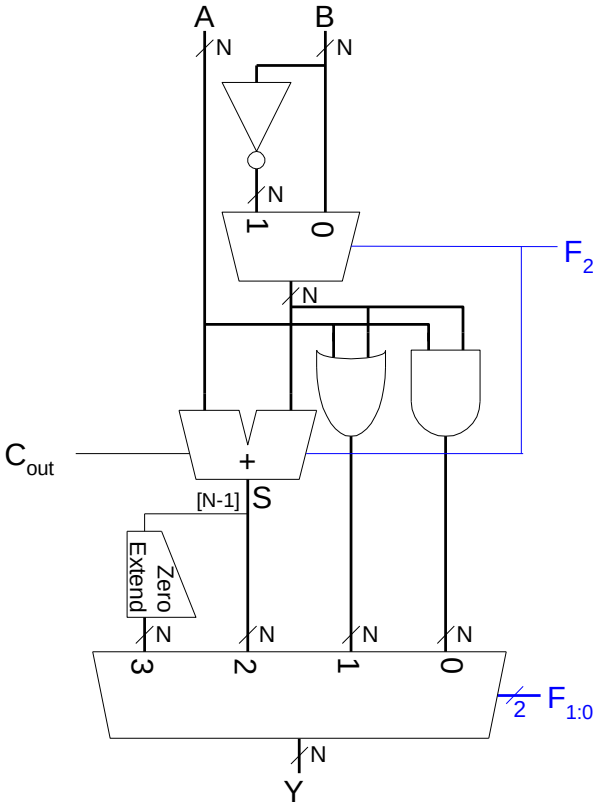
Hacettepe University	Department of Computer Engineering
BBM234 Computer Organization Spring 2021	Instructor: Prof. Dr. Suleyman TOSUN
Homework 1 Solutions	
Assigned date: 28.03.2021	**Submission deadline: 05.04.2021 at 13:59:59 via submit.cs.hacettepe.edu.tr**

Q1. Consider the ALU given below. This ALU has N-bit A and B inputs and 3-bit F input to control the operation mode of the ALU. It outputs N-bit output Y based on the selected operation. We would like to add more functionality to this ALU.

[25 pts] Add three 1-bit outputs to the ALU: *LT* (it is 1 when A is less than B), *EQ* (it is 1 when A is equal to B), and *GT* (it is 1 when A is greater than B).
Show your additional circuits on the ALU and explain them for full credit.



$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	not used
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Note that multiple solutions are possible. All correct implementations will be accepted.

SOLUTION 1 (*LT, GT, EQ irrespective of F*):

Consider all possible combinations of sign bits:

