# Indian Institute of Information Technology Allahabad

**Review Test - Second Component (C2) (April 2019)**

*Second semester B.Tech (IT): Section A*

| Course Name | Course Code | Date of Exam | MM | Time |
|---|---|---|---|---|
| Computer organization and Architecture | ICOA230C | April 23, 2019 | 30 | 1 Hr |

***Important Instructions**: All questions are compulsory.*

1. **(4+4+2 = 10 marks) (*Memory Organization and Cache*)**:

   (a) **Professor Snape** has just discussed about the *Pepperup* potion at *Hogwarts*. **Harry Potter** wants to store the name "*Pepperup*" in his magic kit as back up, lest he forgets. Harry's kit is a little endian machine which stores data in the memory in the form of 32-bit value. Assume the memory to be byte addressable, how will Harry store *0xPepperup*. Write the contents of memory locations.Start the address at any number (use base 10 to write the address). Harry's friend **Hermione Granger** also wants to store "*Pepperup*" in her magic kit. However, she wishes to load *0xPepperup* into a register of her machine. She knows her machine is little endian. Should she place *0xPepperup* in the register or *0xuperppPe* into the register (or neither)?

**Solution:**

```
 Address        Value
+--------------+------------+
|   1000       | 1011 1110  | (up)
+--------------+------------+
|   1001       | 1101 1010  | (er)
+--------------+------------+
|   1002       | 1110 1111  | (pp)
+--------------+------------+
|   1003       | 1100 1010  | (Pe)
+--------------+------------+
```

**Solution:**

```
0xPepperup, of course. Endianness doesnt matter when you are loading
into a register.  Endianness is a way of breaking up a multi-byte
quantity into bytes so it can be stored as individual bytes
in memory or in a file.

Even when it's stored in memory (in any endianness),
the endianness tells which byte is most significant.
Thus, in any endianness, 0xPe would be the MSByte,
which means it should be the uppermost byte of a register.
```

   (b) Consider a direct mapped cache with 16 cache lines, indexed 0 to 15, where each cache line can contain 32 integers (block size : 128 bytes). Consider a two-dimensional, 32x32 array of integers a . This array is laid out in memory so that a [0;0] is next to a [0;1], and so on. Assume the cache is initially empty, but that a [0;0] maps to the first word of cache line 0. Consider the following column-first traversal:

```
int sum = 0;
 for (int i = 0; i < 32; i++) {
 for( int j=0; j < 32; j++) {
 sum += a[i,j];
```

```
  }
}
```

and the following row-first traversal :

```
int sum = 0;
  for (int i = 0; i < 32; i++) {
  for( int j=0; j < 32; j++) {
  sum += a[j,i];
  }
}
```

Compare the number of cache misses produced by the two traversals, assuming the oldest cache line is evicted first. Assume that i , j , and sum are stored in registers. Assume that no part of array, a , is saved in registers. It is always stored in the cache.

**Solution:**

```
 Number of cache misses in column first  traversal = 32.
 Miss rate = 3.1%.
Number of cache misses in row first  traversal = 32*32 = 1024.
Miss rate = 100%
```

(c) Assume a cache that has n levels. For each level, the hit time is x cycles, and the local miss rate is y per cycle. Try to formulate a recursive formula to calculate the average memory access time?

**Solution:** T(n) = x + y*T(n-1)

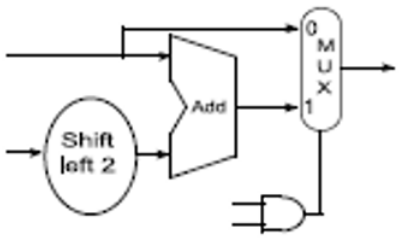
2. **(2+5+3 = 10 marks) (*Data Path and Addressing Mode*)**:

(a) Estimate the number of memory accesses required in each of the following instructions.
i. ADD $t1, $t2, $t3
ii. ADD $t1, $t2, 0($t3)

**Solution:**

```
i. 1 : To fetch the instruction
ii. 2 : First to fetch the instruction and then to fetch the contents
of location pointed by $t3
```

(b) The hardware shown below was inserted when the MIPS 32 datapath was extended for a specific instruction. Identify the instruction.



