ 0 on passe à l'instruction qui se trouve à l'adresse 18
0E	48 30	STORE 30	On enregistre la nouvelle valeur à l'adresse 30
10	40 32	LOAD 32	On charge la valeur qui se trouve à l'adresse 32
12	20 1	ADD 1	On ajoute 1
14	48 32	STORE 32	On enregistre la nouvelle valeur à l'adresse 32
16	10 8	JUMP 8	On retourne à l'instruction qui se trouve à l'adresse 08 (on boucle)
18	41 32	OUT 32	On affiche la valeur qui se trouve à l'adresse 32
1A	01 00	END PROGRAM	On met fin à l'exécution du programme

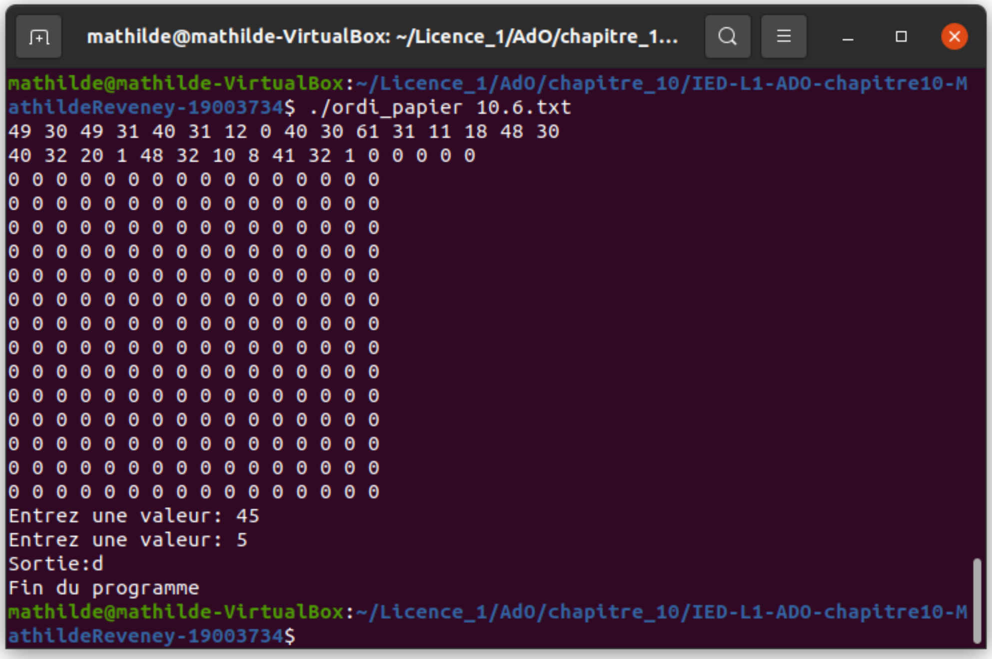

Figure 4: Le programme de l'exercice 10.6 - 45 / 05 = 0D

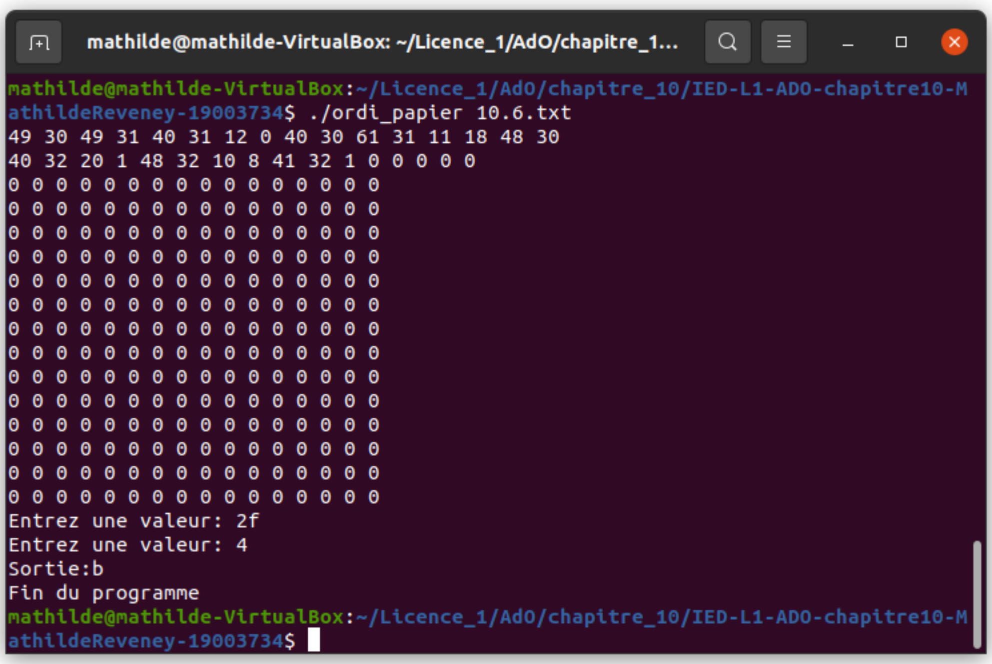

Figure 5: Le programme de l'exercice 10.6 - 2F / 04 = 0B

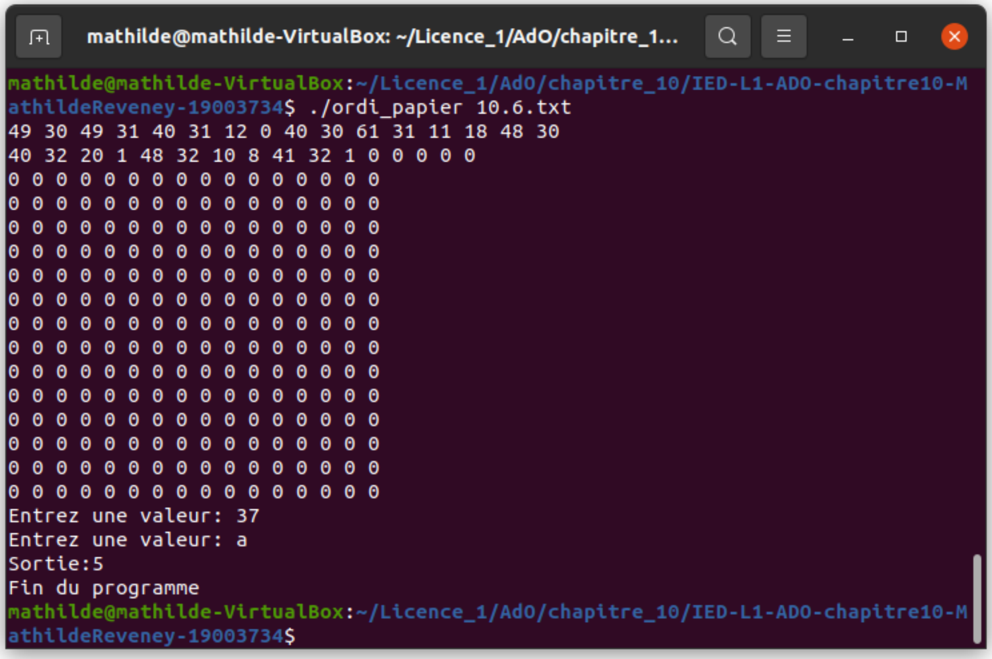

Figure 6: Le programme de l'exercice 10.6 - 37 / 0A = 05

2.8 Exercice 10.7 (A Rendre)

En vous inspirant de l'Exercice 10.5... Indiquez les mnémoniques qui correspondent aux valeurs suivantes dans la mémoire (Voir le tableau ci-dessous). Commentez le programme. Que fait-il ?