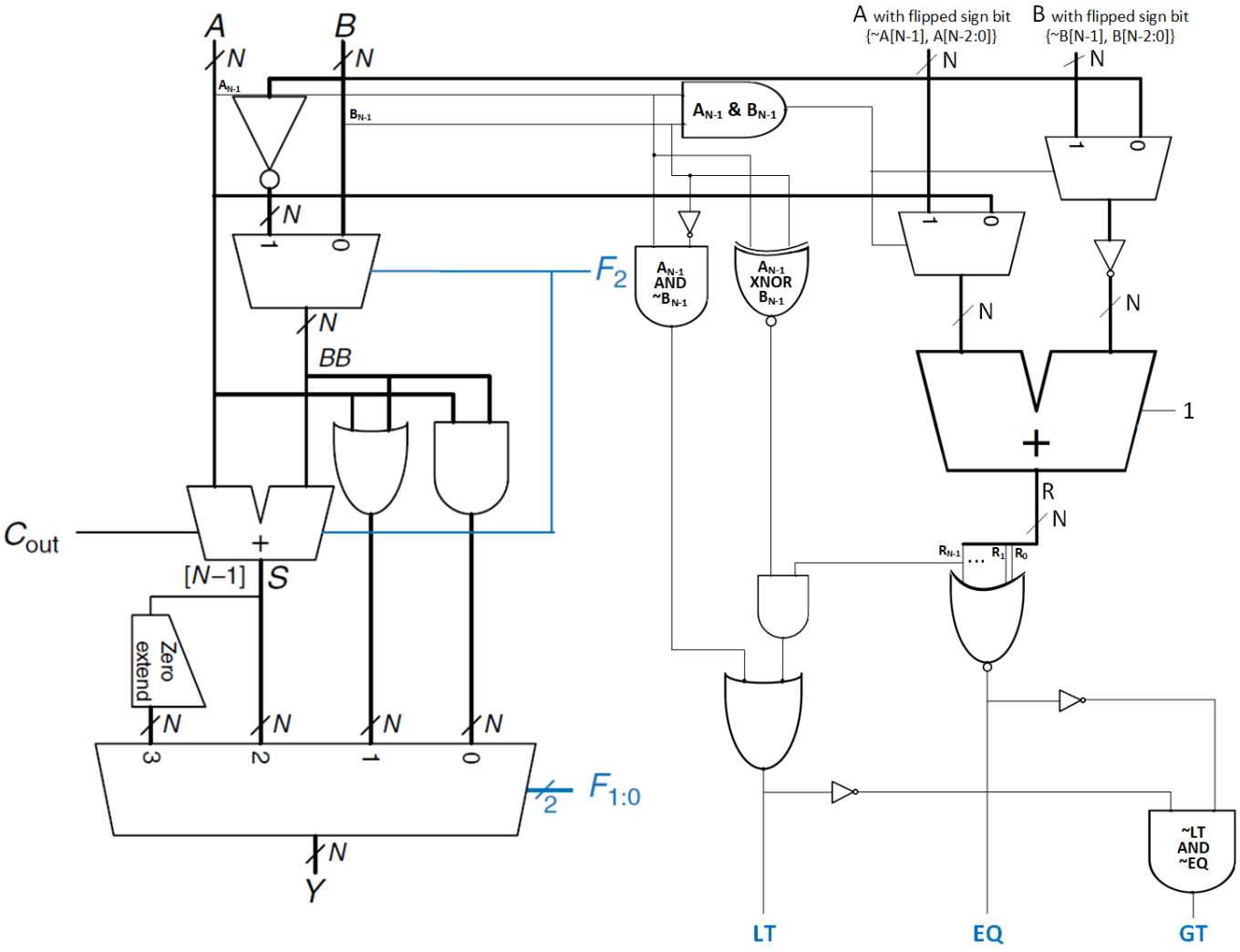
A_{N-1}	B_{N-1}	Interpretation	Solution
0	0	A≥0 AND B≥0	Use signed comparator
0	1	A≥0 AND B<0	A > B
1	0	A<0 AND B≥0	A < B
1	1	A<0 AND B<0	Use signed comparator with a twist

In case A and B are negative numbers, simply flip their most significant bit (flipping the bits has no effect). We can use a single comparator to perform both kinds of comparisons if we add a control signal S to tell the comparator whether to do unsigned (S=0) or 2's complement (S=1) comparison based on the most significant bits of A and B.

One way to reason about the solution if we implement the comparator using an adder to get A - B:

- **Case EQ = 1**: A - B = 0 (result from the comparator is zero)
  - `Verilog: assign EQ = ~|result;`
- **Case LT = 1**: (A_{N-1} = 1 AND B_{N-1}= 0) OR (A_{N-1} = B_{N-1} and A - B < 0)
  - `Verilog: assign LT = (A[N-1]&~B[N-1]) | (result[N-1] & (A[N-1]~^B[N-1]));`
- **Case GT = 1**: EQ = 0 AND LT = 0
  - `Verilog: assign GT = (~LT & ~EQ);`

A possible implementation using adder, MUX, and some logic gates:

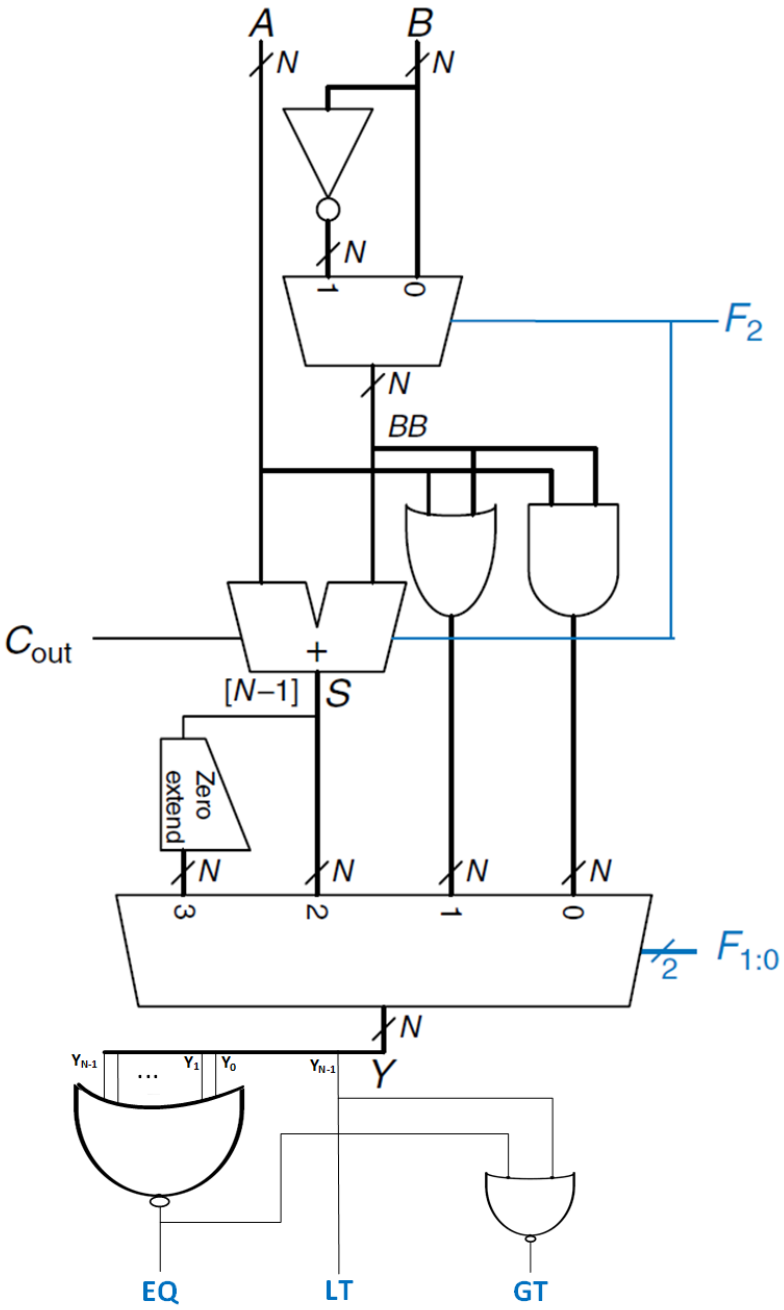


SOLUTION 2 (Taking $F_{2:0}$=110, so Y = A - B) - This solution will also be accepted:

When $F_{2:0}$=110, ALU will perform subtraction, so Y = A – B

- **Case EQ = 1**: Y = 0
  - `Verilog: assign EQ = ~|Y;`
- **Case LT = 1**: Y_{N-1} = 1, the most significant bit of Y is 1, the result is negative, so A < B
  - `Verilog: assign LT = Y[N-1];`
- **Case GT = 1**: EQ = 0 AND LT = 0
  - `Verilog: assign GT = (~LT & ~EQ); or assign GT = ~(LT | EQ);`

A possible implementation of solution 2 using some additional logic gates:



Q2.

