1 Fundamental CPU architecture

(a) List typical CPU components and describe the steps that need to be taken from these components to execute the following classes of instruction. Extra credits are given if you could mention the corresponding effects in terms of pipeline stages, I-Cache, D-Cache and execution units.

   (i) arithmetic operations
   (ii) load instruction
   (iii) store instruction
   (iv) conditional branch
   (v) push instruction
   (vi) pop instruction

(b) If a processor executes a jump to subroutine, how are state and control passed between the function and the caller? You could list one example from the ISA you are familiar with, from the following ISAs (x86, ARM, MIPS, PowerPC or RISC-V). Indicate what state is passed, but details of particular register numbers are not required.

(c) The flow control of a processor could be changed using branch or jump instructions. Besides these instructions, interrupts and exceptions could also change the PC and the content of instruction fetch of the CPU. State why interrupts and exceptions are needed. What are the differences between them. What are the software and hardware approach towards handling the interrupts and exceptions?

(d) Given that the MIPS processor has a branch delay slot, what will the following contrived piece of code do?

   ```
   foo:   slti $t0, $t0, 5
          beq $t0, $zero, foo
          addi $t0, $t0, -1
   ```

2 RISC-V Architecture

(a) OpenRISC or MIPS are existing open source ISAs. Why people introduced another open source RISC-V ISA?

(b) Why RISC-V specifies its register x0 always to be zero? How do you move the value from one general register to another?
(c) What technique was introduced in the RISC-V to increase the code density? How does it work?

(d) What instruction sequence might be used to put a 32-bit constant into a register without using a load instruction?

(e) If register \( x1 \) stores an integer, what instruction sequence might be used to perform a multiplication of 17 and store it back?

(f) For the following C program, what should it print. You could write a sample test program, run it on a 64-bit machine. Explain why it prints the value.

```c
#include <stdio.h>
#include <stdint.h>

void main() {
    uint64_t hexspeak = 0xDEADBEEF8BADC0DE;
    uint64_t addr = (uint64_t)(&hexspeak);
    uint8_t value = *(uint8_t *)(addr+3);
    printf("value = %x\n",value);
}
```

3 Bubble Sort Assembly

Consider the following code which takes an array \( d \) of n 32-bit integers and performs a bubble sort.

```c
for(i=0; i<n; i++) {
    int m = n-i-1;
    for(j=0; j<m; j++)
        if(d[j]>d[j+1]) {
            t = d[j];
            d[j] = d[j+1];
            d[j+1] = t;
        }
}
```

(a) Given the following register allocation, how would the inner loop be encoded in assembler for a RISC-V processor?

<table>
<thead>
<tr>
<th>register</th>
<th>variable</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x5</td>
<td>d</td>
<td>base address of array d</td>
</tr>
<tr>
<td>x6</td>
<td>m</td>
<td>value of m</td>
</tr>
<tr>
<td>x7</td>
<td>j</td>
<td>register used to hold j</td>
</tr>
<tr>
<td>x8</td>
<td>t</td>
<td>register to hold temporary t</td>
</tr>
</tbody>
</table>

(b) The classic 32-bit ARM instruction set makes every instruction conditional. What had to be sacrificed in order to make space in the instruction format for the condition code bits? How can your code make use of conditional instructions to reduce the number of data-dependent branches? Why don’t RISC-V don’t have conditional codes?

(c) Given the classic 5-stage pipeline (below), how do the control and data hazards differ between your code in parts (a) and (b)? What is the impact on performance?
(d) Give two examples of the different addressing modes that are often supported by a x86 processor. For each addressing mode, show how it would reduce the instruction counts from your code (a).

End of Problems