Adresse	code	mnémonique	commentaire
00	00 00	LOAD #00	Le programme ne contient que des 0 (zeros) jusqu'à l'adresse 50
4E	00 00	LOAD #00	Derniere instruction "vide"
50	49 70	IN 70	L'utilisateur entre une valeur, enregistrée adresse 70
52	40 70	LOAD 70	On charge la valeur qui se trouve à l'adresse 70
54	48 71	STORE 71	On enregistre la valeur à l'adresse 71
56	48 72	STORE 72	On enregistre la valeur à l'adresse 72
58	00 5E	LOAD #5E	On charge la valeur "5E"
5A	48 8D	STORE 8D	On enregistre la valeur à l'adresse 8D
5C	10 74	JUMP 74	On saute à l'instruction qui se trouve à l'adresse 74
5E	40 71	LOAD 71	On charge la valeur qui se trouve à l'adresse 71
60	48 72	STORE 72	On enregistre la valeur à l'adresse 72
62	40 70	LOAD 70	On charge la valeur qui se trouve à l'adresse 70
64	48 71	STORE 71	On enregistre la valeur à l'adresse 71
66	00 6C	LOAD #6C	On charge la valeur "6C"
68	48 8D	STORE 8D	On enregistre la valeur à l'adresse 8D
6A	10 74	JUMP 74	On saute à l'instruction qui se trouve à l'adresse 74
6C	41 71	OUT 71	On affiche la valeur qui se trouve à l'adresse 71
6E	10 6E	JUMP 6E	On saute à l'instruction qui se trouve à l'adresse 6E. Comme nous y sommes déjà, on est arrivé à la fin du programme qui va continuer à boucler indéfiniment sur cette adresse
70	00 00		L'adresse 70 contenait initialement 00 (double zéro), elle contient maintenant la valeur entrée par l'utilisateur au début du programme. L'adresse 71 contient cette même valeur.
72	00 00		l'adresse 72 contient également cette valeur l'adresse 73 contient toujours 00 (double zero)
74	00 00	LOAD #00	On charge la valeur "00"
76	48 73	STORE 73	On enregistre la valeur à l'adresse 73
78	40 71	LOAD 71	On charge la valeur qui se trouve à l'adresse 71
7A	12 88	BRZ 88	Si A = 0 on passe à l'instruction qui se trouve à l'adresse 88
7C	21 01	SUB #01	On soustrait 1
7E	48 71	STORE 71	On enregistre la nouvelle valeur à l'adresse 71
80	40 73	LOAD 73	On charge la valeur qui se trouve à l'adresse 73
82	60 72	ADD 72	On ajoute la valeur qui se trouve à l'adresse 72
84	48 73	STORE 73	On enregistre la nouvelle valeur à l'adresse 73
86	10 78	JUMP 78	On saute à l'instruction qui se trouve à l'adresse 78
88	40 73	LOAD 73	On charge la valeur qui se trouve à l'adresse 73
8A	48 71	STORE 71	On enregistre la valeur à l'adresse 71
8C	10 00	JUMP 00	On saute à l'instruction qui se trouve à l'adresse 00

Ce programme sert à calculer la puissance de 3 d'un nombre entré au début par l'utilisateur. Il enregistre ce nombre à 3 adresses différentes. Le premier servira de compteur pour savoir combien de fois nous avons déjà effectué les opérations nécessaires. Il calcule d'abord le carré du nombre, puis utilise le nombre obtenu pour le multiplier par le nombre entré initialement. Enfin il affiche le résultat final qui correspond au nombre choisi n^3.

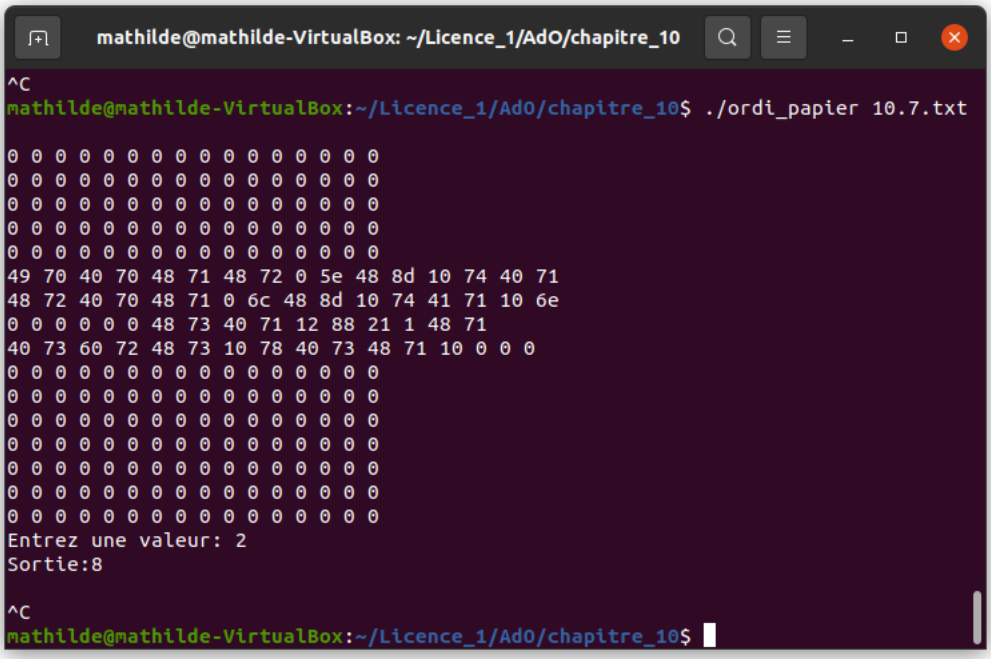

Figure 7: Le programme de l'exercice 10.7 - $2^3 = 08$

NB : Il convient de rappeler que la mémoire de l'ordinateur en papier ne peut enregistrer de nombre supérieur à 255_{10} (FF_{16}). Toute opération produisant un résultat supérieur à 255 affichera donc un nombre incorrect, sans toutefois déclencher une erreur d'exécution. L'ordinateur ne peut pas non plus traiter de nombre négatif.

