Close books, close notes.
Calculator OK.

This exam is given under the Georgia Tech Honor Code System. You must observe and sign the Honor Pledge: “I have neither given nor received aid on this exam.” Your print name and signature below signifies your compliance with this honor code.

Name (Print): SOLUTION

Signature: _________________________________________________

1. _____________ (15 pts)

2. _____________ (15 pts)

3. _____________ (20 pts)

4. _____________ (20 pts)

5. _____________ (20 pts)

6. _____________ (10 pts)

Total (100 pts) ____________
1. **General Q&A.** (15%) Keep your answers concise and brief.

1.1. (2%) What is the performance advantage of using a writeback cache over a write-through cache?

A write-through cache needs to update lower level memory for every store instruction whereas a writeback cache can reduce the bus bandwidth to lower level memory by only updating dirty data during eviction.

1.2. (3%) Is it alright to have a design that results in a TLB hit together with a page fault (i.e. the page not in memory) for the same virtual address? Please explain your answer.

A page fault indicates the corresponding page is not in memory. A TLB hit means there is a valid translation found in the TLB. If TLB indicates a hit, it should not generate a page fault. In other words, when a page is evicted from the memory, its corresponding TLB entry must be invalidated and its page table valid bit should be reset.

1.3. (5%) We never discuss the following code sequence that moves data from one location to another. **(1)** What data dependency it has? **(2)** If forwarding is needed, illustrate in the figure below which pipeline stage the “multiplexor” and “forwarding comparison logic” should be placed? (You have to draw these 2 blocks in the figure. But the only control signals you need to draw are the signals going in and coming out from the mux. You can draw the forwarding logic as a block and insert it in the right stage without any signal.)

```
lw $5, ($10)
sw $5, ($20)
```
1.4. (5%) The following routine is to ensure mutual exclusion for 2 threads, T0 and T1. Two flags, flag0 and flag1 are initialized to 0 for T0 and T1, respectively. Describe a scenario in which the following mutex implementation leads to infinite waiting (or a deadlock) for both threads.

```c
void enteringCriticalSection(int pid) {
    if (pid==0) {
        flag0 = 1;
        while (flag1) {
            thread.yield();
        }
    } else {
        flag1 = 1;
        while (flag0) {
            thread.yield();
        }
    }
}
```

The mutex fails when a context switch occurs after t0 sets flag0=1, and then t1 sets flag1=1. At this moment, both flags are set to 1. Thereafter, both threads t0 and t1 will yield indefinitely and no one will actually enter the critical section.
2. **Performance** (15%) Given two processors P and Q running at 2.8 GHz and 3.0 GHz respectively. These two machines were designed based on the same ISA. The instructions are classified into 4 types: integer, memory, branch and floating-point. Due to the constraint of different frequency objectives, these 2 processors have different instruction latencies as shown in the table. Also note, both processors are single-issue, non-pipelined. You are asked to evaluate the performance by running a benchmark program K. The instructions breakdown for K is shown in the table.

<table>
<thead>
<tr>
<th>Instruction Types</th>
<th>Instruction Latency for P</th>
<th>Instruction Latency for Q</th>
<th># of Instructions in Benchmark K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer (I)</td>
<td>1</td>
<td>2</td>
<td>25 billions</td>
</tr>
<tr>
<td>Memory (M)</td>
<td>3</td>
<td>2</td>
<td>15 billions</td>
</tr>
<tr>
<td>Branch (B)</td>
<td>2</td>
<td>1</td>
<td>30 billions</td>
</tr>
<tr>
<td>Floating-point (F)</td>
<td>1</td>
<td>2</td>
<td>30 billions</td>
</tr>
</tbody>
</table>

2.1. (5%) Which machine actually delivers better overall performance (measured in time)? Please derive your answer.

