Week 07 Tutorial Questions

1. How is the assignment going?

   Does anyone have hints or advice for other students?

2. Give MIPS directives to represent the following variables:

   a. `int v0;`
   b. `int v1 = 42;`
   c. `char v2;`
   d. `char v3 = 'a';`
   e. `double v4;`
   f. `int v5[20];`
   g. `int v6[10][5];`
   h. `struct { int x; int y; } v7;`
   i. `struct { int x; int y; } v8[4];`
   j. `struct { int x; int y; } *v9[4];`

   Assume that we are placing the variables in memory, at an appropriately aligned address, and with a label which is the same as the C variable name.

3. Translate this C program to MIPS assembler.

```c
int max(int a[], int length) {
    int first_element = a[0];
    if (length == 1) {
        return first_element;
    } else {
        // find max value in rest of array
        int max_so_far = max(&a[1], length - 1);
        if (first_element > max_so_far) {
            max_so_far = first_element;
        }
        return max_so_far;
    }
}
```

4. Translate this C program to MIPS assembler.

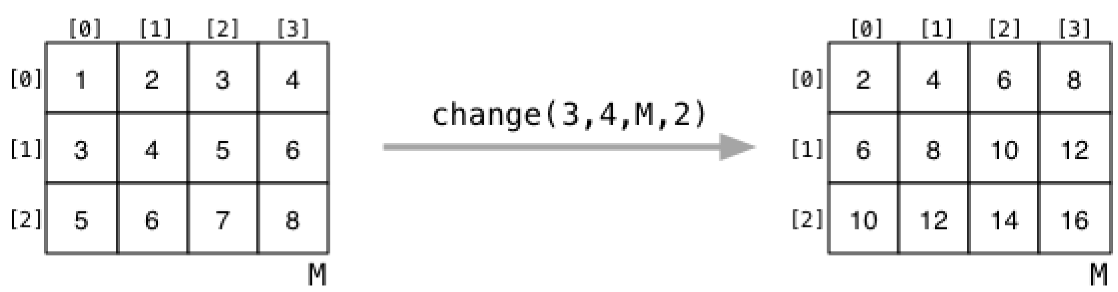
```c
#include <stdio.h>

char flag[6][12] = {
    {'#', '#', '#', '#', '#', '.', '.', '#', '#', '#', '#', '#'},
    {'#', '#', '#', '#', '#', '.', '.', '#', '#', '#', '#', '#'},
    {'.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.'},
    {'.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.'},
    {'#', '#', '#', '#', '#', '.', '.', '#', '#', '#', '#', '#'},
    {'#', '#', '#', '#', '#', '.', '.', '#', '#', '#', '#', '#'}
};

int main(void) {
    for (int row = 0; row < 6; row++) {
        for (int col = 0; col < 12; col++)
            printf ("%c", flag[row][col]);
        printf ("\n");
    }

}
```

5. Consider the following operation that multiplies all of the elements in a matrix by a constant factor:



This operation could be rendered in C99-standard C as

```
void change (int nrows, int ncols, int M[nrows][ncols], int factor)
{
    for (int row = 0; row < nrows; row++) {
        for (int col = 0; col < ncols; col++) {
            M[row][col] = factor * M[row][col];
        }
    }
}
```

