Computer Organisation
Name: Johri Aniket Manish
Roll.no: CS22B028
IIT Tirupati

Lab: 02

This report answers the observation, explanation and runtime of problem set sequentially.

1. *Write a C program to find out if your machine is Big-Endian or Little-Endian.*

```c
#include<stdio.h>

int main()
{
  unsigned int a=1;
  char *c=(char *)&a;
  printf("%x %x %x %x\n",c[0],c[1],c[2],c[3]);
  int x = 0x12345678;
  char *b=(char *)&x;
  printf("%x %x %x %x\n",b[0],b[1],b[2],b[3]);
  if(*c)
  {
    printf("Little endian\n");
  }
  else
  {
    printf("Big endian\n");
  }
  return 0;
}
```

Output:

```
1 0 0 0
78 56 34 12
Little endian
```

Here the unsigned int a has been given the value 1 and c is a pointer to character that has been assigned the address of a. %x c[0] prints the individual bytes of int a in hexadecimal form. Since a is of 4 bytes, it prints four hexadecimal values corresponding to each byte. The other way is to print each value after storing it in hexadecimal form as discussed in class.

Finally, when *c or c[0] is 1, it shows the output as Little endian since the LSB in Little endian machine architecture gets the value 1.

2. *Write a C program to find out if your machine is Big-Endian or Little-Endian using the Union data structure.*

```c
#include<stdio.h>

union q2 {
    int a;
    char c[4];
};

int main()
{
    union q2 q;
    printf("Size of union: %ld\n",sizeof(q));
    q.a = 1;
    printf("%x %x %x %x\n",q.c[0],q.c[1],q.c[2],q.c[3]);
    int x = 0x12345678;
    char *b=(char *)&x;
    printf("%x %x %x %x\n",b[0],b[1],b[2],b[3]);
    if(q.c[0])
    {
        printf("Little endian\n");
    }
    else
    {
        printf("Big endian\n");
    }
    return 0;
}
```

Output:

```
Size of union: 4
1 0 0 0
78 56 34 12
Little endian
```

Here the union data structure takes the maximum byte value as the overall memory and so the size of the union will be 4 bytes. So both the int and char array shares the same memory location. The integer a inside the union q is assigned the value 1. It then prints the individual bytes of a using the character array c. The c[0] is the value of the first byte of the character array c inside the union. If it's non-zero, it means the system is Little endian. Otherwise, it's Big endian.

3. *Write a C program to convert a Big-Endian to Little-Endian and vice-versa. Clearly print the value stored along with its address.*

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

int changeEndian(int n) {
    return ((n & 0xff) << 24) | ((n & 0xff00) << 8) | ((n & 0
   xff0000) >> 8) | ((n & 0xff000000) >> 24);
}

int main() {
    int n = 0x12345678;
    int m = 0x87654321;
    printf("LE is 0x%x and it's BE is 0x%x\n",n,changeEndian(n));
    printf("BE is 0x%x and it's LE is 0x%x\n",m,changeEndian(m));
    return 0;
}
```

Output:

LE is 0x12345678 and it's BE is 0x78563412
BE is 0x87654321 and it's LE is 0x21436587

The changeEndian function takes the value of the int and returns the following.
1) n & 0xff will give the last(rightmost) byte of the hexadecimal number
   and then it is shifted 24 bits to left to make it first(leftmost) byte.
2) n & 0xff00 will give the right middle byte of the hexadecimal number
   and then it is shifted 8 bits to left to make it left middle byte.
3) n & 0xff0000 will give the left middle byte of the hexadecimal number
   and then it is shifted 8 bits to right to make it right middle byte.
4) n & 0xff000000 will give the first(leftmost) byte of the hexadecimal number
   and then it is shifted 24 bits to right to make it last(rightmost) byte.
5) At last it returns the final integer doing OR operation to all above steps.

4. *Download the attached assembly.cpp, and compile using the command mentioned in the first line of the file. What does the output mean to you?*

[NOTE: Reference GPT]

-g:It includes debugging information in the compiled program, which later can be debugged using tools like GDB(GNU Debugger).
-O0:Turns off all optimization,making compiled code more readable for debugging.
-Wa,-aslh: Produce an assembly listing with high-level source interspersed.
-Wa: This option is followed by a comma-separated list of options to be passed to the assembler.
-aslh: These are individual options for the assembler:
    -a: Includes annotated source code in the assembly output.
    -s: Includes source code interspersed with the assembly in the output.
    -l: Includes the generated assembly code.

So gathering information and looking at the O/P, it can be observed that the file is first located and read and then it is parsed in small portions. The O/P generated will be the assembly code for the corresponding assembly instructions for the C++ source code. It also consists the information of locations of registors and value stored in it. It shows each and every piece of detailed info that is generated during compiling, parsing and decoding.

5. *Answer the subquestions.*

```c
#include<stdio.h>
#include<stdint.h>
static inline uint32_t convert(uint32_t x)
{
    return (x>>24) | ((x>>8)&0xff00) | ((x<<8)&0xff0000) | (x<<24);
}

static inline uint32_t to_bigendian(uint32_t x)
{
    union
    {
        int i;
        char c;
    } u = {1};
    return u.c ? x : convert(x);
}

static inline uint32_t to_littleendian(uint32_t x)
{
    union
    {
        int i;
        char c;
    } u = {1};
    return u.c ? convert(x) : x;
}

int main()
{
    uint32_t x = 0x12345678;
    printf("0x%08x\n", convert(x));
    printf("0x%08x\n", to_bigendian(x));
    printf("0x%08x\n", to_littleendian(x));
    return 0;
}
```

Output:

