For all Beta related questions, you should make use of the Beta documentation, the Beta Instruction Summary, and the Beta Diagram.

Beta ISA: 1

4/4 points (ungraded)

In this problem, you will consider a number of plausible hardware faults in an otherwise working Beta processor. Each of the faults involves changing a particular output of the control logic to some new (incorrect) constant value. In each case, you are to evaluate the impact of the fault on each of the following Beta instructions:

```
I1: ST(R0, 0x100, R1)
I2: JMP(LP, R31)
I3: BEQ(R31, .+4, R0)
I4: SUB(R1, R0, R0)
```

For each of the following faults, identify which (if any) of the above instructions will fail to work properly – that is, if the fault might effect the processor state (register and PC values) after the execution of the instruction. Be careful: some of these are tricky!

1. ALUFN stuck at code for "-" (32-bit SUBTRACT)
   Which instruction(s) fail? Select all that apply.
   - [x] I1
   - [ ] I2
   - [ ] I3
   - [ ] I4
   - [ ] None

   ✔
2. RA2SEL stuck at 1
   Which instruction(s) fail? Select all that apply.
   - [ ] I1
   - [ ] I2
   - [ ] I3
   - [ ] I4
   - [x] None

   ✔
3. WERF stuck at 0
   Which instruction(s) fail? Select all that apply.
   - [ ] I1
   - [ ] I2
   - [x] I3
   - [x] I4
   - [ ] None

   ✔
4. BSEL stuck at code 0
   Which instruction(s) fail? Select all that apply.
   - [x] I1
   - [ ] I2
   - [ ] I3

- [ ] I4
- [ ] None

✔

Submit

Beta ISA: 2

40/40 points (ungraded)
Marketing has asked for the following instructions to be added to an Extended Beta instruction set, for implementation on a Beta as implemented in Lecture and in the lab.

```
CLEAR2(Rx, Ry)          // Clear two registers
    Reg[Rx] ← 0
    Reg[Ry] ← 0
    PC ← PC + 4

NOR(Rx, Ry, Rz)         // Bitwise NOR: Rz[i] = NOR(Rx[i], Ry[i])
    Reg[Rz] ← bitwise NOR of Reg[Rx], Reg[Ry]
    PC ← PC + 4

LDRADR(C, Rx )          // Load Relative Address
    Reg[Rx] ← PC+4+4*SEXT(C)
    PC ← PC + 4

LDINCR(Rx,C, Ry )       // Load Incremented
    EA ← Reg[Rx]+SEXT(C)
    Reg[Ry] = Mem[EA] + 1
```

The Marketing people don't care about details of instruction coding (e.g., which fields are used to encode Rx and Ry in the above descriptions), but want to know which if any of the above can be implemented as a **single instruction** in the **existing Beta simply by changing the control ROM**.

Your job is to decide which of the above instructions can be implemented on the existing Beta, making appropriate choices for Rx and Ry, and to specify control signals that implement those instructions.

For each instruction select the appropriate values for the control signals in the table below. Select "None" for all entries in a row if the instruction cannot be implemented using the existing Beta datapath. Select "--" to indicate a "don't care" value for a control signal.

Instr	ALUFN	WERF	BSEL	WDSEL	MOE	MWR	RA2SEL
CLEAR2	NONE ✔	NONE ✔	NONE ✔	NONE ✔	NONE ✔	NONE ✔	NONE ✔
NOR	100001 ✔	1 ✔	0 ✔	1 ✔	- ✔	0 ✔	0 ✔
LDRADR	A ✔	1 ✔	- ✔	1 ✔	- ✔	0 ✔	- ✔
LDINCR	NONE ✔	NONE ✔	NONE ✔	NONE ✔	NONE ✔	NONE ✔	NONE ✔

Submit

Beta ISA: 3

1/1 point (ungraded)
You modify a working BETA by connecting the **JT** input to the PCSEL MUX to the constant 0, rather than to its proper logic. Which instructions, if any, are effected by this change?

Instruction(s) effected (select all that apply):

- [ ] BNE
- [x] JMP
- [ ] ADD
- [ ] ST

- [ ] None

✔

Submit

Beta ISA: 4

2/3 points (ungraded)
For the Beta instruction sequence shown below, indicate the values of the specified quantities after the sequence has been executed.

```
    .=0
    CMOVE(0x6000, SP)
    PUSH(SP)
    HALT()
```

Value left in SP (HEX): 0x

6004 ✔

Value pushed onto stack (HEX): 0x

6000 ✖

Value of WDSEL control signal during CMOVE execution: 0x:

1 ✔

Submit

Beta ISA: 5

11/11 points (ungraded)
Many problems, like sorting, require swapping data in two memory locations. The following assembly code swaps the contents of two words in memory, with addresses stored in R0 and R1:

```
LD(R0, 0, R2)
LD(R1, 0, R3)
ST(R2, 0, R1)
ST(R3, 0, R0)
```

1. Suppose we add a SWAP instruction to the Beta. SWAP swaps the contents of a register and a memory location:

```
SWAP(Ra, literal, Rc)       // Swap register contents with memory
    EA ← Reg[Ra] + SEXT(literal)
    tmp ← Mem[EA]
    Mem[EA] ← Reg[Rc]
    Reg[Rc] ← tmp
    PC ← PC + 4
```

Which of the following pieces of code is a valid rewrite of the code above?

