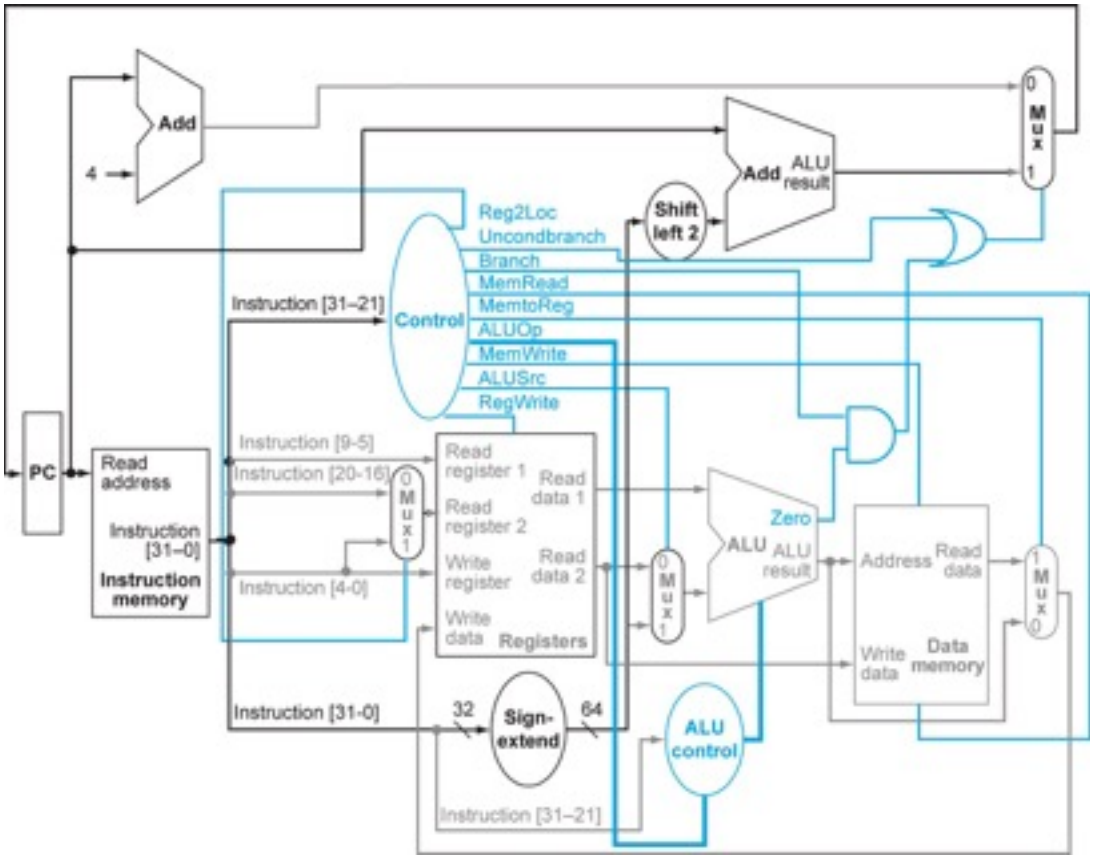
ELEC 3275 - Computer Architecture
Assignment 6 - ARM Decoder

You can have a partner (2 people per team). The ARMv8 control and decoder is used to set up the rest of the datapath based strictly on the ARMv8 instruction. R-Type instructions read 2 operands, write 1, use the ALU, etc. I-Type need an immediate. Branches use a particular set of instructions. Based on the ARMv8 opcodes, the control unit sets the rest of the hardware. Refer to the book and the lecture notes



for more information.

Your task is to create the decoder/control unit using C/C++ code. You need to set the following control signals, given the input of 11 bits. Note that of the 11 bits, most are used for an opcode, but for some operations, it uses less. It will be up to you to determine how to handle this issue. I strongly encourage you to group instructions by type and look for patterns.

- Reg2Loc: the multiplexer that feeds the read register; can be 0 or 1.
- Uncondbranch: feeds the OR which feeds the branch MUX on the top right; can be 0 or 1
- Branch: feeds the AND->OR which feeds the branch MUX on the top right; can be 0 or 1
- MemRead: tells the data memory to be read; can be 0 or 1;
- MemWrite: tells the data memory to write; can be 0 or 1; can't be 1 when MemRead is 1.

- MemToReg: the multiplexer that determines if the ALU result or the data memory result goes to the write destination; can be 0 or 1.
- ALUSrc: the MUX that feeds the ALU's second operand (register output or immediate); can be 0 or 1.
- RegWrite: tells the registers if there is a value that needs to be written back to the registers; can be 0 or 1
- You may ignore ALUOp and ALU Control

Instruction	Opcode	Opcode Size	11-bit opcode range		Instruction Format
			Start	End	
B	000101	6	160	191	B - format
STURB	00111000000	11	448		D - format
LDURB	00111000010	11	450		D - format
B.cond	01010100	8	672	679	CB - format
ORRI	1011001000	10	712	713	I - format
EORI	1101001000	10	840	841	I - format
STURH	01111000000	11	960		D - format
LDURH	01111000010	11	962		D - format
AND	10001010000	11	1104		R - format
ADD	10001011000	11	1112		R - format
ADDI	1001000100	10	1160	1161	I - format
ANDI	1001001000	10	1168	1169	I - format
BL	100101	6	1184	1215	B - format
ORR	10101010000	11	1360		R - format
ADDS	10101011000	11	1368		R - format
ADDIS	1011000100	10	1416	1417	I - format
CBZ	10110100	8	1440	1447	CB - format
CBNZ	10110101	8	1448	1455	CB - format
STURW	10111000000	11	1472		D - format
LDURSW	10111000100	11	1476		D - format
STXR	11001000000	11	1600		D - format
LDXR	11001000010	11	1602		D - format
EOR	11101010000	11	1616		R - format
SUB	11001011000	11	1624		R - format
SUBI	1101000100	10	1672	1673	I - format
MOVZ	110100101	9	1684	1687	IM - format
LSR	11010011010	11	1690		R - format
LSL	11010011011	11	1691		R - format
BR	11010110000	11	1712		R - format
ANDS	11101010000	11	1872		R - format
SUBS	11101011000	11	1880		R - format
SUBIS	1111000100	10	1928	1929	I - format
ANDIS	1111001000	10	1936	1937	I - format
MOVK	111100101	9	1940	1943	IM - format
STUR	11111000000	11	1984		D - format
LDUR	11111000010	11	1986		D - format

Below are the instructions you need to handle. Note that you can handle them in binary or decimal. You do not need to test for each and every one, but I encourage you to test enough to feel confident. Your testing will be considered as part of the grade.

What to submit: You must submit a write-up describing your code and showing proof it works correctly, using screenshots. You can use the template as a guide. Provide analysis of all results and describe any issues you had. You must also submit your C/C++ code.