Machine P: \(1 \times 25 + 3 \times 15 + 2 \times 30 + 1 \times 30 = 25 + 45 + 60 + 30 = 160\) billion cycles

Machine Q: \(2 \times 25 + 2 \times 15 + 1 \times 30 + 2 \times 30 = 50 + 30 + 30 + 60 = 170\) billion cycles

Time for Machine P = \(160 \times (1/2.8) = 57.14\) seconds

Time for Machine Q = \(170 \times (1/3) = 56.66\) seconds

Q is almost 1% faster than P
2.2. (10%) For machine P, architects invent a new instruction called I-M that can combine some integer with memory instruction into one instruction. The new instruction has a 2-cycle latency. For example, if one can find 10 billion such instructions to combine, the I and M instructions for P will be reduced to 15 and 5 billions respectively while with additional 10 billion I-M instructions. However, due to the addition of this new instruction, the frequency is slowed down to 2.7GHz. In order to receive speedup on P for using this new class of instructions, how many I and M instructions need to be found and combined?

Assume there are X instructions that can be combined.
I-M: X I: 25-X M: 15-X B: 30 F: 30
New cycle = X*2 + (25-X)*1 + (15-X)*3 + 30*2 + 30*1 = 2X+25-X+45-3X+90 = 160-2X
New time = (160-2X)*(1/2.7)
(160-2X)/2.7 < 57.14
X >2.857
Thus, at least 2.857 billion instructions need to be found and combined
3. **Pipelining.** (20%) Given the following MIPS program being executed in a 5-stage pipeline as shown in next page. 0x2ff100fc is the address of the first instruction and each instruction is 4 bytes. The corresponding data and control signals are provided in the table next page. **At the end of clock cycle 5** when all instructions are in the pipeline, fill the blanks with the resulting values in **hexadecimal**. Assume the registers are all loaded with the register number prior to execution and each data memory byte is loaded with the **least significant byte** of its own address. For example, 0x00000001 contains 0x01, 0x00000002 contains 0x02, 0x0000000F contains 0x0F, and so on. Thus, if lw from 0x00000000, one will get 0x03020100; lw from 0x00000004 will get 0x07060504. Also assume a data write always takes place in the 1\textsuperscript{st} half of a cycle and a read always takes place in the 2\textsuperscript{nd} half of a cycle. And there is no branch prediction hardware.

\[\begin{align*}
0x2ff100fc: & \quad lw \quad $9, \quad 20($8) \\
& \quad addi \quad $8, \quad $8, \quad -8 \\
& \quad addi \quad $21, \quad $21, \quad 16 \\
& \quad sw \quad $9, \quad 4($21) \\
& \quad j \quad 0x30030004 \\
\end{align*}\]

**Data signals**

Read data 1 in ID = 0x\underline{\phantom{00000000}}\underline{00000015} \underline{\phantom{00000000}}

Read data 2 in ID = 0x\underline{\phantom{00000000}}\underline{1F1E1D1C} \underline{\phantom{00000000}}

Output of sign extend in ID = 0x\underline{\phantom{00000000}}\underline{00000004} \underline{\phantom{00000000}}

ALU output in EX = 0x\underline{\phantom{00000000}}\underline{00000025} \underline{\phantom{00000000}}

Input value to PC = 0x\underline{\phantom{00000000}}\underline{2ff10110} \underline{\phantom{00000000}}

MEMpass = 0x\underline{\phantom{00000000}}\underline{00000008} \underline{\phantom{00000000}}

**Pipeline Control signals**

WB in ID/EX for EX = 0x\underline{\phantom{00000000}}\underline{2} \underline{\phantom{00000000}}

WB in MEM/WB for WB = 0x\underline{\phantom{00000000}}\underline{00000003} \underline{\phantom{00000000}}

M in EX/MEM for MEM = 0x\underline{\phantom{00000000}}\underline{00000000} \underline{\phantom{00000000}}

EX in ID/EX for EX = 0x\underline{\phantom{00000000}}\underline{00000007} \underline{\phantom{00000000}}
Table 1. Control signal table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RegDst</td>
<td>ALUop1</td>
<td>ALUop2</td>
</tr>
<tr>
<td>R-format</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I-format</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>x</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
4. Cache (20%)

4.1. (10%) Given a 4-way 32KB cache with 128 sets. For a 48-bit architecture, how big the entire tag array (in bits or bytes) is needed? (just the tags, you don’t need to account for valid, dirty or LRU bits.)

