1. Consider a pipeline that has five stages: (1) instruction fetch – 30 ns, (2) instruction decode – 10 ns, (3) memory read – 30 ns, (4) instruction execution – 10 ns, and (5) register write – 30 ns. The times required in each stage are shown above following the name of the stage. For example, the instruction fetch requires 30 ns to complete. Assume that every instruction in the instruction set requires the use of all stages of the pipeline. Also, assume that the stages of the pipeline are clocked with a common clock. This means that all information moves synchronously through the pipeline from one stage to the next based on a single clock signal dictating when information advances from one stage to the next.

a. What is the latency of an instruction flowing through the pipeline? Again, remember that you have a common clock.

\[ 30 \text{ ns} \times 5 \text{ stages} = 150 \text{ ns} \]

\[ 1000 \times (40 + 40 + 30) = 1000 \times 110 = 110000 \text{ ns} \]

b. What is the idealized throughput (bandwidth) of the pipeline? (In other words, you may assume that there are no hazards between instructions for calculating this answer) \(1000 \text{ insts} \times \frac{1}{5} \text{ instructions/cycle} = 200 \text{ insts/cycle}\)

\[ 1000 \times 150 = 150000 \text{ insts} \times \frac{1}{5} = 30000 \text{ insts/cycle} \]

c. What is the execution time of 1000 inst program without pipelining?

d. Since all stages of the pipeline are clocked with a single clock signal, what is the fastest frequency of that clock for this pipeline?

\[ 1 \text{ cycle} / 30 \text{ ns} \]

d. Assume that the following sequence of instructions is executed by the processor (assuming no hazards):

\[ I_1, I_2, I_3, I_4 \]

Using the table below, show the instruction that resides in each stage of the pipeline for each clock period.

<table>
<thead>
<tr>
<th>Clock Period</th>
<th>Instruction Fetch</th>
<th>Instruction Decode</th>
<th>Memory Read</th>
<th>Instruction Execution</th>
<th>Memory Write Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>I_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>I_2</td>
<td>I_1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>I_3</td>
<td>I_2</td>
<td>I_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>I_4</td>
<td>I_3</td>
<td>I_2</td>
<td>I_1</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>I_5</td>
<td>I_4</td>
<td>I_3</td>
<td>I_2</td>
<td>I_1</td>
</tr>
<tr>
<td>T6</td>
<td>T_6</td>
<td>I_4</td>
<td>I_3</td>
<td>I_2</td>
<td>I_3</td>
</tr>
<tr>
<td>T7</td>
<td>T_5</td>
<td>T_6</td>
<td>I_4</td>
<td>I_3</td>
<td>I_3</td>
</tr>
<tr>
<td>T8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A central processing unit (CPU) has the following hardware components:

- 2 general-purpose registers (R0-R1)
- A Program Counter (PC)
- An Instruction register (IR)
- An Arithmetic and logic unit (ALU) that can perform 8 logical and 8 arithmetic functions
- A Memory Address Register (MAR)
- A Memory Data Register (MDR)
- An A register to temporarily hold one operand for the ALU
- A C register to temporarily hold the result from the ALU

A single, internal CPU bus is to be used to interconnect the hardware resources.

a. Draw a block diagram of the data path for this CPU.

b. List all of the control signals that are needed in your data path.

Answer: \( \text{RegFileIn, RegFileOut, PC}_{\text{in}}, \text{PC}_{\text{out}}, \text{C}_{\text{in}}, \text{C}_{\text{out}}, \text{A}_{\text{out}}, \text{A}_{\text{in}}, \text{IR}_{\text{in}}, \text{ALU}_{\text{op}}, \text{MAR}_{\text{in}}, \text{MDR}_{\text{out}}, \text{MDR}_{\text{in}} \)
3. Consider the following simple SRC program:

```plaintext
lar r5, Over
brl r6, r5
andi r8, r8, 0
```

A delay slot means that the next instruction will be executed before the branch takes effect?

Over: add r10, r8, r9

4. Assume that this program is running on a pipelined SRC machine that has a single branch-delay slot due to the pipeline. Also, assume that the following initial register contents exist (H means Hexadecimal):

```
R[r8] = 15282343 H
R[r9] = 22410006 H
```

What is the content of register r10 after this program completes the execution of the add instruction? Show your answer in hexadecimal.

Answer: 22410006 Hex

b. If the SRC did not have a branch-delay slot what would be the content of register r10 after the program completed the execution of the add instruction? Show your answer in hexadecimal.

Answer: 37692349 Hex

4. 1) Describe the following types of data hazards, 2) give a short SRC example for each, and 3) explain some techniques for handling these types of hazards

a. RAW

Read after Write: when an instruction contains a read of a register after an instruction which writes to the register but potentially before the register value has been written.

Example:
```
add r0, r1, r2
add r3, r0, r5
```

Techniques:
- Stalling
- Forwarding/bypassing
- Code reordering

b. WAR

Write after Read: This type of hazard can occur when instructions can be executed in parallel or out-of-order.

Example:
```
add r2, r1, r0
sub r0, r4, r5
```

Techniques:
- Register-renaming
- Store result of one in a temporary register until first inst is executed
c. WAW

Write after Write: Write of a register after a write to the same register (order matters)

Techniques: register-renaming

store result of os in temp register until first is executed

5. An exception interrupts the normal sequence of instruction execution. Exceptions can be classified as internal and external.

a. There are many types of exceptions. Describe and explain four different types of exceptions.

p. 179 of textbook

b. The processor must have a process for handling exceptions. Describe 4 things that the processor must be able to identify and/or take care of.

1) Control exception handling
2) Identify interrupting device
3) Save processor state
4) Disable exceptions during critical operations
5) Invoke exception handler (find)
6. Pipelining is one means to increase a processor's instruction execution rate by subdividing the instruction fetch and execution process into smaller subtasks that can be parallelized. Multiple issue is another technique by which several instructions are issued (sent) to the same pipeline stage at once. In order to be able to issue more than one instruction to the same pipeline stage at once, there must not be any dependences (data or control) between them. The textbook describes two ways in which multi-issue can be implemented: VLIW (static multi-issue, Very Long Instruction Word) and superscalar (dynamic multi-issue).

Describe static multi-issue (VLIW) and dynamic multi-issue (superscalar) in terms of how and when decisions about which instructions can be issued into the pipeline are made.

**Static multi-issue - VLIW**

- Compiler makes decisions about instructions which can be scheduled at compile time

**Dynamic multi-issue - superscalar**

- Made at runtime, requires additional hardware
- Instructions are fetched into an instruction buffer and are analyzed for dependencies

<table>
<thead>
<tr>
<th>Check</th>
<th>Instruction Fetch</th>
<th>Instruction Decode</th>
<th>Instruction Execute</th>
<th>Store</th>
<th>Write Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T7</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>