(a) [9 pts] For 8-bit signed integer x = 0xA2, do the following calculations. Write your results in decimal.

x >> 4 = **10**

```
  10100010
Logical shift >>  4
  01010001
  00101000
  00010100
  00001010
```

x >>>4= **-6**

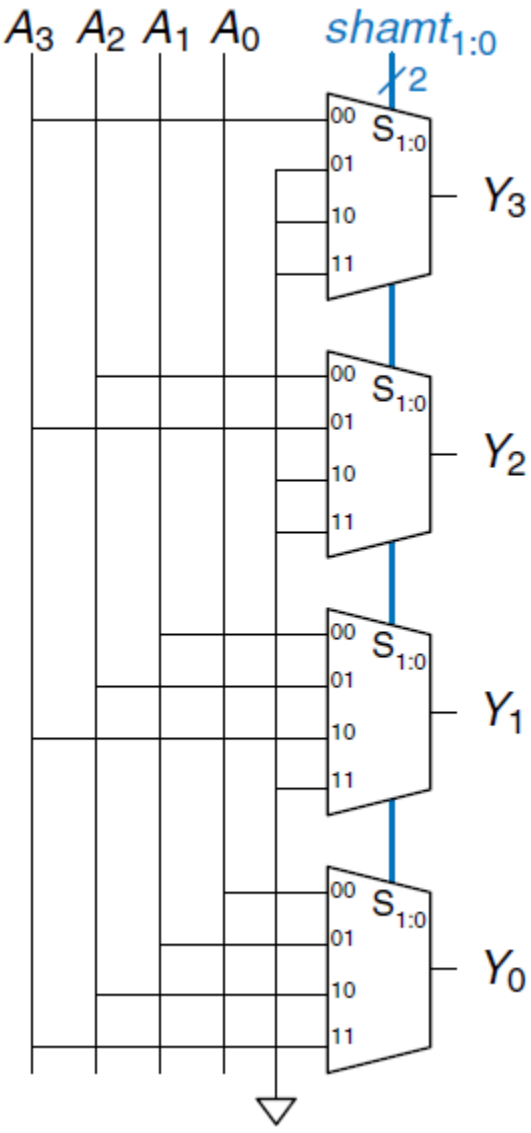
```
  10100010
Arithmetic shift >>>  4
  11010001
  11101000
  11110100
  11111010
```

x << 4 = **32**

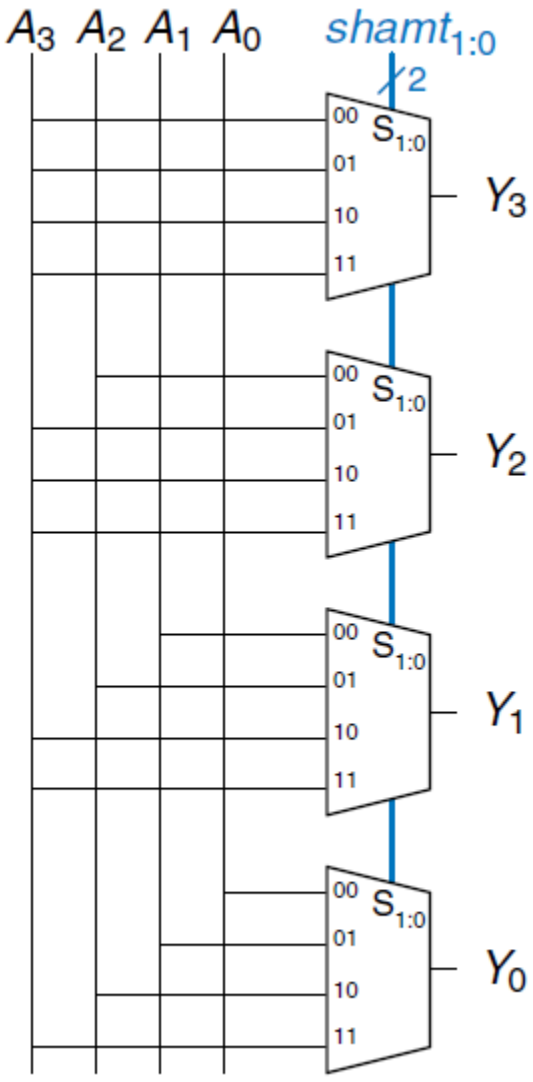
```
  10100010
Logical shift <<  4
  01000100
  10001000
  00010000
  00100000
```

