Building the Beta

Problem 1.

Consider the following potential additions to the Beta Instruction set:

```
// Swap register contents with memory location
MSWP(Ra,literal,Rc):
        EA ← Reg[Ra] + SXT(literal)
        tmp ← Mem[EA]
        Mem[EA] ← Reg[Rc]
        Reg[Rc] ← tmp

// Move if zero
MVZ(Ra,Rb,Rc):
        if (Reg[Ra] = 0)
            Reg[Rc] ← Reg[Rb]

// Move constant if zero
MVZC(Ra,literal,Rc):
        if (Reg[Ra] = 0)
            Reg[Rc] ← SXT(literal)
```

a) Specify the control signals configurations needed to execute these instructions on a Beta.

	MSWP	MVZ	MVZC
ALUFN	"+"	"B"	"B"
WERF	1	Z?1:0	Z?1:0
BSEL	1	0	1
WDSEL	2	1	1
WR	1	0	0
RA2SEL	1	0	-
PCSEL	0	0	0
ASEL	0	-	-
WASEL	0	0	0

b) Explain why the following instruction cannot be added to our Beta instruction set without further modifications:

```
// Push Rc onto stack pointed to by Ra
PUSH(Rc, 4, Ra):
        Mem[Reg[Ra]] ← Reg[Rc]
        Reg[Ra] ← Reg[Ra] + 4
```

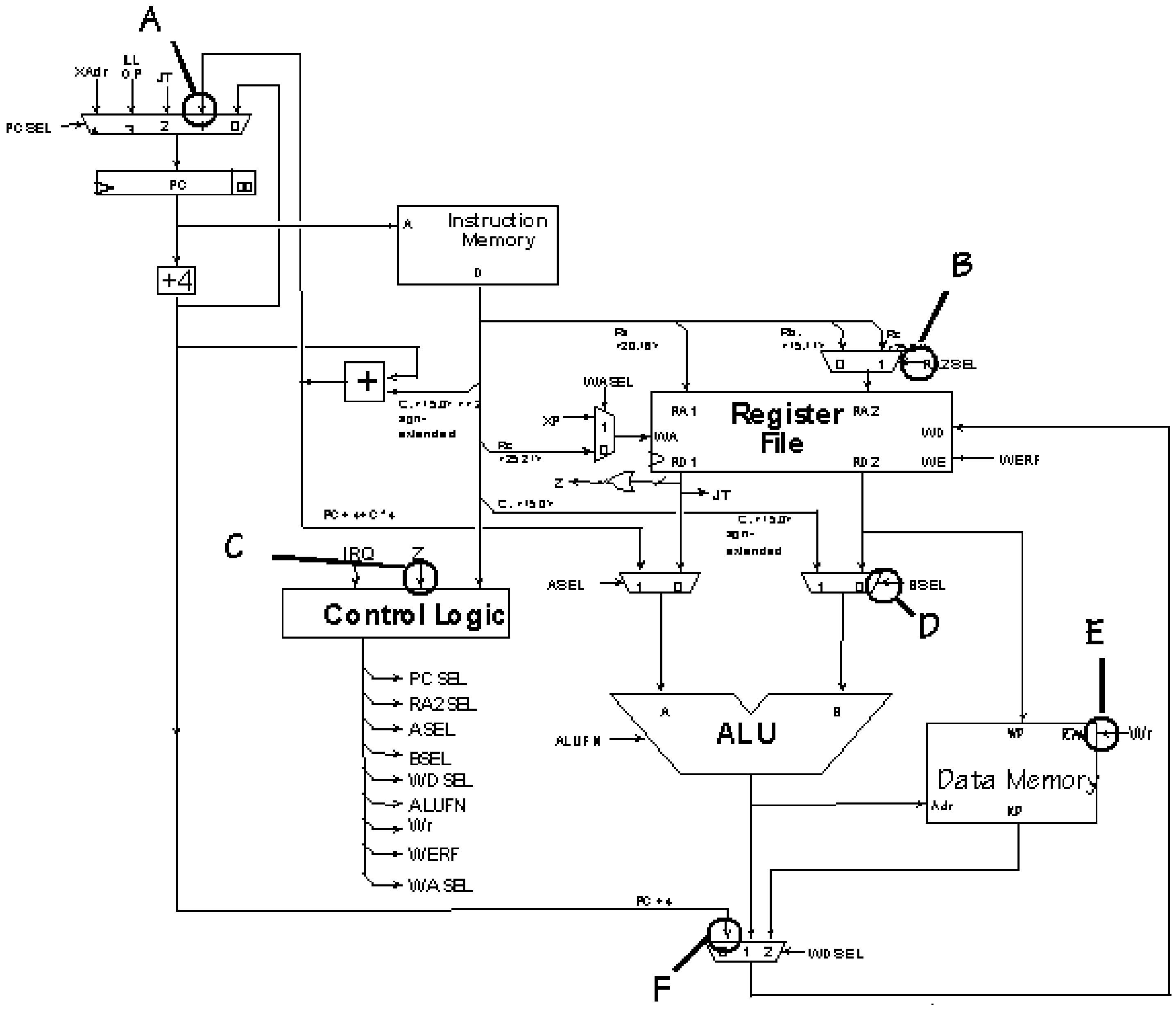
To implement this PUSH, the ALU would have to produce two values: Reg[Ra] to be used as the memory address and Reg[Ra]+4 to be written into the register file.

