Problem 1. Beta Implementation (6 points)

Ben Bitdiddle is proposing the short assembly language program shown to the right as a manufacturing test to ensure the correct operation of the Control ROM.

He is assuming – and you may too – that the Beta datapath components (e.g., Memories, ALU, muxes, register file, adders) are working correctly and that any errors in execution are due to faulty signals from the Control ROM. Ben’s plan is to run the program then look at the value in the memory location labeled ANS. If the value is 0x6004, the test passes, otherwise the Beta being tested is declared faulty and discarded.

For each of the following faults, indicate the value that the faulty Beta will store into ANS.

(A) (2 points) RA2SEL is stuck at the value 0.

Value stored in ANS by faulty Beta: ______________

(B) (2 points) WDSEL[1:0] is stuck at the binary value 00.

Value stored in ANS by faulty Beta: ______________

(C) (2 points) PCSEL[2:0] is stuck at the binary value 000.

Value stored in ANS by faulty Beta: ______________

Test: LD(R31,X,R0)
ADDC(R0,1,R1)
BNE(R1,L1,R31)

L1: ST(R1,ANS,R31)
HALT()

X: .LONG(0x6003)
ANS: .LONG(0)
Problem 2. Caches (12 points)

Consider a 2-way set-associative cache where each way has 4 cache lines with a **block size of 2 words**. Each cache line includes a valid bit (V) and a dirty bit (D), which is used to implement a write-back strategy. The replacement policy is least-recently-used (LRU). The cache is used for both instruction fetch and data (LD,ST) accesses. Please use this cache when answering questions (A) through (D).

(A) (2 Points) Using this cache, a particular benchmark program experiences an average memory access time (AMAT) of 1.3 cycles. The access time on a cache hit is 1 cycle; the miss penalty (i.e., additional access time) is 10 cycles. What is the hit ratio when running the benchmark program? You can express your answer as a formula if you wish:

\[
\text{Hit ratio for benchmark program: } \frac{\text{AMAT}}{\text{AMAT} + \text{Miss penalty}}
\]

(B) (3 Points) The circuitry for this cache uses various address bits as the block offset, cache line index and tag field. Please indicate which address bits A[31:0] are used for each purpose by placing a “B” in each address bit used for the block offset, “L” in each address bit used for the cache line index, and “T” in each address bit used for the tag field.

**Fill in each box with “B”, “L”, or “T”**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(C) (2 Points) This cache needs room to store new data and based on the LRU replacement policy has chosen the cache line whose information is shown to the right for replacement. Since the current contents of that line are marked as dirty (D = 1), the cache must write some information back to main memory. What is the address of each memory location to be written? Please give each address in hex.

**Addresses of each location to be written (in hex):**

Way: 0
Cache line index: 3
Valid bit (V): 1
Dirty bit (D): 1
Tag field: 0x123
(D) (5 Points) This cache is used to run the following benchmark program. The code starts at memory address 0; the array referenced by the code has its first element at memory address 0x2000. First determine the number of memory accesses (both instruction and data) made during each iteration through the loop. Then estimate the steady-state average hit ratio for the program, i.e., the average hit ratio after many iterations through the loop.

```assembly
.data
  . = 0
  CMOVE(0,R0)       // byte index into array
  CMOVE(0,R1)       // initialize checksum accumulator
loop:
  LD(R0,array,R2)   // load next element of array
  SHLC(R1,1,R1)     // shift checksum
  ADDC(1,R1,R1)     // increment checksum
  ADD(R2,R1,R1)     // include data value in checksum
  ADDC(R0,4,R0)     // byte index of next array element
  CMPLTC(R0,1000,R2) // process 250 entries
  BT(R2,loop)
  HALT()

.array
  . = 0x2000
  array:
    ... array contents here ...
```

Number of memory accesses made during each iteration of the loop: __________

Estimated steady-state average hit ratio: __________
Problem 3. Pipelined Beta (12 points)

In answering this question, you may wish to refer to the diagram of the 5-stage pipelined beta provided with the reference material.

The loop on the right has been executing for a while on our standard 5-stage pipelined Beta with branch annulment and full bypassing. The pipeline diagram below shows the opcode of the instruction in each pipeline stage during 10 consecutive cycles of execution.

<table>
<thead>
<tr>
<th>Cycle #</th>
<th>300</th>
<th>301</th>
<th>302</th>
<th>303</th>
<th>304</th>
<th>305</th>
<th>306</th>
<th>307</th>
<th>308</th>
<th>309</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>SUBC</td>
<td>CMPLTC</td>
<td>BF</td>
<td>LD</td>
<td>LD</td>
<td>ST</td>
<td>BNE</td>
<td>BNE</td>
<td>BNE</td>
<td>ADDC</td>
</tr>
<tr>
<td>RF</td>
<td>SUBC</td>
<td>CMPLTC</td>
<td>BF</td>
<td>NOP</td>
<td>LD</td>
<td>ST</td>
<td>ST</td>
<td>ST</td>
<td>BNE</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>SUBC</td>
<td>CMPLTC</td>
<td>BF</td>
<td>NOP</td>
<td>LD</td>
<td>NOP</td>
<td>NOP</td>
<td>ST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td>SUBC</td>
<td>CMPLTC</td>
<td>BF</td>
<td>NOP</td>
<td>LD</td>
<td>NOP</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>SUBC</td>
<td>CMPLTC</td>
<td>BF</td>
<td>NOP</td>
<td>LD</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(A) (4 Points) Indicate which bypass/forwarding paths are active in each cycle by drawing a vertical arrow in the pipeline diagram from pipeline stage X in a column to the RF stage in the same column if an operand would be bypassed from stage X back to the RF stage that cycle. Note that there may be more than one vertical arrow in a column.

Draw bypass arrows in pipeline diagram above

(B) (2 Points) Assume that the previous iteration of the loop executed the same instructions as the iteration show here. Please complete the pipeline diagram for cycle 300 by filling in the OPCODEs for the instructions in the RF, ALU, MEM, and WB stages.

Fill in OPCODEs for Cycle 300

For the following questions think carefully about when a signal would be asserted in order to produce the effect you see in the pipeline diagram.

(C) (2 Points) During which cycle(s), if any, would the IRSrcIF signal be 1?

Cycle number(s) or NONE: ______________

(D) (2 Points) During which cycle(s), if any, would the IRSrcRF signal be 1?

Cycle number(s) or NONE: ______________

(E) (2 Points) During which cycle(s), if any, would the STALL signal be 1, i.e., cycle(s) when the IF and RF stages would be stalled?

Cycle number(s) or NONE: ______________

END OF QUIZ3!