Gouthami . C
A20445435

2.7. Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

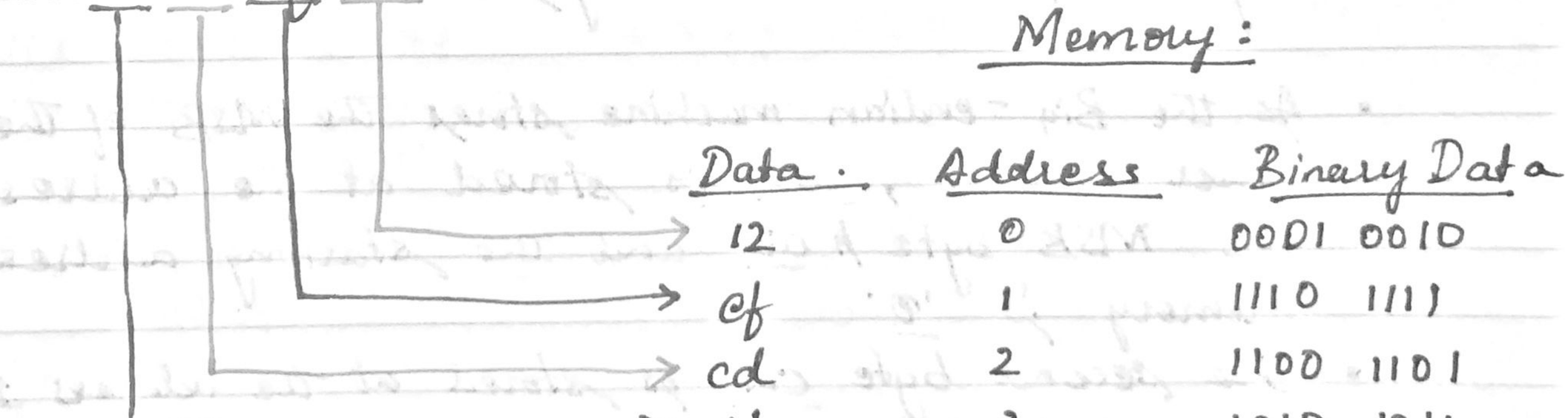
answer: Consider the given information as follows:

- The given data is 0xabcdef12.
- Starting address of the memory is 0.
- The given word size is 4 bytes.

* Little-endian machine: The least significant bit (LSB) of the data is placed at the lower address of the memory

- Arrangement of the data in memory of a little-endian machine:

The below figure shows the arrangement of the given data in the memory. (View the data bytes from right to left to store the memory)



- As the little-endian machine stores the LSB of the data in the lower address, 12 is stored at the address 0. As the LSB byte is 12 and the starting address is 0.

- The second byte ef is staed at the address 1
- The thIRd byte cd is stoled at the address 2.
- The fourth byte ab is stored at the address 3

* Big-endian Machine: The most significant Bit (MSB) of the data is placed at the lower address of the memory

- Arrangement of the data in memory of a Big-endian Machine:

The above figure shows the arrangement of the given data in the memory. (view the data ~~bytes~~ bytes from left to right to store the memory)

- As the Big-endian machine stores the MSB of the data in the lower address, ab is stored at the address 0. Because the MSB byte is ab and the starting address of the memory is '0'
- The second byte cd is stored at the address 1.
- The thIRd byte ef is stored at the address 2.
- The fourth byte 12 is stored at the address 3.

2.14 Provide the type and assembly language instruction for the following binary value : 0000 0010 0001 0000 1000 0000 0010 0000 two

answer: Last 6-bits correspond to the opcode of the ADD instruction by Assuming MIPS assembly set.

000000	10000	10000	10000	00000	100000
Op=0	Rs=16	Rt=16	Rd=16	shamt=0	funct=0x20 (add)
(add)	($s0)	($s0)	($s0)		

Thus this is a R-type instruction : add $s0, $s0, $s0.

2.15 Provide the type and hexadecimal representation of the following instruction : sw $t1, 32($t2).

answer: Comparing the given instruction with:

sw rt, offset(base).

Bits 31-26 : 101011 → opcode for sw
Bits 25-21 : base → $t2 → 10 decimal → 01010 binary
Bits 20-16 : rt → $t1 → 9 decimal → 01001 binary
Bits 15-0 : offset → 32 decimal → 0000,0000,010000 binary

∴ So the binary equivalent of the instruction is :

101011 01010 01001 00000 00000 010000

2.19. Assume the following register contents

$t0 = 0xAAAAAAAA , $t1 = 0x12345678

2.19.1. For the register values shown above, what is the value of $t2 for the following sequence of instructions?

```
sll   $t2 , $t0 , 44
or    $t2 , $t2 , $t1.
```

answer:

```
sll   t2 , t0 , 4.
```

1010 1010 1010 1010 1010 1010 1010 1010

shifting 4 bits left..

