Name: Solution

a) Translate the following code into Tiny8v1 assembly (5 points):

```
int *src_ptr = 0x.....
int *dest_ptr = 0x....
int I = 0;
int sum = 0;
for(int I = 0; I < 16; i++) {
   sum += src_ptr[2*i] * 20
   dest_ptr[i] = sum
}
```

```
; r0 holds ~~address of~~ src_ptr = 0xA0
; r1 holds ~~address of~~ dest_ptr = 0xC0
; r2 is uninitialized
; r3 initially holds I = 15
; acc is 0x45 initially
; YOUR CODE HERE
```

b) Your boss takes a look at Tiny8v1 and commends your initiative. However, she tells you that there is a huge functional omission which would make running the program given in part A impossible in practice. What common operation is impossible to perform with Tiny8v1? (3 points)

Name: Solution

c) With the realization of part b), you decide to fix the ISA for Tiny8v2. How would you add the operation from part b) without losing any original functionality? (3 points)

d) Your boss is happy with the capabilities of Tiny8v2. However, DRAM accesses are very costly in terms of energy for the prototype Tiny8v2 chips running the code from part a). She suggests that you examine your d-cache organization and minimize number of memory accesses without adding any instructions.

The cache specifications are:

- 32 byte capacity
- 4 byte cache lines
- direct-mapped

Propose a software optimization to minimize cache misses. (4 points)

Name: _____

2. MP

Testing and debugging are critical to processor development. You, as a student in ECE 411, just finished implementing RTL code for MP1, and have written some simple test code to test the load instructions.

The first instruction you want to test is LDR. Following is the test code. Note: keyword data8 puts an 8-byte word into memory with a little endian fashion.

```
ORIGIN 4x0000

LDR R4, R0, long9

HALT:
BRnzp HALT

long1: data8 4x12345678abcdefff
long2: data8 4x12345678abcdefff
long3: data8 4x12345678abcdefff
long4: data8 4x12345678abcdefff
long5: data8 4x12345678abcdefff
long6: data8 4x12345678abcdefff
long7: data8 4x12345678abcdefff
long8: data8 4x12345678abcdefff
long9: data8 4x12345678abcdefff
long10: data8 4x12345678abcdefff
```

a) The above test code, when executed, will cause an error due to hardware constraints. Identify what the bug is. (3 points)

Name: _____

b) You changed the code to be

```
ORIGIN 4x0000

ADD R0, R0, 2
LDR R4, R0, long1

HALT:
BRnzp HALT
```

And the data segment stays the same. What is the value expected to be loaded to R4? (2 points)

The second instruction you want to test is LDI. Following is the test code.

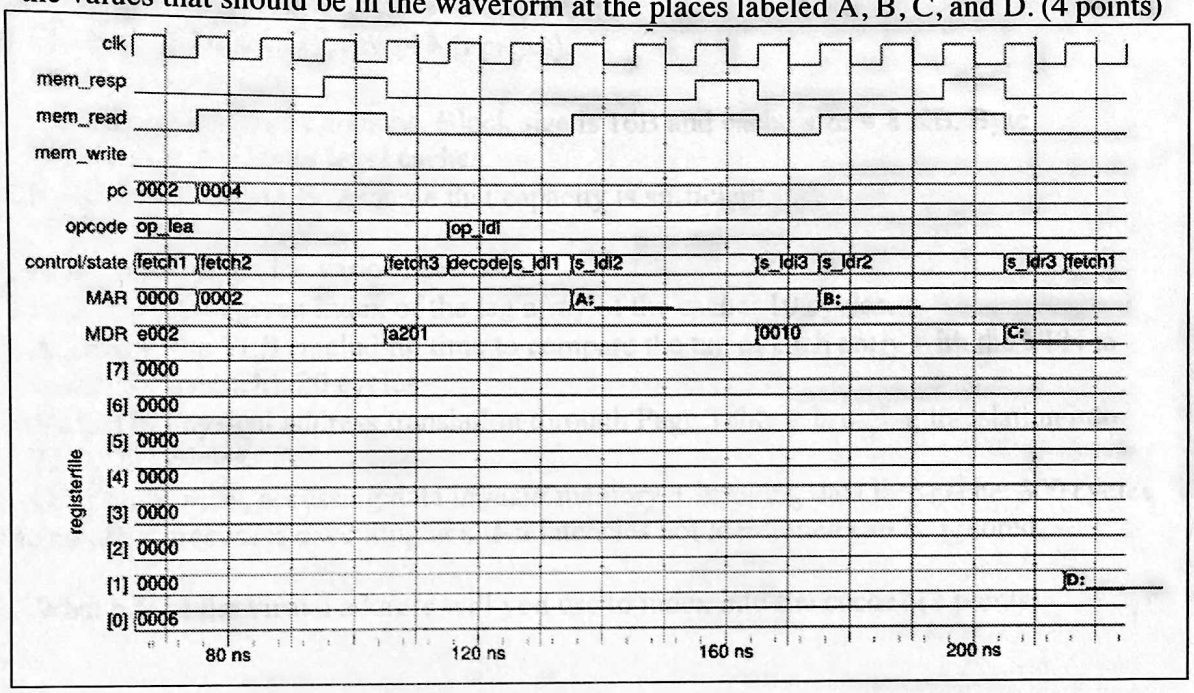
```
ORIGIN 4x0000
SEGMENT codeBlock:

LEA R0, dataBlock
LDI R1, R0, ptr2
HALT:
BRnzp HALT

SEGMENT dataBlock:
ptr1: data2 val1
ptr2: data2 val2
ptr3: data2 val3
val1: data2 4xbaad
val2: data2 4x600d
val3: data2 4xffee
```