```
0x78563412
0x12345678
0x78563412
```

(1) The complete code is as below:
1) 0xff0000
2) x
3) convert(x)
4) convert(x)
5) x

(2) The purpose of this code is to convert the unsigned 32 bit integer from host machine's endianess to other endianess.

(3) The steps to convert endianess are same as mentioned above using bit masking. The additional thing about thing code is that if the number is already in Big endian, then it remains as it or else it converts to Little endian and vice versa.

6. *Answer the subquestions.*
[NOTE: Reference Valgrind Documentation] Valgrind Documentation

(1) The bug in the code is that we are trying to access the 11th element from the array which doesn't even exist as we have allocated only 10 spaces for the array ie from 0 to 9. Another bug is that we have not freed the memory.

(2) and (3) It indicates that we are trying to access some memory which is not allocated and so it throws error as shown below. Also since heap is not free, it shows where leaked memory was allocated as shown below.

```
==17434==
==17434== Invalid write of size 4
==17434==    at 0x109195: f (in /home/aniket/Desktop/CS22B028/Computer_Organisation_Lab/Lab2/q6.out)
==17434==    by 0x1091AA: main (in /home/aniket/Desktop/CS22B028/Computer_Organisation_Lab/Lab2/q6.out)
==17434==  Address 0x4a7e068 is 0 bytes after a block of size 40 alloc'd
==17434==    at 0x4845828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==17434==    by 0x10917E: f (in /home/aniket/Desktop/CS22B028/Computer_Organisation_Lab/Lab2/q6.out)
==17434==    by 0x1091AA: main (in /home/aniket/Desktop/CS22B028/Computer_Organisation_Lab/Lab2/q6.out)
==17434==
```

Figure 1: Indicating that unallocated memory is being accessed.

```
==17434==
==17434== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==17434==    at 0x4845828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==17434==    by 0x10917E: f (in /home/aniket/Desktop/CS22B028/Computer_Organisation_Lab/Lab2/q6.out)
==17434==    by 0x1091AA: main (in /home/aniket/Desktop/CS22B028/Computer_Organisation_Lab/Lab2/q6.out)
==17434==
```

Figure 2: Memory is not freed after allocation in the heap.

(4) -The number at beginning of each line represents the PID number ie Process ID number.
-The first line ("Invalid write...") tells you what kind of error it is. Here, the program wrote to some memory it should not have due to a heap block overrun.
-Below the first line is a stack trace telling you where the problem occurred. Stack traces can get quite large, and be confusing, especially if you are using the C++ STL. Reading them from the bottom up can help.

If the stack trace is not big enough, use the --num-callers option to make it bigger.
-The code addresses (eg. 0x109195) are usually unimportant, but occasionally crucial for tracking down weirder bugs.
-The stack trace tells you where the leaked memory was allocated. Memcheck cannot tell you why the memory leaked, unfortunately.

- ps -ef | grep 17434 When this command is run on another terminal after adding sleep function, we get different PID values ie 17436, 17438, 17440, 17442 as shown in the figure.

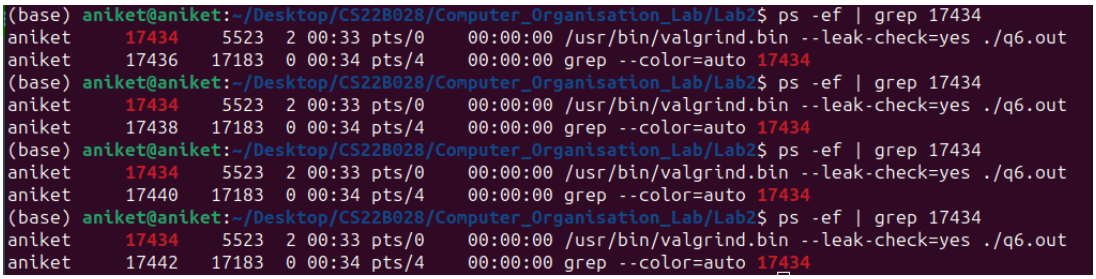

Figure 3: Different PID values.

7. *Answer the subquestions.*

After changing processor to RISC V 32 bit and single cycle processor:
addi x1, x0, 10
The value of x1 becomes 0x0000000a
After reseting F3
addi ra, zero, 10
The value of x1 still becomes 0x0000000a. This is because ra is alias of x1 and zero is alias of x0 and so it is nothing but same command as above.
Assembly and its O/P for f = (g+h) - (i+j); is as follows:

```
#let take value of g,h,i,j as 10,5,2,3 respectively

addi x10, x0, 10     # g = 10 in x10
addi x11, x0, 5      # h = 5 in x11
add x10, x10, x11    # g = g+h in x10
addi x12, x0, 2      # i = 2 in x12
addi x13, x0, 3      # j = 3 in x13
add x12, x12, x13    # i = i+j in x12
sub x10, x10, x12    # g = g-i in x10
```

x9	s1	0x00000000
x10	a0	0x0000000a
x11	a1	0x00000005
x12	a2	0x00000005
x13	a3	0x00000003
x14	a4	0x00000000

Figure 4: Output

8. *Answer the question.*
[NOTE: Reference Decompiler Explorer] Decompiler Explorer

This is a Decompiler. A decompiler does the opposite of a compiler. It takes binaries and turns them back into source code (with varying degreees of success depending on the compiler, compiler settings, language, architecture, complexity, and many other factors).

The simple C code for adding 2 number is:

```
#include<stdio.h>

int main()
{
    int a=3;
    int b=4;
    printf("The sum of a and b is %d\n",a+b);
    return 0;
}
```

This will generate some binary file. That binary file is again converted to the code. Clicking on above link will show the output for the above code.