ECE203 HW6 Solution

3. (10 pts)

For the RSE architecture we started discussing in class, show reasonable encodings for each of the eight instructions. For example, the add instruction might have the following encoding:

instruction	0–2	3–5	6–8	9–11	12–15
add	000	R_D	R_{S1}	R_{S2}	XXXX

instruction	0-2	3-5	6-8	9-11	12-15
add	000	R_D	R_{S1}	R_{S2}	XXXX
sub	001	R_D	R_{S1}	R_{S2}	XXXX
ldm	010	R_D	$[R_S]$	XXX	XXXX
stm	011	$[R_D]$	R_S	XXX	XXXX
ldi	100	R_D	i_7-i_5	i_4-i_2	i_1i_0XX
ldpc	101	R_S	XXX	XXX	XXXX
blz	110	R_T	R_C	XXX	XXXX
bz	111	R_T	R_C	XXX	XXXX

4. (5 pts)

Write RSE assembly code to multiply the contents of register A by three.

```
add B,A,A
add A,A,B
```

5. (10 pts)

Write RSE assembly code to write a value K…K +n, into a given memory range, M … M+n. For example, if K=5, M=24, and n=2, then [24]=5, [25]=6 and [26]=7. K is initially in register A, M is initially in register B, and n is initially in register C. Assume your first instruction sits at memory location 2. For each instruction, supply a short comment (perhaps something as simple as “K=K+1”) explaining the purpose of the instruction. Remember that you have five general-purpose registers.

This is just one possible solution.

instruction	comment
ldi D, 1	store 1 to D
ldi E, 6	
stm [B], A	store a value to mem located at [B]
add A, A, D	K=K+1
add B, B, D	M=M+1
sub C, C, D	stop at 0.
blz E,C	if n<0, go to LOOP

6. (5 pts)
How many bits must RSE's PC register have? Justify your answer in two or fewer sentences.

For the width of PC, it depends on several factors: first, whether there is any instruction which will use the PC address. Second, whether the designer gives the specification on how big the memory is. In our case, since we're told that our memory is limited to 255 addresses, we can use 8-bits for the PC.

7. (5 pts)
If you were permitted to make one or two simple changes to the RSE architecture, what would they be?

Again, this answer is open to some variation, but some of the better ideas to change RSE would be to add extra instructions. Since our opcode is 16 bits long, and all of the codes use at most 14 bits, we can double the possible number of operations by simply increasing our instruction code by 1 bit. Thus we can add useful operations including a branch when greater than zero. Another possible change would be to increase the number of registers. This can be done without any loss of bits because we only have 5 registers, but we can have up to 8 registers (since we use 3-bits to reference them). This can save us clock cycles by allowing us to minimize the number of store/load operations.