Numéro étudiant : ———-

Question 1

a et b seront stocké dans les registres s_x car ce sont des constantes(ici on choisira arbitrairement s_0 et s_1).
Les valeurs seront ensuite copiés dans les registres a_x car elle seront passés en paramétre de la fonction ADD utilisé dans "mon_addition_entiere"(ici on choisira arbitrairement a_0 et a_1).

Remarque : Il est possible également que le compilateur remarque que a et b sont des constantes et procéde à une optimisation : les passer directement en paramètre. a et b seront alors directement stockés dans les registres a_0 et a_1. Il est important de noter que cette optimisation n'aura lieu que si la fonction "mon_addition_entiere" est déclaré dans le même fichier que a et b (lorsqu'on utilise un compilateur C par exemple).

Question 2

On va utiliser "ADDI" (add immediate) afin de charger 5 et 7 dans s_0 et s_1 en les additionnant au registre "zero" contenant 0. On utilisera ensuite "ADDI" en additionnant avec 0 afin de récréer un équivalent de la fonction "MV" (moove) qui n'existe pas dans notre jeu d'instruction.

```
1 ADDI s0, zero, 5 # Charge la constante 5 dans le registre s0
2 ADDI s1, zero, 7 # Charge la constante 7 dans le registre s1
3 ADDI a0, s0, 0   # Copie de s0 dans a0
4 ADDI a1, s1, 0   # Copie de s1 dans a1
```

Question 3

On initialise avec le code de la Question 2 (on a donc 5 et 7 dans a_0 et a_1). On va stocker le retour dans t_0, après l'exécution de ce code, nous aurons 12 dans t_0.

```
1 ADD t0, a0, a1  # Ajoute a0 et a1 dans t0, ignore l overflow
```

♠

Question 4

Cette instruction permet de modifier le registre "pc" (qui désigne l'adresse de la prochaine instruction) par l'adresse du label de la fonction "ma_fonction".

Cela induit donc que suite à l'instruction "jal", "ma_fonction" est executé.

Après avoir finit l'execution de la fonction "ma_fonction, il place la variable de retour de fonction dans un emplacement et stocke l'adresse de cet emplacement dans ra. Attention, ce n'est pas la variable qui est stocké dans ra mais l'adresse correspondant à l'endroit où est stocké la variable.

Question 5

```
1 jalr rd, rs1, offset
```

D'après la documentation, cette instruction permet de : "*Copie du compteur programme PC dans rd, puis écriture dans le compteur programme PC la valeur de rs1 où l'on a ajouté offset (12 bits avec extension de signe).*".

```
1 jalr zero, ra, 0
```

En executant cette instruction, on copie le compteur programme dans zero (ne fais rien puisque zero est un registre contenant toujours 0). Ensuite on copie la valeur de ra avec aucun offset dans le compteur programme. De cette manière, on a l'adresse du retour de fonction dans le registre PC.

♠

Question 6

```
ADDI a1, zero, 5 # Charge la constante 5 dans le registre a1
ADDI a2, zero, 7 # Charge la constante 7 dans le registre a2

jal ra,mon_addition_entiere
jal ra,done
mon_addition_entiere:
    ADD t0, a1, a2    # Ajoute a1 et a2 dans t0
    JALR zero,ra,0    # Fin de la fonction "mon_add..."
done:
    ADDI a0, t0, 0    # déplace t0 dans a0
```

Question 7

hello $\Longleftrightarrow$ 0x68656C6C6F

Avec le caractère de fin :
hello $\Longleftrightarrow$ 0x68656C6C6F00

Question 8

```
ADDI s0, zero, 0x10 # on stocke dans s0 l'adresse de départ
ADDI t0, zero, 0x68 # "h"
SB t0, 0(s0)
ADDI t0, zero, 0x65 # "e"
SB t0, 1(s0)
ADDI t0, zero, 0x6c # "l"
SB t0, 2(s0)
SB t0, 3(s0)
ADDI t0, zero, 0x6F # "o"
SB t0, 4(s0)
SB zero, 5(s0)      # "caractère de fin"
```

Ce qui permet de stocker ”h” sur 0x10, ”e” sur 0x11, ”l” sur Ox12 et 0x13, et enfin ”o” sur 0x14.