Problem 2.

One Beta manufacturer is having quality-control problems with their design. In particular, they've had reliability issues with various device connections that are circled in the attached diagram.

Your job is to write some test programs to help determine if a machine is fault-free. Assume that when a device connection is "faulty," the indicated bus or signal is always "stuck-at 0" instead of the expected value. For each of the circled connections, write an instruction sequence that when executed for a specified number of cycles would indicate whether the connection was okay by leaving a "1" in R0 and leaves some other value in R0 if the connection was faulty. You can assume that all registers are reliably set to 0 before each sequence is executed.

Give your instruction sequence for each of the six indicated faults and briefly explain how each sequence detects the fault and produces something besides "1" in R0 when the fault is present.



a) Fault A: Input 1 of PCSEL mux stuck at 0.
(Note that not PCSEL is stuck at 0, but the signal at input 1 is always 0)

```
. = 0
BEQ(R0,.+4,R31)        0x0
ADDC(R0,1,R0)          0x4
```

Execute for 2 cycles (i.e., execute two instructions). If fault A is not present, R0 contains 1 after the second cycle, since the second instruction is fetched from location 4. If fault A is present, the second instruction is fetched from location 0 (instead of 4, since the input 1 to the PCSEL mux is 0), so the value of R0 stays 0.
Note that the label ".+4" means "memory location of current instruction + 4", which is "0+4" here.

Also note that there is no unique solution on how to implement these tests, you may think of other alternatives.

b) Fault B: RA2SEL mux control signal stuck at 0.

```
. = 0
ADDC(R1,1,R1)
ST(R1,0,R0)
LD(R0,0,R0)
```

Execute for 3 cycles. If fault B is not present, the ST instruction writes the value 1 into location 0, which is then LDed into R0. If fault B is present, the ST instruction writes the contents of R0 instead (ie, the value 0), so now the LD instruction puts 0 into R0.

c) Fault C: Z input to control logic stuck at 0.

```
. = 0
BEQ(R0,.+8,R31)
ADDC(R0,0,R0)
ADDC(R0,1,R0)
```

Execute for 2 cycles. If fault C is not present, R0 is incremented to 1 since the branch to memory location 8 is taken. If fault C is present, the BEQ instruction never branches, executing the instruction at location 4, which leaves the contents of R0 unchanged (i.e., it's still 0).

d) Fault D: BSEL mux control signal stuck at 0.

```
        . = 0
        ADDC(R0,1,R0)
```

Execute for 1 cycle. If fault D is not present, R0 is increment to 1. If fault D is present, the high-order 5-bits of the literal field (i.e., where Rb is encoded) is used as a register address, and the contents of that register is added to R0. Since the literal is "1", the second register is R0 (containing 0), so the value written into R0 is 0.

e) Fault E: WR memory control signal stuck at 0.

```
        . = 0
        ADDC(R1,1,R1)
        ST(R1,X,R31)
        LD(R31,X,R0)

        . = 0x100
X:      LONG(0)
```

Execute for 3 cycles. If fault E is not present, the ST instruction writes the value 1 into X, which is then LDed into R0. If fault B is present, the ST instruction has no effect, so now the LD instruction loads the original value of location X into R0.

f) Fault F: Input 0 of WDSEL mux stuck at 0.

```
        . = 0
        BEQ(R0,.+4,R1)
        SUBC(R1,3,R0)
```

Execute for 2 cycles. If fault F is not present, the BEQ instruction loads 4 into R1 and the SUBC loads 1 into R0. If fault F is present, the BEQ instruction loads 4 (= PC + 4) into R1 and the SUBC loads 8 (= PC+4) into R0.

Shorter alternative:

```
        . = 0
        ADDC(R0,1,R0)
```

Execute for 1 cycle. If fault F is not present, the ADDC instruction will write 1 into R0, otherwise, 4 (= PC+4) will be written into R0.