Standard Coding Sheet **Student Number: 100225776**

Question 1a:

Shift a number stored in a0 five places to the right and store the result in b0.

Register	0	1	2	3	4	5	6	7
Purpose		Number to Shift	Mask 00000111 (07)	Result of Shift				

00	11a0	Load Register 1 with the value at address a0, the number to be shifted
02	2207	Load Register 2 with bit pattern 07 (0000 0111), the AND mask
04	a105	Rotate the value in Register 1 five places to the right
06	8312	AND the rotated result with the mask in Register 2, store in Register 3
08	33b0	Store the result from the masking at address b0
0a	c000	Halt execution

Question 1b:

Swap the 1st 4 bits of a value in a0 with the last 4 bits, store the result in b0.

There doesn't need to be any information stored in registers for this since it is essentially rotating the number 4 bits around and saving it.

Register	0	1	2	3	4	5	6	7
Purpose		Number to Shift						

00	11a0	Load Register 1 with the value at address a0, the number to be rotated
02	a104	Rotate the value in Register 1 four places to the right
04	31b0	Store the shifted number to address b0
06	c000	Halt execution

Question 1c:

Given two values, one at a0 and one at a1 determine if the last 4 bits are the same. If they are store 01 at address b0, if different store 00 at b0.

Register	0	1	2	3	4	5	6	7
Purpose	Second number to compare	First number to AND/ compare	Second number to AND	AND mask (0F)	Value 00 (if bits don't match)	Value 01 (if bits do match)		

00	11a0	Load Register 1 with the value at address a0, the first number
02	12a1	Load Register 2 with the value at address a1, the second number
04	2307	Load Register 3 with bit pattern 0F (0000 1111), the AND mask
06	2400	Load Register 4 with bit pattern 00, the first number for storage
08	2501	Load Register 5 with bit pattern 01, the second number for storage
0a	8113	AND Register 1 and 3 to clear first 4 bits from Register 1 value, store in Reg1
0c	8023	AND Register 2 and 3 to clear first 4 bits from Register 2 value, store in Reg0
0e	b114	Jump to instruction 14 if the output from the last instruction matches reg1
10	34b0	Store value in Register 4 (00) at address b0, as the numbers don't match
12	c000	Halt execution
14	35b0	Store value in Register 5 (01) at address b0, since numbers match
16	C000	Halt execution

Question 2a:

Multiply a number in a0 in the range 0 – 8 by 8 and store the result in b0.

Since we know the number to multiply with will always be 1000_2, we don't need to store or reference it, saving RAM, registers and cycles. The first three bits are 0 so we essentially ignore them and rotate the number left by three. Since the value in a0 is only using 4 bits we don't need to worry about the last four bits flowing back around so we can just store the result of the shift.

Register	0	1	2	3	4	5	6	7
Purpose		Value to multiply						

00	11a0	Load Register 1 with the value at address a0 (value must be 00_{hex} to 08_{hex})
02	a105	Shift Register 1 three places left (five to the right)
04	31b0	Store the shifted number to address b0
06	c000	Halt execution

Question 2b:

Sort two numbers in a0 and a1 in descending order and store the result in a0 and a1. I take the first number away from the second and work out if the number is negative, if it is than it means the first number was larger and should be rotated with the second, the values are then stored. The downside of this is that both numbers are treated as signed integers therefore values over $7F_{hex}$ (127_{10}) are treated as negative decimal numbers and will be sorted accordingly.

Register	0	1	2	3	4	5	6	7
Purpose	Bit pattern 80	First number to compare	Second number to compare	Result	Bit pattern 01	Register for swapping values	XOR mask ff – 2s comp	

00	2080	Load Register 0 with the bit pattern 80 for negative test
02	11a0	Load Register 1 with the value at address a0, the first number
04	12a1	Load Register 2 with the value at address a1, the second number
06	2401	Load Register 4 with the value 01 for adding in 2s complement
08	26ff	Load Register 6 with bit pattern ff (1111 1111) for XOR mask
0a	9316	XOR Register 1 with ff to find 2s complement, store in Register 3
0c	5343	Add one to Register 3 to complete 2s complement
0e	5323	Add Register 2 and 2s complement in Register 2, overwrite Register 3
10	8303	AND Register 0 with the result to see if the number is negative
12	b31e	Jump to end if the result is negative (as numbers started in correct places)
14	4025	Move the second number to temp register
16	4012	Move the first number to second number's register
18	4051	Move the second number over into the first number's register
1a	31a0	Store the highest value to address a0
1c	32a1	Store the lowest value to address a1
1e	c000	Halt execution

