1. [10 Points] Calling Conventions.

   (a) What are calling conventions?

   (b) Why are they necessary?

   (c) Give an example of a MIPS calling convention, and say why it is useful.
2. [30 points] **MIPS Programming**

The following C code does a binary search for a value in a sorted array A of integers (each integer is 32 bits in size, and they are packed together in the array). Translate this C code into MIPS assembly using the template on the next page. *Hint*: you may find the BGTZ/BLTZ instructions useful.

Your solution will not be graded for syntax, but you must use the proper opcode and register names. You should make use of the following assumptions:

- $a0 contains A
- $a1 contains lengthOfA
- $a2 contains value
- $v0 should be used to hold the return value
- your function will be called by some other function.
- you are allowed to use pseudo-instructions.

**C version**

```c
int binSearch(int *A, int lengthOfA, int value) {
    int low = 0;
    int high = length - 1;
    int mid = 0;

    while (low <= high) {
        mid = (low + high) / 2;
        if (A[mid] > value)
            high = mid - 1;
        else if (A[mid] < value)
            low = mid + 1;
        else
            return mid; /* found, return array index */
    }
    return -1; /* not found */
}
```

**MIPS assembly version (next page)**
MIPS assembly version

binSearch:
    li $t0 <- 0  # low
    addi $t1 <- $a0, -1  # high
    li $t2 <- 0  # mid
3. [15 Points] **Datapath.**

Taking the single-cycle processor developed in class, we want to add a new instruction `jrmi imm16(rs)` which executes a jump to the instruction at the specified address in memory. `jrmi` is an I-type instruction. The 16-bit immediate (a word offset) and the register `rs` specify an address via register base + offset (in the same manner as `lw/sw`).

(a) Draw the necessary modifications to implement the above instruction on the figure of the single-cycle data-path provided below.

(b) What is/are the new control signal(s) required to implement `jrmi`?
4. [20 Points] **Datapath Control Signals.**

Given the single-cycle datapath above (control signals are marked with dashed lines), fill in the blanks in the table below. You must either give the control signals (0, 1, X) for a particular MIPS instruction, or give the MIPS instruction that uses the specified control signals.

Each control signal must be specified as 0, 1 or X (don’t care). Writing a 0 or 1 when an X is more accurate is not correct.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>RegDst</th>
<th>RegWrite</th>
<th>ALUSrc</th>
<th>ALUOp</th>
<th>MemWrite</th>
<th>MemRead</th>
<th>MemToReg</th>
<th>PCSrc</th>
</tr>
</thead>
<tbody>
<tr>
<td>nor</td>
<td></td>
<td></td>
<td>nor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td></td>
<td></td>
<td>sub</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>∧ ALU.Zero</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
5. [15 Points] **Pipeline Hazards.**

Consider the sequence of MIPS instructions below:

\[
\begin{align*}
\text{add} & \quad \$2 \leftarrow \$3, \$4 \\
\text{or} & \quad \$5 \leftarrow \$2, \$4 \\
\text{lw} & \quad \$6 \leftarrow 0(\$4) \\
\text{addi} & \quad \$7 \leftarrow \$6, 0x5 \\
\text{sub} & \quad \$8 \leftarrow \$8, \$4
\end{align*}
\]

(a) Draw arrows on the instructions above indicating all the data dependences.

(b) Reorder the instructions into a new schedule that will execute without any stalls on a 5-stage pipelined processor with forwarding. For reference, you may refer to the pipeline diagram below.

(c) Reorder the instructions into a new schedule that will execute without any stalls on a 5-stage pipelined processor *without* forwarding. For reference, you may refer to the pipeline diagram below.
6. [10 Points] **Pipelining.**

Suppose you have a program that is 1 instruction long. Will it execute faster on a pipelined processor than on a single-cycle processor (assuming equal processor frequencies)? Why or why not?