- ( )
```
LD(R1, 0, R2)
SWAP(R1, 0, R2)
ST(R2, 0, R0)
```

- (•)
```
LD(R0, 0, R2)
SWAP(R1, 0, R2)
ST(R2, 0, R0)
```

```
LD(R0, 0, R2)
SWAP(R1, 0, R2)
ST(R2, 0, R1)
```

```
SWAP(R0, 0, R1)
```

✔

Explanation
LD(R0, 0, R2): load R2 with Mem[R0]
SWAP(R1, 0, R2): swaps the contents of R2 with the contents of Mem[R1] - so Mem[R1] now holds what was initially in Mem[R0]
ST(R2, 0, R0): stores R2 into Mem[R0] - so Mem[$0] now holds what was initially in Mem[R1]

2. Can we implement the SWAP instruction using the unpipelined Beta datapath, by simply changing the control ROM? Recall that, in the unpipelined datapath, the data memory has combinational reads, and writes are clocked, happening at the end of the cycle. Also assume that the memory still returns valid data when WR=1. Either fill in the control signals below, or select NONE for all cases if SWAP cannot be implemented.

Instr	ALUFN	WERF	BSEL	WDSEL	MOE	MWR	RA2SEL
SWAP	A+B	1	1	2	1	1	1
	✔ Answer: A+B	✔ Answer: 1	✔ Answer: 1	✔ Answer: 2	✔ Answer: 1	✔ Answer: 1	✔ Answer: 1

Explanation

- The EA is generated by the ALU. This means that the ALU must add Reg[Ra] to SEXT(literal). In order to do this, ALUFN = 'A+B', ASEL = 0, and BSEL = 1.
- In order to write the value of Reg[Rc] into this memory location, RA2SEL = 1, and MWR = 1.
- In order to read the value at Mem[EA] and store it into Reg[Rc], MOE = 1, WERF = 1, WASEL = 0, and WDSEL = 2.
- In order to increment PC by 4, PCSEL = 0.

Submit

Answers are displayed within the problem

Beta ISA: 6

4/4 points (ungraded)
A "stuck-at" fault happens when a signal is shorted to VDD or GND and is permanently a logic 1 or logic 0. For each of the following stuck-at faults there is a list of instruction opcodes – please select the opcodes whose execution might be affected by the indicated fault.

1. RA2SEL stuck-at logic "0"

- [ ] ADD
- [ ] ADDC
- [ ] LD
- [x] ST
- [ ] JMP
- [ ] BEQ
- [ ] BNE
- [ ] LDR
- [ ] Illop

✔

2. BSEL stuck-at logic "1"

- [x] ADD
- [ ] ADDC
- [ ] LD
- [ ] ST
- [ ] JMP
- [ ] BEQ
- [ ] BNE
- [ ] LDR
- [ ] Illop

✔

3. WDSEL[1] (high-order bit of WDSEL mux select) stuck-at logic “0”

- [ ] ADD
- [ ] ADDC
- [x] LD
- [ ] ST
- [ ] JMP
- [ ] BEQ
- [ ] BNE
- [x] LDR
- [ ] Illop

✔

4. WERF stuck-at logic “1”

- [ ] ADD
- [ ] ADDC
- [ ] LD
- [x] ST
- [ ] JMP
- [ ] BEQ
- [ ] BNE
- [ ] LDR
- [ ] Illop

Submit

✔ Correct (4/4 points)

Beta ISA: 7

4/6 points (ungraded)
Your summer internship involves adding new instructions to the Beta processor. You're given a list of proposed new Beta instructions, each of which is to perform a given operation during the **single clock cycle** in which the instruction executes. You decide to sort the proposals into four classes:

- **Macro**: those instructions that can be implemented on a stock Beta, simply by defining an appropriate macro;
- **CTL**: those that can be implemented on a stock Beta, by defining an appropriate macro and making appropriate changes to the control ROM;
- **HW**: those instructions that require hardware changes beyond reprogramming the control ROM.
- **None**: The instruction as described can't be implemented, even with hardware changes.

For each of the following descriptions, identify which of the above categories the instruction falls into.

1. **ADD37(r)** which adds 37 to the contents of specified register r.
   - (•) Macro
   - ( ) CTL
   - ( ) Hardware
   - ( ) None

   ✔

   Explanation
   ADD37(r) can be implemented as a macro that calls ADDC with the contant 37.

2. **ZERO2(rx,ry)** which sets the contents of both registers rx and ry to zero.
   - ( ) Macro
   - ( ) CTL
   - (•) Hardware
   - ( ) None

   ✔

   Explanation
   In order to be able to write to two different registers in one clock cycle, the hardware of the Beta would need to be modified.

3. **GETPC(rx)** which sets the contents of register rx to the address of the following instruction.
   - ( ) Macro ✔
   - (•) CTL
   - ( ) Hardware
   - ( ) None

   ✖

   Explanation
   GETPC(rx) could be implemented as a macro that calls a branch that is never taken, like BNE(R31,0,rx)

4. **GETADR(loc, rx)** which sets the contents of register rx to the address of a nearby location tagged loc.