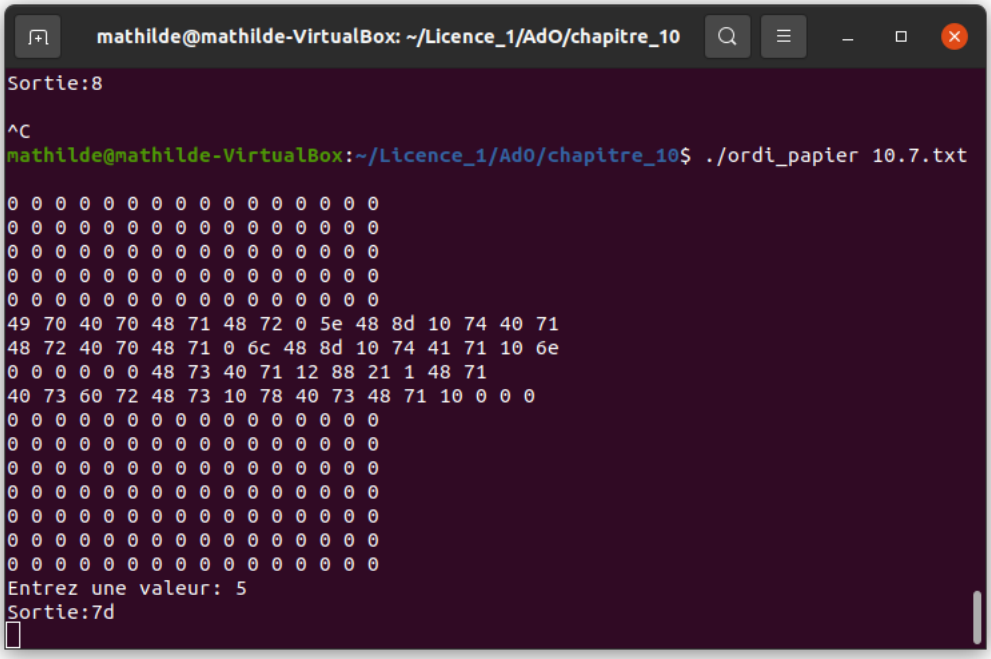

Figure 8: Le programme de l'exercice 10.7 - $5^3 = 7D$

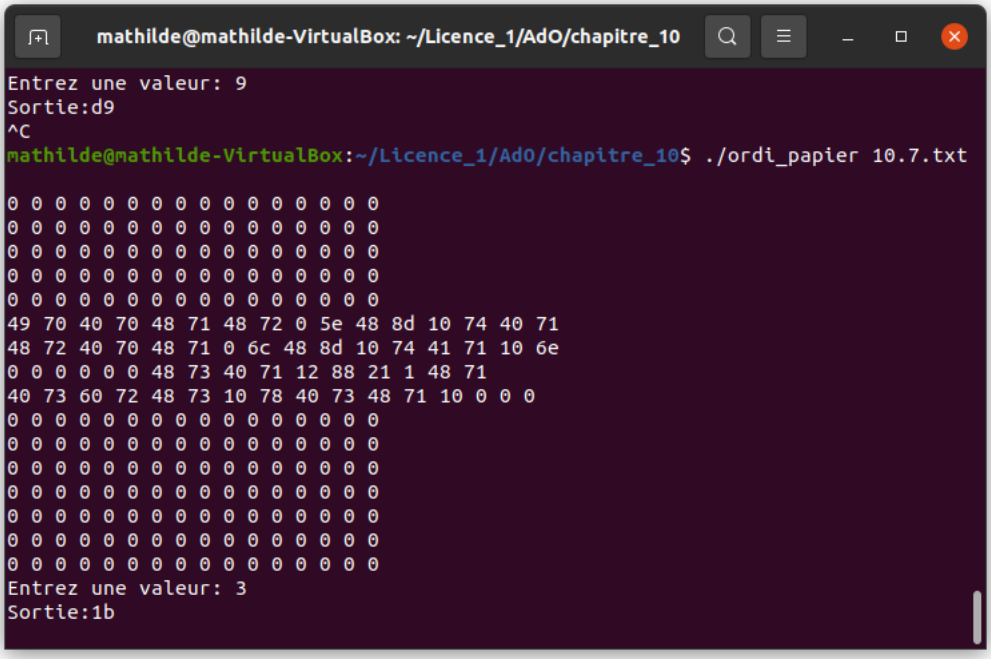

Figure 9: Le programme de l'exercice 10.7 - $3^3 = 1B$