EN605.204 – Computer Organization

Module 3 Assignment – MIPS Assembly Language

Question #1: Assembling MIPS Instructions

Assemble (MIPS -> binary/hex) the following instructions into their binary and hex equivalents (please show your work):

1. add $t4, $t0, $t1
2. or $s0, $s2, $s3
3. andi $t3, $s0, 0x7123
4. jr $ra
5. j 0x1234

Question #2: Disassembling MIPS Instructions

Disassemble (binary/hex -> MIPS) the following instructions from binary/hex into their MIPS equivalent (please show your work)::

1. 00000010100101101000100000100010
2. 0x01694024
3. 00110101101101100001001000110100
4. 0x2288FACE

Question#3: Endianness

Assemble the following instructions from MIPS to hex and give the hex representation in both big- and little-endian format:

1. `or $s4, $t2, $t7`
2. `andi $t6, $s0, 0x4040`

or (little)				
Memory Address	0x0	0x1	0x2	0x3

or (big)				
Memory Address	0x0	0x1	0x2	0x3

andi (little)				
Memory Address	0x3	0x2	0x1	0x0

andi (big)				
Memory Address	0x3	0x2	0x1	0x0

Question 4: Registers

Please answer each of the following questions and justify your answer when needed.

1. To which value is the $zero register hard-coded?
2. Which register would you use to return a value from a function call?
3. Which two registers are used for storing data on the call stack?
4. What is the difference between ‘s’ and ‘t’ registers?
5. Should the $at register be used by developers? Why/why not?

$fp, If the callee uses a frame pointer, then it usually sets it to the the stack pointer. The old frame pointer
e saved on the stack before this happens.

restores stack pointer back to the way it found it just as it was being called. Thus, stack pointer now points
eturn value.

t variables are temporary caller saved registers, while the $s registers are callee saved.

s Reserved for pseudo-instructions