**Note:**  (Late: -15% Penalty).

1. (5 pts) Fill in the following table to show how the given integers are represented, assuming 16-bits are used to store values and the machine uses 2’s complement notation.

|  |  |  |  |
| --- | --- | --- | --- |
| Integer | Binary | Hex | 4 Byte Little Endian  (hex value as seen in memory) |
| 28 | 0000000000011100 | 001C | 1C00 |
| 2216 | 0000100010101000 | 08A8 | A808 |
| -18675 | 1011011100001101 | B70D | 0DB7 |
| -12 | 1111111111110100 | FFF4 | F4FF |
| 31456 | 0111101011100000 | 7AE0 | E07A |

1. (5 pts) Convert the following expressions from infix to reverse Polish (postfix) notation.
   1. (8–6)/2

Answer: a) 86–2/

* 1. (2+3)\*8/10

Answer: b) 23+8\*10/

* 1. (5×(4+3)×2–6)

Answer: c) 5 4 3 + × 2 × 6 –

1. (5 pts) Explain how a stack is used to evaluate the RPN (reverse polish notation) expression 3 5 7 + 2 1 - \* 1 + +

push 3, push 5 and push 7 on the stack.

The plus operator pops 7, pops 5, adds 5 + 7, then pushes 12 onto the stack.

Then push 2 and push 1 on the stack.

The minus operator pops 1, pops 2, subtracts 1 from 2, then pushes 1 onto the stack.

The times operator pops 1, pops 12, multiplies the operands and pushes 12 onto the stack.

The 1 is pushed, then the plus operator pops 1, pops 12 adds 12 plus 1, and pushes 13 onto the stack.

The plus operator pops 13, pops 3 adds 3 + 13, and pushes 16 onto the stack.

|  |
| --- |
| 7 |
| 5 |
| 3 |

|  |
| --- |
|  |
| 12 |
| 3 |

|  |
| --- |
| 1 |
| 2 |
| 12 |
| 3 |

|  |
| --- |
|  |
| 1 |
| 12 |
| 3 |

|  |
| --- |
|  |
|  |
| 13 |
| 3 |

|  |
| --- |
|  |
|  |
|  |
| 16 |

1. (5 pts) Define:
   1. Immediate addressing

**Immediate addressing** is an **addressing** form in which the byte value to be used or retrieved in the instruction, is located immediately after the opcode for the instruction itself.

* 1. Direct addressing

Direct addressing mode means that the value for a given instruction in assembly programming is pointed to by a given value. This means the value is variable, based on what is stored in memory at a given address

* 1. Indirect addressing

Indirect addressing uses an address held in a register or other location to determine what memory location to read or write. The idea here is that the instruction itself isn’t directly telling you the address to access, but rather indirectly telling the CPU where to find that address. The processor may also allow you to add a small offset to the indirect address, giving an indirect-indexed addressing mode

* 1. Indexed addressing

Indexed addressing means that the final address for the data is determined by adding an offset to a base address. Very often, a chunk of data is stored as a complete block in memory

1. (5 pts) A nonpipelined system takes 200ns to process a task. The same task can be processed in a 5-segment pipeline with a clock cycle of 40ns. Determine the speedup ratio of the pipeline for 200 tasks. What is the maximum speedup that could be achieved with the pipeline unit over the nonpipelined unit?

SpeedUp = (200ns x 200)/((5+200-1)(40ns)) = 40000/8160 = 4.9019 Max SpeedUp = 5

1. (5 pts) Suppose we have the instruction “Load1000”. Given memory and register R1contain the values below, and assuming R1 is implied in the indexed addressing mode, determine the actual value loaded into the accumulator and fill in the table below:

|  |  |
| --- | --- |
| Memory | |
| Address | Data |
| 0x1000 | 0x1400 |
|  |  |
| 0x1100 | 0x400 |
|  |  |
| 0x1200 | 0x1000 |
|  |  |
| 0x1300 | 0x1100 |
|  |  |
| 0x1400 | 0x1300 |

|  |
| --- |
| R1 |
| 0x200 |

|  |  |
| --- | --- |
| Mode | Value Loaded into AC |
| Immediate | 0x1000 |
| Direct | 0x1400 |
| Indirect | 0x1300 |
| Indexed | 0x1000 |

1. Assuming the same stages as in Example 5.11, explain the potential pipeline hazards (if any) in each of the following code segments.
   1. X=R2+Y;  R4= R2+X

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time  Period | 1 | 2 | 3 | 4 | 5 | 6 |
| X=R2+Y | Fetch Ins | Decode | Fetch Y | Add & store in X |  |  |
| R4=R2+X |  | Fetch Ins | Decode | Fetch X |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Answer:

The problem is a resource conflict at time 4, as both instructions need to access memory.

* 1. R1=R2+X; X =R3+Y; Z =R1+X

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time  Period | 1 | 2 | 3 | 4 | 5 | 6 |
| R1=R2+X | Fetch Ins | Decode | Fetch X | Add & store in R1 |  |  |
| X = R3+Y |  | Fetch Ins | Decode | Fetch Y | Add & store in X |  |
| Z=R1+X |  |  | Fetch Ins | Decode | Fetch X |  |
|  |  |  |  |  |  |  |

The problem is a resource conflict at time 3 and a data dependency at time 5.

1. (15 pts) Write working assembly code that successfully executes the following stack reverse Polish (postfix) based code.
   1. Your code must include the use of sub-routines Push, Pop, Subtract, Add

/Push A

Load A

Store Temp /Stores A in a memory location from which Push will retrieve it

JnS Push /Jumps to Push subroutine and stores return address

/Push B

Load B

Store Temp

JnS Push

/Push C

Load C

Store Temp

JnS Push

/Add

JnS Add

/ Push D

Load D

Store Temp

JnS Push

/ Push F

Load F

Store Temp

JnS Push

/Subtract

/Subtract

JnS Subtract

JnS Subtract

/Push E

Load E

Store Temp

JnS Push

/Add

/Add

JnS Add

JnS Add

/Pop X

JnS Pop

Halt /Halts the execution of the program

Push, Hex 0 /Return address will be stored here

Load Pointer

Add One /Increments Pointer by 1

Store Pointer

Load Temp /Loads value to be pushed into the stack

StoreI Pointer /Stores loaded value using the address indicated by Pointer

JumpI Push /Returns from the Push subroutine

Pop, Hex 0

LoadI Pointer /Loads a value from the address indicated by Pointer

Store Temp

Load Pointer

Subt One /Decrements the Pointer

Store Pointer

Load Temp

JumpI Pop

Add, Hex 0 /Add subroutine start

JnS Pop

Store Op /Stores the operand

JnS Pop

Add Op /Performs summation

Store Temp

JnS Push /Pushes the sum into the stack

JumpI Add

Subtract, Hex 0 /Subtract subroutine start

JnS Pop

Store Op /Stores the operand

JnS Pop

Subt Op /Performs subtraction

Store Temp

JnS Push /Pushes the result into the stack

JumpI Subtract

One, Dec 1

Pointer, Hex 10A

Temp, Dec 0

Op, Dec 0

A, Dec 3

B, Dec 5

C, Dec 7

D, Dec 2

E, Dec 4

F, Dec 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* 1. What is the final value for X? Answer: 18 : 3+(((5+7)-(2-1))+4) = 18