|  |  |  |  |
| --- | --- | --- | --- |
| Instructor |  | Due Date |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Part | **1** | **2** | **3** | **4** | Total |
| *Maximum Points* | **25** points | **25** points | **25** points | **25** points | **100**G101010 pointsG |
| ***Your Score*** |  |  |  |  |  |

**Textbook Reading Assignment**

Thoroughly read Chapter(s) 5 in your Computer Architecture and Organization textbook.

**Part 1 Glossary Terms - Instruction Set Architectures ( ISAs )**

Define, in detail, each of these glossary terms from the realm of computer architecture and computer topics, in general. If applicable, use examples to support your definitions. Consult your notes

or course textbook(s) as references or the Internet by visiting Web sites such as:

[**http://www.bing.com**](http://www.bing.com) or [**http://www.webopedia.com**](http://www.webopedia.com/)

**(a) endian**

|  |
| --- |
| “Endian” refers to an architecture’s byte order, in which the most significant bytes are stored either at lower addresses (little endian) or higher addresses. Most UNIX machines (and computer networks) are big endian, most PCs are little endian.   E.g., on a little endian machine, and integer with 4 bytes (bytes 3,2,1 and 0) would be arranged in memory as 0-1-2-3, and on a big endian machine, it would be arranged as 3-2-1-0. |

**(b) infix Notation**

|  |
| --- |
| Infix notation is the standard representation of operands and their values (X+Y). |

**(c) instruction - level pipelining**

|  |
| --- |
| Instruction level pipelining is used to implement instruction level parallelism, in which, like an assembly line on a factory, each pulse of a clock can be execute multiple instructions in the fetch-decode-execute cycle. This eliminates bottlenecks and makes for much faster processing.  For example, an instruction can be fetched on the first clock pulse; on the second, another instruction can be fetched while the first is being decoded; on the third, yet another instruction can be fetched while the second is decoded and the first is executed, etc. |

**(d) instruction set**

|  |
| --- |
| And instruction set is the set of Assembly language instructions that a machine can execute. These instructions contain both an operand and an address for where the instruction is located.   Some types of instructions include those for moving data (MOVE, LOAD, STORE, POP, PUSH, etc.), arithmetic (ADD, SUBTRACT, INCREMENT), Boolean logic (AND, NOT, OR, XOR, etc.), bit manipulation (SHIFT, ROTATE), I/0 instructions (which vary greatly between manufacturers), transfer of control (for branching, skips, procedure calls, returns and termination) and special purpose instructions (which vary but often contain instructions for high-level language support and string processing). |

**(e) postfix Notation**

|  |
| --- |
| Postfix notation is a representation of operands and their values in which the operands follow their values, and in architecture is commonly implemented as a stack of registers to evaluate arithmetic expressions. |

**Part 2 Exercises - Instruction Set Architectures ( ISAs )**

For each of the following, enter True or False.

**TRUE (1)** Most computers typically fall into one of three types of CPU organization:

(1) general register organization; (2) single accumulator organization; or (3) stack organization.

**FALSE (2)** The advantage of zero - address instruction computers is that they have short programs; the disadvantage is that the instructions require many bits, making them very long.

**TRUE (3)** An instruction takes less time to execute on a processor using an instruction pipeline than on a processor without an instruction pipeline.

**TRUE (4)** The term " endian " refers to an architecture’s byte ordering.

**TRUE (5)** Stack architectures have good code density and a simple model for evaluation of expressions, but do not allow random access, which can cause a problem with the generation of efficient code.

**FALSE (6)** Most architectures today are accumulator - based.

**TRUE (7)** Fixed - length instruction format typically results in better performance than variable length instruction format.

**FALSE (8)** Expanding opcodes make instruction decoding much easier than when it is not used.

**FALSE (9)** Instruction set orthogonality refers to the characteristic in an instruction set architecture where each instruction has a " backup " instruction that performs the same operation.

**TRUE (10)** The effective address of an operand is the value of its actual address in memory.

**Part 3 Exercises - Reverse Polish Notation ( RPN )**

Convert each of the following and use an online calculator, such as that shown below, to check your answers.

[**https://www.mathblog.dk/tools/infix-postfix-converter/**](https://www.mathblog.dk/tools/infix-postfix-converter/)

**(1)** Convert the following expression from infix to Reverse Polish ( postfix ) Notation.

( 8 − 6 ) / 2   
 **8 6 – 2 /**

**(2)** Convert the following expression from infix to Reverse Polish ( postfix ) Notation.

( 2 + 3 ) × 8 / 10

**2 3 + 8 x 10 /**

**(3)** Convert the following expression from infix to Reverse Polish ( postfix ) Notation.

5 × ( 4 + 3 ) × 2 − 6  
 **5 4 3 + x 2 x 6 -**

**(4)** Convert the following expressions from infix to Reverse Polish ( postfix ) Notation.

X × Y + W × Z + V × U   
 **X × Y W × Z + V × U +**

**(5)** Convert the following expressions from Reverse Polish Notation to infix notation.

W X Y Z − + ×   
 **W \* (X + Y - Z)**

**Part 4 Exercises - Instruction Set Architectures ( ISAs )**

Write a complete answer for each of these. You can use an online converter such as that available at the following Web address.

[**http://scanftree.com/Data\_Structure/prefix-postfix-infix-online-converter**](http://scanftree.com/Data_Structure/prefix-postfix-infix-online-converter)

**(1)** Is the following a valid postfix operation? If so, explain how a stack is used to evaluate the RPN expression or explain why it is invalid.

1. 8 3 1 + − /

|  |
| --- |
| **This is a valid postfix operation and evaluates to the infix representation:  12 / (8 - (3 + 1))  The postfix expression is read from left to right, and if an operand (number) is encountered, it is pushed onto the stack. If the element is an operator, pop operands from the stack and evaluate, then push the result back onto the stack. When the expression is ended, the final number is the result of the postfix operation.** |

**(2)** Determine the arithmetic statement for which the following program was written to evaluate the statement using a stack organized computer with zero - address instructions ( so only pop and push can access memory ) .

Push A

Push B

Subtract

Push C

Push D

Push E

Mult

Push F

Subtract

Mult

Add

Push G

Push H

Push K

Mult

Add

Div

Pop X

|  |
| --- |
| **Not sure what to write here! Should this be infix / postfix arithmetic expression?** |

**(3)** What is the difference between using direct and indirect addressing? Give an example.

|  |
| --- |
| **Direct addressing is an instruction format in which the address following an opcode is the address of the instruction, while in indirect addressing the instruction’s address contains a pointer to a value which is the “real” address of the instruction.   E.g., direct addressing: if I have an instruction with an address of 008, the value found at that address is operated on. For indirect addressing, the value at address 008 is the “real” address of the value to be operated on, and that gets operated on instead.** |