♠

Question 9

```c
#include <string.h>
#include <stdio.h>

char * strcpy2(char* destination, const char* source)
{
        size_t i = 0;
        while(source[i] != '\0')
        {
                destination[i] = source[i];
                i++;
        }
        destination[i] = '\0';
        return destination;
}

int main()
{
        //on déclare nos chaines de caractères
    char str1[] = "hello\0";
    char str2[6];

    strcpy2(str2, str1); //on appelle notre fonction

    printf("%s\n", str1);//on print la chaine source
    printf("%s\n", str2);//on print la chaine destination

    return 0;
}
```

Question 10

```assembly
ADDI a0, zero, 0x0      # adresse 1er bit destination
ADDI a1, zero, 0x10     # adresse 1er bit source

```

♠

```
4  JAL ra, strcpy
5  JAL ra,done
6
7  strcpy:
8      ADDI t0, a0, 0       # copie : adresse  1er bit destination
9      ADDI t1, a1, 0       # copie : adresse 1er bit source
10
11 .loop:
12     LB tp, 0(t1)         # Charge le caractère courant dans tp
13     SB tp, 0(t0)         # Copie le char dans la chaîne de dest
14     ADDI t0, t0, 0x01    # Incrémente le bit destination
15     ADDI t1, t1, 0x01    # Incrémente le bit source
16     BNE tp, zero, .loop # Si le char non nul, on boucle
17     JALR zero,ra,0       # Fin de la fonction
18
19 done:
```

Si on initialise avec le code de la Question 8, puis qu'on utilise le code ci-dessus, on passe de :

Memory Address	Decimal	Hex	Binary
0x00000000	0	0x00000000	0b00000000000000000000000000000000
0x00000004	0	0x00000000	0b00000000000000000000000000000000
0x00000008	0	0x00000000	0b00000000000000000000000000000000
0x0000000c	0	0x00000000	0b00000000000000000000000000000000
0x00000010	1819043176	0x6c6c6568	0b01101100011011000110010101101000
0x00000014	111	0x0000006f	0b00000000000000000000000001101111

à :

Memory Address	Decimal	Hex	Binary
0x00000000	1819043176	0x6c6c6568	0b01101100011011000110010101101000
0x00000004	111	0x0000006f	0b00000000000000000000000001101111
0x00000008	0	0x00000000	0b00000000000000000000000000000000
0x0000000c	0	0x00000000	0b00000000000000000000000000000000
0x00000010	1819043176	0x6c6c6568	0b01101100011011000110010101101000
0x00000014	111	0x0000006f	0b00000000000000000000000001101111

La fonction recopie bien dans 0x0 la chaine commençant à 0x10.

♠

Question 11

Non car on a pas la place d'écrire "hello" étant sur 6 octets (car on écrit "hello\0") soit 48 bits sur des espaces de 32 bits. L'overflow dans mon code par exemple écrase le premier octet source pour écrire le 'o' et écrase le second octet source pour écrire le '\0'.

Question 12

L'overflow va écraser la chaine source puisque que la chaine fait 6 octets et il n'y a que 4 octets de place sur 32bits (comme expliqué précedemment à la Question 11).
Avec le simulateur, on passe de ça :

Memory Address	Decimal	Hex	Binary
0x00000000	0	0x00000000	0b00000000000000000000000000000000
0x00000004	1819043176	0x6c6c6568	0b01101100011011000110010101101000
0x00000008	111	0x0000006f	0b00000000000000000000000001101111

à ça :

Memory Address	Decimal	Hex	Binary
0x00000000	1819043176	0x6c6c6568	0b01101100011011000110010101101000
0x00000004	1819017327	0x6c6c006f	0b01101100011011000000000001101111
0x00000008	111	0x0000006f	0b00000000000000000000000001101111

Question 13

Plusieurs méthodes possible :

- Regarder la taille de la chaine de caractère destination et ne copier que les premiers à l'exceptions du dernier copié qu'on peux remplacer par un '\0' afin de finir la string.
- Choisir l'adresse de destination, et stocker l'adresse du premier bit de destination dans a0 par exemple. Après avoir executé la fonction on sait ou trouver la copie et la copie est entière.
- Et bien d'autres solutions auxquelles je n'ai pas pensé...