Question 2c:

Reverse the order of bits in a byte stored in a0. Store the results in b0. I made this one more efficient by using Register 0 to both hold the value 01 for the AND mask as well as be the value that would increase the counter and the value that the counter was counting towards. This does require that the counter overflow from ff to 00 however, which I accept might not be suitable for all instruction sets.

I believe I have found a bug in the Brookshear Machine through testing my code with this question. All numbers that I have been able to test produce the correct answer except the value 01, which gives 00 as an answer rather than 80. It seems that when the machine runs the 8^{th} iteration of the below loop of code and executes instruction 5545 it should do nothing to the contents of register 5, as register 4 holds the value 0, however it instead it causes register 5 to reset from 80_{hex} to 00 before continuing correctly. From what I understand on the topic I believe my code as shown below should otherwise perform correctly and the fault lies within the program, so I have chosen not to write in lines to address this single use-case.

Register	0	1	2	3	4	5	6	7
Purpose	AND mask / count 01	Number to reverse	Counter		Current bit	Running total		

00	2001	Load Register 0 with the bit pattern 01 AND mask / counter (as above)
02	11a0	Load Register 1 with the value at address a0
04	22f9	Load Register 2 with bit pattern f9 for a rotation counter
06	2500	Load Register 5 with bit pattern 00 to initialise running total
08	8401	AND given value with 01 mask to keep significant digit, store in register 4
0a	5545	Add the value in Register 4 to the running total
0c	5202	Add Register 0 (01) to counter to increase it
0e	b216	Jump to end of program if counter reaches 01
10	a101	Rotate Register 1 one place to right
12	a507	Rotate Register 5 one place to the left (right 7)
14	b008	Jump back to start of loop if counter is not 0
16	35b0	Store the running total to address b0
18	c000	Halt execution

Question 3:

Sort 3 numbers stored in a0, a1 and a2 into descending order and store the result in a0, a1 and a2. We will test the first two numbers first to see if they're in order, and if not swap them, then check the second and third numbers and swap them if they are not in order. Finally, we go back to the first two numbers to ensure they are in the correct order before storing the results.

Register	0	1	2	3	4	5	6	7
Purpose	Bit pattern 80	First number to compare	Second number to compare	Third number to compare	Comparison Result	Bit pattern 01	XOR mask ff – 2s comp	Register for swapping values
Register	8							
Purpose	Final loop 'flag' bit							

00	2080	Load Register 0 with the bit pattern 80 for negative / counter check
02	11a0	Load Register 1 with the value at address a0
04	12a1	Load Register 2 with the value at address a1
06	13a2	Load Register 3 with the value at address a2
08	2501	Load Register 5 with the value 01 for adding in 2s complement
0a	26ff	Load Register 6 with bit pattern ff (1111 1111) for XOR mask
0c	2801	Load Register 8 with bit pattern 01 as a flag for looping the program
0e	9416	XOR Register 1 with ff to find 2s complement, store in Register 4
10	5454	Add one to Register 4 to complete 2s complement
12	5424	Add Register 2 and 2s complement in Register 4, overwrite Register 4
14	8404	AND Register 0 with the result to see if the number is negative
16	B41e	Jump past moving numbers if the result is negative (first number is larger)
18	4027	Move the second number to temp register
1a	4012	Move the first number to second number's register
1c	4071	Move the second number from temp over into the first number's register
1e	b834	Jump to the end of the program if the final loop flag is set to 80_{hex}
20	4008	Move 80 from Register 0 to Register 8 to set the loop flag
22	9426	XOR Register 2 with ff to find 2s complement, store in Register 4
24	5454	Add one to Register 4 to complete 2s complement
26	5434	Add Register 3 and 2s complement in Register 4, overwrite Register 4
28	8404	AND Register 0 with the result to see if the number is negative
2a	b434	Jump to end if the result is negative (first number is larger)
2c	4037	Move the third number to temp register
2e	4023	Move the second number to third number's register
30	4072	Move the third number over from temp into the second number's register
32	b00e	Jump back to the first comparison to ensure they're in the correct order
34	31b0	Store the highest value to address a0
36	32b1	Store the middle value to address a1
38	33b2	Store the lowest value to address a2
3a	c000	Halt execution