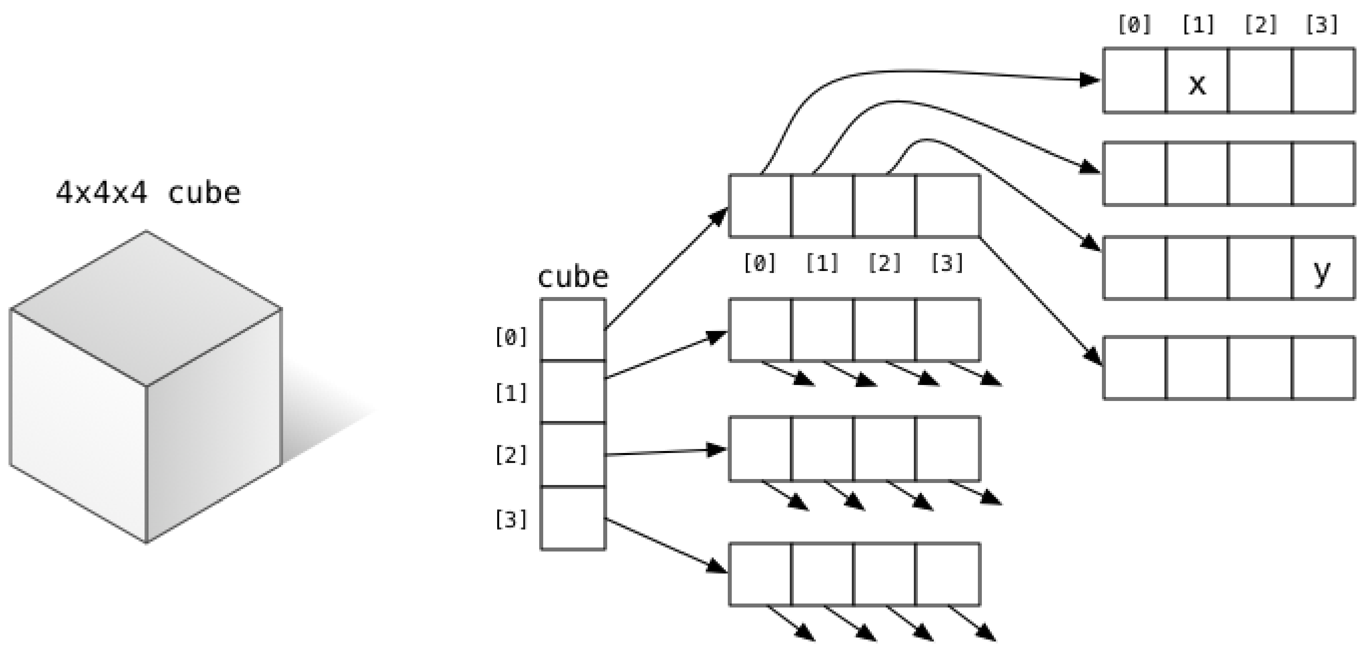
Write a function in MIPS assembly equivalent to the above C code. Assume that the arguments are placed in the `$a?` registers in the order given in the function definition. e.g., the function could be called as follows in MIPS:

```
    li   $a0, 3
    li   $a1, 4
    la   $a2, M
    li   $a3, 2
    jal  change
```

Where `M` is defined as:

```
    .data
M:  .word 1, 2, 3, 4
    .word 3, 4, 5, 6
    .word 5, 6, 7, 8
```

6. Consider the following 3-d array structure:



The cube could be implemented as an array of pointers, each of which points to a slice of the cube. Each slice of the cube is also an array of pointers, and each of those points to an array of `int` values.

For example, in the diagram above, the cell labelled `x` can be accessed as `cube[0][0][1]`, and the cell labelled `y` can be accessed as `cube[0][2][3]`.

a. Write MIPS assembler directives to define a data structure like this.
b. Write MIPS assembler code to scan this cube, and count the number of elements which are zero.

In other words, implement the following C code in MIPS.

```
int cube[4][4][4];

int nzeroes = 0;
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
        for (int k = 0; k < 4; k++)
            if (cube[i][j][k] == 0)
                nzeroes++;
```

7. For each of the following `struct` definitions, what are the likely offset values for each field, and the total size of the `struct`:

a.
```
struct _coord {
    double x;
    double y;
};
```

b.
```
typedef struct _node Node;
struct _node {
    int value;
    Node *next;
};
```