**Solution:**

```
This hardware is used to compute the branchtarget address,
and determine which address to pass back to the PC.
That's relevant only to beq.
```

(c) What is wrong with the following register transfer statements?

i. XT : AR $\leftarrow$ (AR)' , AR $\leftarrow$ 0

ii. $YT : R1 \leftarrow R2, R1 \leftarrow R3$

iii. $ZT : PC \leftarrow AR, PC \leftarrow PC + 1$

**Solution:**
i. Cannot complement and clear the same register at the same time.
ii. Cannot transfer two different values (R2 and R3) to the same register (R1) at the same time.
iii. Cannot transfer a new value into a register (PC) and increment the original value by one at the same time.

3. **(2+3+5 = 10 marks) (*Assembly Language programming : MIPS*)**

(a) Match the region of a running program's memory with what to store there. Some regions may have zero, one, or multiple answers. Justify your choice.
**What to store:**
a) an array of characters representing a message that might be printed
b) a constant
c) return address of a procedure call
d) the instructions for the program
e) a binary tree that may have elements inserted while the program is running
f) values for registers that need to be preserved across a function call

**Regions:** *stack :— ; heap : —– ; .data : ——; .text : ——-*

```
stack : c,f ; heap : e ;
.data : a,b; .text : d
```

(b) The following MIPS code tries to reverse the contents of array A of words. The base address starts in register $a0 and the length of the array starts in register a1.

```
 reverse:
add $t0, $zero, $a0 # t0=a0 points to start of A
addi $t1, $a1, -1 # t1 = a1-1
sll$t1,$t1,____ # t1=t1x4
add $t1, $a0, $t1 # t1 points to last element of A
loop:
lw $t2,0($t0) # t2 gets 1st element of array A
lw $t3,0($t1) # t3 gets last element of array A
sw ___, 0($t1) # store ___ into end of A
sw ___, 0($t0) # store ___ into start of A
addi $t1, $t1, ___ # Update $t1
addi $t0, $t0, __ # Update $t0
bgt $t1, $t0, loop # Continue until $t0>=$t1
```

Fill in the blanks (there are five) to make the program work.

**Solution:**

```
 reverse:
add $t0, $zero, $a0 # t0=a0 points to start of A
addi $t1, $a1, -1 # t1 = a1-1
sll$t1,$t1,__2__ # t1=t1x4
add $t1, $a0, $t1 # t1 points to last element of A


loop:
lw $t2,0($t0) # t2 gets 1st element of array A
lw $t3,0($t1) # t3 gets last element of array A
sw _$t2__, 0($t1) # store ___ into end of A
sw $t3___, 0($t0) # store ___ into start of A
addi $t1, $t1, -4___ # Update $t1
addi $t0, $t0, _-4_ # Update $t0
bgt $t1, $t0, loop # Continue until $t0>=$t1
```

(c) Each card in a 52-card deck (4 suits and 13 values A,2,3,...,K) can be represented by a 6-bit binary number. Below is how a card is stored in a 32-bit integer

```
26 unused bits| _fvalue  _ _ _ | _suit _
```

Suppose we want to write a function check_fvalue. The prototype of the function as it would appear in C is: **int check_fvalue(int cardA, int cardB)** *CardA* and *CardB* are integers such that the 6-bit representation of the card is in the least significant bits and the rest of the bits are unknown. The function returns 0 if the cards had different fvalues and 1 if the cards had the same fvalue. Write check_fvalue as a function in MIPS. Use the conventions for functions written in MIPS.

**Solution:**

```
check_fvalue:
srl $t0, $a0, 2
srl $t1, $a1, 2
beq $t0,$t1,same
addiu $v0, $zero, 0
jr $ra
same:
addiu $v0, $zero, 1
jr $ra
```