(b) [16 pts] Below you will implement a logical and arithmetic shifter. The output Y will be the input A shifted by 0 to 3 bits depending on the value of the 2-bit shift amount $shamt_{1:0}$. For both shifters, when $shamt_{1:0}$= 00, Y=A.

Logical Right Shift



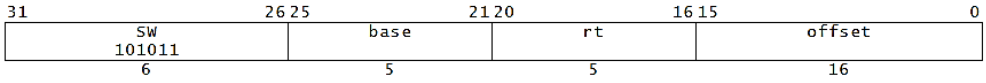
Arithmetic Right Shift



Q3. [25 pts] Write the machine code for the instructions given in bold. Indicate their instruction type, fill the binary machine code by showing the corresponding fields, and write their hexadecimal values into the boxes.

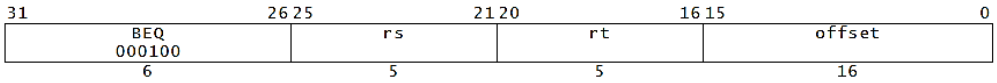
Address		Instruction	Opcode or funct	Register numbers
0x90	fact:	addi $sp, $sp, -8		
0x94		**sw $a0, 4($sp)**	#opcode: 0x2B	a0: 4, sp: 29
0x98		sw $ra, 0($sp)		
0x9C		addi $t0, $0, 2		
0xA0		slt $t0, $a0, $t0		
0xA4		**beq $t0, $0, else**	#opcode: 0x04	t0: 8
0xA8		addi $v0, $0, 1		
0xAC		addi $sp, $sp, 8		
0xB0		**jr $ra**	#funct: 0x08	ra: 31
0xB4	else:	addi $a0, $a0, -1		
0xB8		**jal fact**	#opcode: 0x03	
0xBC		lw $ra, 0($sp)		
0xC0		lw $a0, 4($sp)		
0xC4		addi $sp, $sp, 8		
0xC8		mul $v0, $a0, $v0		
0xCC		jr $ra		

`SW rt, offset(base)`



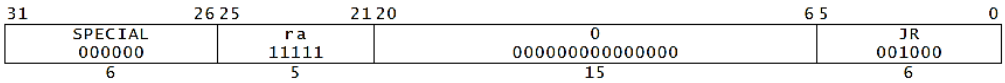
Instruction	Type	Binary Code	Hex code
sw $a0, 4($sp)	I-type	10101111101001000000000000000100	0xAFA40004

`BEQ rs, rt, offset`



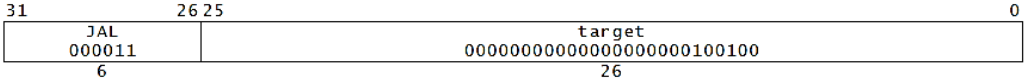
Instruction	Type	Binary Code	Hex code
beq $t0, $0, else	I-type	00010001000000000000000000000011	0x11000003

`JR rs`



Instruction	Type	Binary Code	Hex code
jr $ra	R-type	00000011111000000000000000001000	0x03E00008

`JAL target`



Instruction	Type	Binary Code	Hex code
jal fact	J-type	00001100000000000000000000100100	0x0C000024

Q4. [25 pts] You have four instructions stored in the memory as given in the following table:

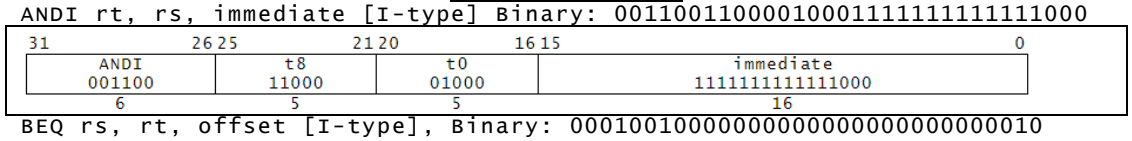
Instructions	Address	Instruction
Inst1	0x00400000	0x3308FFF8
Inst2	0x00400004	0x12000002
Inst3	0x00400008	0x01098020
Inst4	0x0040000C	0x08100001
Inst5	0x00400010	---

a) Write the binary values for each instruction. Clearly show which bits corresponds to which field in the instruction format (opcode, rs, rt, rd, etc.).

Instructions — Instruction format

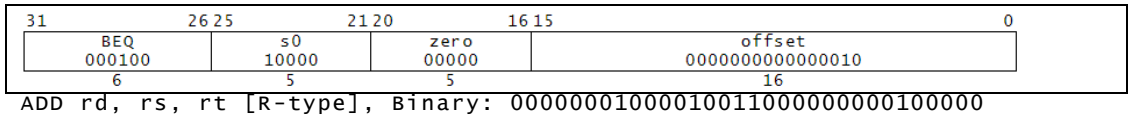
0x3308FFF8

ANDI rt, rs, immediate [I-type] Binary: 00110011000010001111111111111000



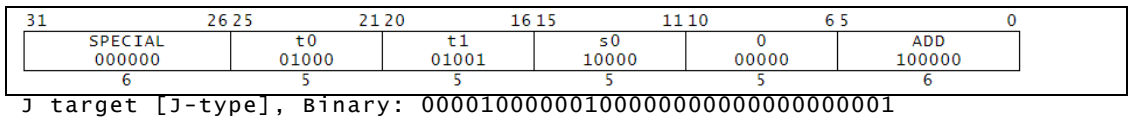
0x12000002

BEQ rs, rt, offset [I-type], Binary: 00010010000000000000000000000010



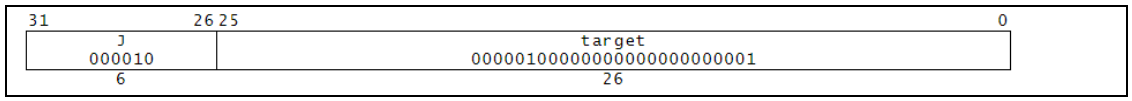
0x01098020

ADD rd, rs, rt [R-type], Binary: 00000001000010011000000000100000



0x08100001

J target [J-type], Binary: 00001000000100000000000000000001



b) Write down the corresponding MIPS assembly code below for each machine code.

Instructions	MIPS Code
Inst1	**andi $t0, $t8, -8 (0xfff8)**
Inst2	**Label: beq $s0, $0, Done**
Inst3	**add $s0, $t0, $t1**
Inst4	**j Label**
	Done:

Name	Register
$0	0
$at	1
$v0-$v1	2-3
$a0-$a3	4-7
$t0-$t7	8-15
$s0-$s7	16-23
$t8-$t9	24-25
$k0-$k1	26-27
$gp	28
$sp	29
$fp	30
$ra	31

Instruction	Opcode
j	000010
jal	000011
beq	000100
bne	000101
addi	001000
slti	001010
andi	001100
ori	001101
xori	001110
lui	001111
lw	100011
sw	101011

Instruction	Funct
sll	000000
srl	000010
sra	000011
jr	001000
div	011010
add	100000
sub	100010
and	100100
or	100101
xor	100110
nor	100111
slt	101011