Question 4:

Place the value ff in 10 memory matrix cells, one below the other, starting in cell 27. Only two STORE instructions (instruction 3) should be used. Since the active cell is being increased at the same amount each time, I decided to simply have register 0 hold the value for the cell after the last one I want to save to.

Register	0	1	2	3	4	5	6	7
Purpose	Counter check (c7)	Value to store (ff)	Counter / current address	Bit pattern 10				

00	20c7	Load Register 0 with bit pattern c7 for checking counter
02	21ff	Load Register 1 with bit pattern ff, as value to be stored
04	2227	Load Register 2 with bit pattern 27 for current cell address / counter
06	2310	Load Register 3 with bit pattern 10 for increasing cell address / counter
08	3127	Store the value ff in the currently active cell
0a	5232	Add 10 to the address of the currently active cell / counter
0c	b212	Jump to the end if the current value of the counter is c7
0e	3209	Store the value of the current cell in the program to modify the save location
10	b008	Jump back to the start of the loop
12	c000	Halt execution

Question 5:

Add 13 numbers in adjacent memory matrix cells 28 to e8 (downwards again) and store the result in f8. This should be done in a maximum of 14 lines of code (can be done in fewer). Here I used the location of the active cell as the counter to save some lines of code.

Register	0	1	2	3	4	5	6	7
Purpose	Bit pattern f8		Counter / cell address	Bit pattern 10		Running total	Current number	

00	20f8	Load Register 0 with bit pattern f8 for checking counter
02	2228	Load Register 2 with bit pattern 28 for current cell address
04	2310	Load Register 3 with bit pattern 10 for increasing cell address / counter
06	2500	Load Register 5 with bit pattern 00 for running total
08	1628	Load Register 6 with the value at the current active cell
0a	5565	Add current number to running total
0c	5232	Add 10 to the counter / cell address
0e	B214	Jump to the end if the current value of the counter is e8
10	3209	Store the value for the current location into the program to modify it
12	b008	Jump back to the start of the loop
14	35f8	Store the total at address f8
16	c000	Halt execution

Brookshear Machine Instruction Set

Op-code	Operand	Description	Example
1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY.	14A3 would cause the contents of the memory cell located at address A3 to be placed in register 4.
2	RXY	LOAD the register R with the bit pattern XY.	20A3 would cause the value A3 to be placed in register 0
3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY.	35B1 would cause the contents of register 5 to be placed in the memory cell whose address is B1.
4	ORS	MOVE the bit pattern found in register R to register S.	40A4 would cause the contents of register A to be copied into register 4
5	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R.	5726 would cause the binary values in registers 2 and 6 to be added and the sum placed in register 7.
6	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R	634E would cause the values in registers 4 and E to be added as floating-point values and the result to be placed in register 3.
7	RST	OR the bit patterns in registers S and T and place the result in register R	7CB4 would cause the result of ORing the contents of registers B and 4 to be placed in register C
8	RST	AND the bit patterns in registers S and T and place the result in register R.	8045 would cause the result of ANDing the contents of registers 4 and 5 to be placed in register 0.
9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R.	95F3 would cause the result of EXCLUSIVE ORing the contents of registers F and 3 to be placed in register 5.

A	RoX	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end.	A403 would cause the contents of register 4 to be rotated 3 bits to the right in a circular fashion.
B	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.)	B43C would first compare the contents of register 4 with the contents of register 0. If the two were equal, the pattern 3C would be placed in the program counter so that the next instruction executed would be the one located at that memory address. Otherwise, nothing would be done and program execution would continue in its normal sequence.
C	000	HALT execution.	C000 would cause program execution to stop.

16 registers labelled 0 – F. Each hold 8 bits.

256 memory locations labelled 0 – FF. Each hold 8 bits.

All programs start at memory address 00.

All data starts at memory address a0.