t_2 = 1010 1010 1010 1010 1010 1010 1010 0000

OR 0001 0010 0011 0100 0101 0110 0111 1000

t_2 = 1011 1010 1011 1110 1111 1110 1111 1000

= B A B E F E F 8

2.19.2 For the register values shown above, what is the value of $t2 for the following sequence of instructions?

```
sll   $t2 , $t0 , 4
andi  $t2 , $t2 , -1
```

answer: sll 1010 1010 1010 1010 1010 1010 1010 0000

-1 in binary is represented by taking 2's compliment

0001 → 1110 + 1 → 1111

andi 1010 1010 1010 1010 1010 1010 1010 0000

1111 1111 1111 1111 1111 1111 1111 1111

t_2 = 1010 1010 1010 1010 1010 1010 1010 0000

= A A A A A A A 0

2.19.3. For the register values shown above, what is the value of $t2 for the following sequence of instructions ?

```
srl     $t2 , $t0 , 3
andi    $t2 , $t2 , 0XFFEF
```

answer: 0001 0101 0101 0101 0101 0101 0101 0101 | 010 → Right shift

t2 = 0001 0101 0101 0101 0101 0101 0101 0101

andi = 0000 0000 0000 0000 1111 1111 1110 1111

0000 0000 0000 0000 0101 0101 0100 0101

t2 = 5 5 4 5

2.23 Assume $t0 holds the value 0x00101000. What is the value of $t2 after the following instructions ?.

```
        slt   $t2 , $0 , $t0
        bne   $t2 , $0 , ELSE
        j     DONE
ELSE :  addi  $t2 , $t2 , 2
DONE :
```

answer: Register $t0 contains a value as 0x00101000

slt $t2 , $0 , $t0

slt compares the values in the registers and sets the value to 1 if value in first register is less than the second value. Otherwise, it sets the value to 0. Since 0 is less than $t0 (0x00101000), 1 is set into $t2.

bne $t2 , $0 , ELSE

$t2 contains value 1. Since 1 is not equal to 0, ELSE instruction is executed.

ELSE : addi $t2 , $t2 , 2

This instruction adds the value, 2 to the value in $t2 (1) and stores it in $t2.

Adding value of $t2 and 2:

```
  0000 0000 0000 0000 0000 0000 0000 0001
+ 0000 0000 0000 0000 0000 0000 0000 0010
= 0000 0000 0000 0000 0000 0000 0000 0011
```

The hexadecimal value of $t2 after executions of instructions in binary :

0000 0000 0000 0000 0000 0000 0000 0011

Therefore, the value of $t2 after the instructions is 3.

4.27 Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of a, b, i and j are in registers $s0, $s1, $t0 and $t1, respectively. Also, assume that registers $s2 holds the base address of the array D.

```
for (i=0 ; i<a ; i++)
    for (j=0 ; j< b; j++)
        D[4*j] = i + j;
```

answer:

```
        addi $t0 , $zero , -1 // i = -1
for1tst: addi $t0 , $t0 , 1 // i += 1
        slt  $t2 , $t0 , $s0 // i < a
        beq  $t2 , $zero , exit // if $t2 == 0, go to exit
        and  $t1 , $t1 , $zero // j=0
```

```
for2tst : slt  $t2 , $t1 , $s1   // j<b
          beq  $t2 , $zero , for1tst // if $t2==0,
                                     branch to for1tst
          add $t2 , $t0 , $t1 // reg $t2 = i+j.
          sll  $t4 , $t1 , 4  // reg $t4 = 4*j.
          add $t3 , $s2 , $t4 // reg $t3 = &D[4*j]
          sw   $t2 , 0($t3)   // stole $s2 in
                                address pointed by $t3
          addi $t1 , $t1 , 1  // j+=1
          j for2tst           // jump to for2tst
exit:
```

2.39 Write the MIPS assembly code that creates the 32-bit constant 0010 0000 0000 0001 0100 1001 0010 0100 two and stores that value to register $t1.

answer: Consider the given 32-bit constant as shown below:

(0010 0000 0000 0001 0100 1001 0010 0100)$_2$

store the value to the register $t1 with the below MIPS instructions:

```
lui $t1, 0x2001  // loads the upper immediate value
ori $t1, $t1, 0x4924 // or immediate with the lower
                        value.
```

0010 0000 0000 0001

0100 1001 0010 0100

The instruction load Upper Immediate (lui) loads the MSB 2 bytes into the register $t1. The instructi OR immediate (ori) performs an operation with the register content $t1 and LSB 2 bytes. ∴ the 32-bit word stored in the register $t1.

2.42 If the current value of the PC is 0X1FFFF000, can you use a single branch instruction to get to the PC address as shown in Exercise 2.39 ?

<u>answer</u>: The current value of the PC is 0X1FFFF000.
The branch instruction can branch within + or − 2^{15} words of the PC. It is equivalent to 2^{17} in decimal.

The 2^{17} in decimal is equivalent to 20000 bytes in hexa-decimal. Now, the current value of the PC is added to the branch as shown below:

0X1FFFF000 + 20000 = 2001F000

The single branch Instruction needs to get the PC address as "0010 0000 0000 0001 0100 1001" "0010 0100" which is equivalent to 20014924 in hexadecimal. The address 20014924 is well - within the range of branch 2001F000 which is the current PC address calculated above.

Therefore, a single branch instruction can be used to get the PC address <u>20014924</u>