Name: _____

c) The following waveform shows the LDI instruction executing for the above code. Fill in the values that should be in the waveform at the places labeled A, B, C, and D. (4 points)



A: _____ B: _____

C: _____ D: _____

d) You passed the unit test. But when you run a larger test code, LDI still causes error. What aspect of the LDI is not tested in the given code? Assume that all the registers values shown in the trace are set correctly (including the ones you filled in). (2 points)

Cache+VM

Consider a system with the following characteristics :

- 32 bit virtual address space (4 KB pages) .
- 1 GB main memory
- VIPT direct mapped L1 cache . Block size is 16B and cache size = 8KB . Byte addressable . No lower level caches .
- Fully associative TLB . Assume that capacity is sufficient .

These are the times taken for various accesses :

- Indexing into a given index of the tag array of the cache : 10 cycles .
- Accessing the TLB (including time to compare the tag of each entry with the VPN to search for a match) : 20 cycles
- Virtual to Physical address translation through Page Table + bringing translation into TLB : 200 cycles .
- On a cache miss , accessing data in main memory + bringing data into cache : 300 cycles

Assume no other process is executing and that latencies not mentioned can be ignored .

a) What bits of the virtual address will you use to index into the cache ?

b) What are the number of bits in the physical address and what is the bit range that the Physical Page number occupies ?

c) Consider the following sequence of **virtual address** accesses . Fill in the cycles required for each access and also fill in the accessed/created TLB and cache entries. . Assume that both the cache and TLB are initially empty.

Virtual address accessed	Time taken for access (cycles)
0x00000000	
0x0000000f	
0x0001000a	
0x0000de00	

0x0001010d	

L1 cache - Fill in the accessed indices along with the tag stored at each index after the sequence of accesses. Values can be in decimal.

Index	Tag

Required translations are given in the **page table** . Assume all VPN-PPN mappings are valid for the executing process .

VPN	PPN
0	10
4	12
3	7
16	20
13	4
32	8

TLB - Fill in the VPN - PPN mappings created during the sequence of accesses .

VPN	PPN

d) Now assume 2 processes P1 and P2 run on the CPU (only one at a time) .

- They share a physical page with PPN x . They also have a producer-consumer relationship , where P1 writes into the physical page x at a known offset , and P2 reads from physical page x at that particular offset . Assume this offset is known to both processes .

1. In process P1 , virtual page with VPN (2*k1) is mapped to PPN x, and in process P2, virtual page with VPN (2*k2+1) is mapped to PPN x, where k1 and k2 are integers. Explain the problem that arises due to this mapping , using an example if necessary.

2. Keeping the L1 cache VIPT , name 2 modifications that can be made to the cache in order to solve the problem encountered .