CS365 Final Examination
Section 002, Fall 2005

Print Your Name: __________________________

GMU ID: __________________________

☐ Print your initials on the upper right corner of every page.

☐ You have 120 minutes to earn up to 360 points

STOP! Do NOT turn to the next page until further instructions.
**Question 1:** (10pt) Give the -10 in 5-bit 2’s complement

Answer: 10110

**Question 2:** (30pt) Execute the following multiplication by shift and add

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>Initial Product</th>
<th>Product after 1st shift</th>
<th>Product after 2nd shift</th>
<th>Product after 3rd shift</th>
<th>Product after 4th shift</th>
<th>Product after 5th shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 1</td>
<td>0 0 0 0 0 0 1 0 1 1</td>
<td>0 1 0 0 1 0 1 0 1</td>
<td>0 1 1 0 1 1 0 1 0</td>
<td>0 0 1 1 0 1 1 0 1</td>
<td>0 0 1 1 0 0 0 1 1</td>
<td>0 0 0 1 1 0 0 0 1 1</td>
</tr>
</tbody>
</table>
Question 3: (30pt total) Given the 32-bit word below, answer the following questions.

3.A (15pt) Interpret it as a MIPS machine instruction, show its assembly counterpart.

3.B (15pt) Interpret it as a 2’s complement number, give an mathematic expression that calculates its value in decimal notation.

Answer to 3.A: lw $a0, 9($t0)
Note: in MIPS assembly, the destination register always goes first

Answer to 3.B: $-2^{31} + 2^{27} + 2^{26} + 2^{24} + 2^{18} + 2^3 + 2^0$
**Question 4**: (50pt total) Consider a byte-addressable architecture that supports 36 bit addresses $A_0$ to $A_{35}$. Its cache is 32 way associative and has 64 sets. A block contains 256 bytes. Answer the following questions.

4.A (10pt) Give the address bits that give the block number.

A35 to A8

4.B (10pt) Calculate the total capacity of the cache in bytes

$32 \times 64 \times 256$

64 sets, each set containing 32 block, each block 256 bytes

4.C (15pt) Give the address bits stored in a cache TAG.

A35 to A14
First 6 bits in a block # (A13 to A8) used to determine the set #
Remaining bits stored in TAG

4.D (15pt) Give the address bits used to determine the set to store a block.

A13 to A8
Question 5: (40pt) Consider a 4-way set associative cache with two sets. The size of a block is 4, and the replacement policy is LRU. Starting from the cache empty, give the contents of the cache after the following sequence of memory references:

<table>
<thead>
<tr>
<th>Addr</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>0</td>
</tr>
<tr>
<td>001010</td>
<td>1</td>
</tr>
<tr>
<td>010011</td>
<td>2</td>
</tr>
<tr>
<td>000100</td>
<td>3</td>
</tr>
<tr>
<td>010011</td>
<td>4</td>
</tr>
<tr>
<td>011001</td>
<td>5</td>
</tr>
<tr>
<td>100010</td>
<td>6</td>
</tr>
<tr>
<td>111001</td>
<td>7</td>
</tr>
</tbody>
</table>

Set 0:

<table>
<thead>
<tr>
<th>Addr</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>1</td>
</tr>
<tr>
<td>111000</td>
<td>1</td>
</tr>
<tr>
<td>010000</td>
<td>1</td>
</tr>
<tr>
<td>011000</td>
<td>1</td>
</tr>
</tbody>
</table>

Set 1:

<table>
<thead>
<tr>
<th>Addr</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>1</td>
</tr>
</tbody>
</table>

Counts:

<table>
<thead>
<tr>
<th>Addr</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>6</td>
</tr>
<tr>
<td>001010</td>
<td>7</td>
</tr>
<tr>
<td>010011</td>
<td>5</td>
</tr>
<tr>
<td>000100</td>
<td>4</td>
</tr>
<tr>
<td>010011</td>
<td>3</td>
</tr>
<tr>
<td>011001</td>
<td>3</td>
</tr>
<tr>
<td>100010</td>
<td>6</td>
</tr>
<tr>
<td>111001</td>
<td>7</td>
</tr>
</tbody>
</table>
**Question 6.** We add new instruction AddN to the YH16
OPCODE: 111111
Format: II
Actions:
   rt = rt + rs
   if rs == 1, PC = PC + Immd
   rs = rs – 1

6.A (50pt) Give the cycle by cycle execution of AddN

Answer:
Cycle 0:
   IR = Mem[PC]
   C = PC + 1
Cycle 1:
   PC = C
   A = Reg[rs]
   B = Reg[rt]
   MDR = Mem[C]
Cycle 2:
   C = PC + 1
Cycle 3:
   PC = C
   C = PC + MDR
Cycle 4:
   Z = (A == 1)
Cycle 5:
   PC = C, if Z
   C = A – 1
Cycle 6:
   Reg[rs] = C
   C = A + B
Cycle 7:
   Reg[rt] = C
6.B (30pt) Identify the cycle where you calculate rs-1. Give the control signals for that cycle. (Cycle 5 in my answer)

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Question 7.** (30pt) Give the Boolean expression of control UseR0 in the YH16 architecture.

\[
\text{UseR0} = S_2' S_1' S_0 (I_{15} I_{14} I_{13} I_{12} I_{11} I_{10} I_9' + I_{15} I_{14} I_{13} I_{12} I_{11} I_{10} I_9)
\]

(state 1 of BF and BT)
**Question 8:** (40pt) Assuming the standard MIPS ISA, identify instances of internal forwarding, including ALU and memory forwarding. In the boxes below, fill in a register number in box\[i,j]\ if instruction \(i\) forward the value of the given register to instruction \(j\). If there is no forwarding, leave the box blank.

1. \(\text{sub} \ r12, r4, r3\)
2. \(\text{lw} \ r5, 0(r12)\)
3. \(\text{add} \ r10, r12, r1\)
4. \(\text{xor} \ r6, r10, r5\)
5. \(\text{slt} \ r20, r5, r12\)

**Question 9:** (30pt) Describe the concept of delayed branches and explain how it helps improve program performance.

The concept of delay branches shift some complexities in handling control hazards to software. With it, the instruction after a conditional branch (the delay slot) is always executed regardless the condition of the branch. It is up to the programmer/compiler to find a useful instruction to place in the delay slot, avoiding the pipeline stalling after the branch and hence improving performance.
**Question 10** (40pt) Consider the code below running on a 10MHz pipelined MIPS. The loop iterates 5000 times. (15pt) 5% of the memory references are cache misses. A cache miss take extra 20 cycles to access the main memory. Compute CPI, and MIPS of the program.

This is similar to the last practice question in the final-review handouts. Answers are omitted.

```assembly
loop:
    lw     r4, 0(r1)
    addi   r1, r1, 4
    xor    r10, r3, r4
    add    r6, r4, r6
    sw     r6, 0(r3)
    addi   r7, r7, 1
    bne    r7, r8, loop
    addi   r3, r3, 4
```