Problem 4 (30 points)

We wish to enhance the x86 ISA by adding a new instruction. The new instruction is called `STI`, "Store Indirect", and its format is:

```
STI Ra,Rb,Offset
```

The opcode of `STI` is 1010, and its binary encoding is (2-Byte long):

Opcode <4-bit>	Ra <3-bit>	Rb <3-bit>	Offset <6-bit>

`STI` operates as follows: We compute a virtual address (call it `A`) by adding the sign-extended `Offset` to the contents of register `Rb`. The memory location specified by `A` contains the virtual address `B`. We wish to store the contents of register `Ra` into the address specified by `B`.

The processor has a simple one-level virtual memory system. There is also a 2-entry TLB. You are given the following information:

- Virtual Address Space: 64 KB
- Physical Memory Size: 4 KB
- PTE Size: 2 Bytes
- The format of a PTE is shown below. The MSB is the valid bit, and the lower several bits are for the physical page number (PPN). Note that the exact number of bits for PPN is for you to determine. The rest bits are always padded with 0.

Valid <1-bit>	0...0	Physical Page Number

- `%eax`: 0x8000
- `%ebx`: 0x401E
- Program Counter (`%eip`): 0x3048

The TLB state before any instructions related to this problem are executed:

Valid	Virtual Page Number (VPN)	PTE	
		Valid	Physical Page Number (PPN)
1	0x0C1 0000 1100 0001	1	0x01A
1	0x182 0001 1000 0010	1	0x024

Part a (2 points): In this particular TLB, the Valid bit in the first column and the Valid bit in the third column are the same in both TLB entries. In general, is it possible that these two valid bits have different values?

Yes

Part b (6 points): What is binary encoding for `STI %eax,%ebx,0`? Assume that `%eax` is encoded as 0 and `%ebx` is encoded as 1.

1010 000 001 000000

Part c (4 points): To process the `STI` instruction, one must go through the Fetch, Decode, etc. instruction cycle. What is the maximum number of physical addresses that can be accessed in processing an `STI` instruction?

Hints:

1. Instruction fetch is the necessary first step in processing any instruction
2. In the one-level virtual memory system, the page table lives in the physical memory

6

This instruction accesses virtual memory three times: fetch instruction, load from address A, and store to address B. Each virtual memory access could at most lead to 2 physical memory accesses: one for accessing the PTE and the other for accessing the actual data.

Part d (18 points): Now the processor executes `STI %eax,%ebx,0`. It turned out that five physical memory accesses were needed. The table below shows the Virtual Address (VA), Physical Address (PA), Data, and whether or not there was a TLB hit for each of these five physical memory accesses in the order they occurred. Some of the blanks are intentionally left for you to fill in.

	Virtual Address	**Physical Address**	**Data**	**TLB Hit?**
Fetch Inst.	0x3048 PC	0x488	0xA040	Yes
Get PTE for A	**N/A**	**0x660**	**0x8040**	No
Load from A	0x401E %ebx (A)	0x81E	**0x40FE** B	No
Get PTE for B	**N/A**	0x66E	0x800E	No
Write to B	0x40FE	**0x1DE**	0x8000 %eax	No

Complete the table and fill in the following three boxes. You can assume that no page faults occurred.

Hints:

1. What does the first TLB hit mean?
2. Recall how to use PTBR and VPN to access the page table.
3. Use the first and last accesses to figure out the page size first. Everything else will follow.

(3 points): What is the page size?

32 Bytes

(3 points): What is the total number of physical pages?

128

(3 points): What is the data in the page table base register (PTBR)?

0x260

(9 points; 1 point per blank): Please complete the table above. "I Don't Know" is accepted on a per blank basis.

* The first virtual memory access must be to fetch the instruction, so the first VA must be the PC, which is 0x3048.
* The first access is a TLB hit. Analyzing the VA and the two TLB entires, you would know that the page offset must be either 5 or 6.

```
0x0C1:    0000 1100 0001
0x182:    0001 1000 0010
0x3048:  0011 0000 0100 1000
```

* The second access must be to get the PTE of virtual address A, which returns 0x8040 as the PTE. Using the PTE, we form the physical address, which the third access uses to get the data 0x40FE, which is essentially B.
* The fourth access must be then to get the PTE of virtual address B and form its physical address, which is 0x1DE. The fifth access must be to store %eax to 0x1DE (corresponding to virtual address B). So its VA must be 0x40FE, same as the data returned from the third access. The data in the fifth access is 0x8000 (i.e., %eax).
* Focus on the last access. We know its VA is 0x40FE and the PA is 0x1DE. Their page offsets must match. Analyzing the two address, we know the page offset must be 5 bits rather than 6.

```
0x1DE:           0001 1101 1110
0x40FE:  0100 0000 1111 1110
```

* If the page offset is 5 bits, we could easily get that the physical page number is 12-5=7 bits since the total physical memory size is 4K = 2^12.
* Given the 5 bits page offset, we could also easily get know that the TLB hit in the first access hits on the second TLB entry, which allows us to form the physical addresses of the first and the third access, which are 0x480 and 0x81E, respectively.
* To get the PTBR value, focus on the second physical address, which is calculated by PTBR + VPN * 2. Since we know the page offset is 5, we would know that VPN in that access is 0x200, and so we can get the PTBR = 0x260.
* Given the PTBR, we could also get the physical address of the fourth access, which is PTBR + VPN * 2 = 0x66E.