Line size = 32KB/(4*128) = 64B
Each tag has = 48 – 7 – 6 = 35 bits
Total tag array for 4 ways = 35 * 128 * 4 = 17920 bits = 2240 Bytes = 2.1875 KBytes

4.2. (10%) Following the previous question, now the architects decide to alter the cache into a fully-associative cache but keeping the 32KB capacity and each cache line size the same. For the same 48-bit architecture, how big the entire tag array (in bits or bytes) is needed?

For a fully associative cache, there is one single set, thus it has 32KB/64B = 512 ways
Each tag has = 48 – 6 = 42 bits
Total tag array for such a cache = 42 * 512 = 21504 bits = 2688 bytes = 2.625 KB
5. Virtual memory (20%) Given a 36-bit processor with 4 active processes being executed concurrently. Please answer the following questions. Show all the addresses of your answer in hex number. If a translation cannot be found, enter page fault.

5.1. (5%). Assume an inverted page table (IPT) is used by the OS. The IPT is shown below (Only Valid, PID and VPN are shown). Each page size is 4MB. What “virtual address” of which “process” maps to the physical address “0x363055B”?

4MB → 22 bits in the page offset

0x363055B = 0011 0110 0011 0000 0101 0101 1101

PPN = 1101 = 13

Look up the 13th entry in the table (from 0 to 15), it maps to a valid entry used by

PID = 9, VPN = 0x020D

Virtual Address= 1000 0011 0110 0011 0000 0101 0101 1101

= 0x8363055B of Process PID=9
5.2. **Indexed based page table** (5%) Now we switch to use an **index-based linear page table**, how much memory (in KB) is required for just process A? Assume each page table entry (PTE) contains a valid and dirty bit.

VPN = 36 – 22 = 14 bits → 16K entries
Each PTE = 1 valid + 1 dirty + PPN
From the table, we know the PPN has 4 bits (16 entries)
Each PTE = 6 bits
Page table size for one process = 6 * 16K = 96 Kbits = 12 KB

5.3. **Hierarchical page table** (10%). Now we consider to switch to **2-level hierarchical linear page tables**. The address split for indexing is half-half for the first and the second level. Based on the given page table in 5.1, how much memory (in KB) is required for process A? Assume each **pointer is 4 bytes** and each PTE also contains a valid bit and a dirty bit.

There are 6 valid pages for process A in the physical memory
VPN = 0x3630, 0x3640, 23A4, 0D7C, 0DF0, 31A2

Breaking up VPN to be 7 + 7 for two levels
0x3630 = 11 0110 0011 0000 = 6C | 30
0x3640 = 11 0110 0100 0000 = 6C | 40
0x23A4 = 01 0011 1010 0100 = 27 | 24
0x0D7C = 00 1101 0111 1100 = 1A | 7C
0x0DF0 = 00 1101 1111 0000 = 1B | 70
0x31A2 = 11 0001 1010 0010 = 63 | 22

For the first level, we need \(2^7 \times 4\) bytes pointers = 512 bytes
For the second level, there are 5 distinctive page tables for pointer address 6C, 27, 1A, 1B and 63
5 * \(2^7\) * (1+1+4) bits = 30 * 128 bits = 480 Bytes
Total size = 512 + 480 = 992 bytes
6. Process Scheduling. (10%) Show your schedule with timeline and Calculate the **average “turnaround” time** when use the **multi-level feedback queue** as below. (Please take arrival time into account.) Note that the priority of the top 2 queues is based on arrival times.

<table>
<thead>
<tr>
<th>Process ID</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

This is the same table, shown here for your convenience

<table>
<thead>
<tr>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 18</td>
</tr>
<tr>
<td>B = 28</td>
</tr>
<tr>
<td>C = 18</td>
</tr>
<tr>
<td>D = 21</td>
</tr>
<tr>
<td>E = 6</td>
</tr>
</tbody>
</table>

*Average = 91/5 = 18.2*