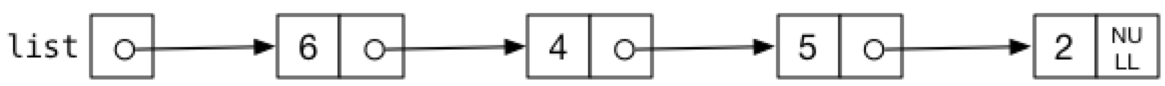
c.
```
struct _enrolment {
    int stu_id;         // e.g. 5012345
    char course[9]:     // e.g. "COMP1521"
    char term[5];       // e.g. "17s2"
    char grade[3];      // e.g. "HD"
    double mark;        // e.g. 87.3
};
```

d.
```
struct _queue {
    int nitems;     // # items currently in queue
    int head;       // index of oldest item added
    int tail;       // index of most recent item added
    int maxitems;   // size of array
    Item *items;    // malloc'd array of Items
};
```

Both the offsets and sizes should be in units of number of bytes.

8. **Challenge exercise, for those who have seen linked data structures in C.**

Consider a linked list



which could be defined in MIPS as:

```
    .data
list:   .word node1
node1:  .word 6, node2
node2:  .word 4, node3
node3:  .word 5, node4
node4:  .word 2, 0
```

Write a MIPS function that takes a pointer to the first node in the list as its argument, and returns the maximum of all of the values in the list. In other words, write a MIPS version of this C function:

```
typedef struct _node Node;
struct _node { int value; Node *next; };

int max (Node *list)
{
    if (list == NULL) return -1;
    int max = list->value;
    Node *curr = list;
    while (curr != NULL) {
        if (curr->value > max)
            max = curr->value;
        curr = curr->next;
    }
    return max;
}
```

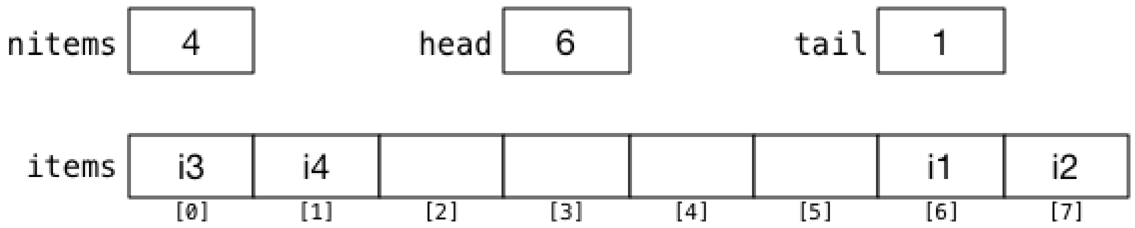
You can assume that only positive data values are stored in the list.

9. If we execute the following small MIPS program:

```
   .data
x: .space 4
   .text
   .globl main
main:
   li  $a0, 32768
   li  $v0, 1
   syscall
   sw  $a0, x
   lh  $a0, x
   li  $v0, 1
   syscall
   jr  $ra
```

... we observed that the first `syscall` displays `32768`, but the second `syscall` displays `-32768`. Why does this happen?

10. FIFO queues can be implemented using circular arrays. For example:



And the C code to manipulate such a structure could be:

```
// place item at the tail of the queue
// if queue is full, returns -1; otherwise returns 0
int enqueue (int item)
{
    if (nitems == 8) return -1;
    if (nitems  > 0) tail = (tail + 1) % 8;
    queue[tail] = item;
    nitems++;
    return 0;
}

// remove item from head of queue
// if queue is empty, returns -1; otherwise returns removed value
int dequeue (void)
{
    if (nitems == 0) return -1;
    int res = queue[head];
    if (nitems  > 1) head = (head + 1) % 8;
    nitems--;
    return res;
}
```

Assuming that the items in the queue are `int`s, and the following MIPS data definitions are used:

```
nitems: .word 0
head:   .word 0
tail:   .word 0
items:  .space 32
```

... implement the `enqueue()` and `dequeue()` functions in MIPS.

Use one of the standard function prologues and epilogues from lectures.

COMP1521 20T2: Computer Systems Fundamentals is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1521@cse.unsw.edu.au
CRICOS Provider 00098G