- For an out-of-order superscalar processor, what are false dependencies on register names and what hardware technique is often used to remove them?
- Do you think exceptions cause more of a performance penalty in out-of-order or in-order processors and why?
- Are there any limitations of VLIW? Illustrate with examples?
- Modern superscalar processors are able to support hundreds of instructions "in-flight" at the same time and schedule instructions dynamically (i.e. support out-of-order execution). What advantages does dynamic scheduling offer when compared to an in-order superscalar processor?

- Vector instructions extensions are added to a small 32-bit microcontroller. The vector length is 128-bits. The register bank in the processor's floating-point unit (32 x 32-bit single-precision registers) is reused for vector processing and eight 128-bit vector registers alias onto it. The processor can only issue a single instruction per cycle. It has a 32-bit wide memory datapath and a single 32-bit multiplier. What is the advantage of allowing many vector instructions to be able to access both vector registers and registers in the scalar register file? Describe one way in which a vector instruction-set extension may efficiently handle cases where the number of elements we wish to process is not a precise multiple of the maximum vector length supported in hardware?

- Identify all RAW, WAW, WAR dependencies in the loop shown below. Write down the dependencies within a single iteration only.

```
                INSTRUCTION
        LOOP:
                // upon entry into loop,
                // F0 = a (constant), F3 = 0, R1 = 0
I1              L.D   F1, 0(R1)    ;load X(i)
I2              MUL.D F2, F1, F0   ;multiply a*X(i)
I3              ADD.D F3, F3, F2   ;add a*X(i) to F3
I4              MUL.D F2, F1, F1   ;multiply X(i)*X(i)
I5              S.D   F2, 0(R1)    ;store to memory
I6              ADDI  R1, R1, 8
I7              SGTI  R2, R1, 800
I8              BEQZ  R2, LOOP
```

- For the code given below, Using a branch history table with 2-bit saturating counters, **exactly how many mispredictions will there be?** Assume there are enough table entries to avoid conflicts. Assume the table entries are initially 0:

Consider the following C code:

```
int code (void) {
        int i, j;
        int c = 0;
        i = 1;
loop:   j = 1;
loop2:  c = c + i + j;
        j++;
        if (j <= 7) goto loop2;
        i++;
        if (i <= 1000) goto loop;
        return c;
}
```

- Given a code sequence as shown below, we would like to trace its behavior in a custom machine

```
loop:
(1)add   x1, x1, x2
(2)sw    x5, (x2)
(3)addi  x6, x6, x6
(4)beq   x1, x2, loop
(5)sw    x7, (x2)
(6)fadd  f8, f8, f8
(7)beq   x8, x7, loop
(8)lw    x9, (x2)
```

- We use a processor that has the following ROB structure, and issues one instruction per cycle. Instructions get written into the ROB at decode time (at the end of the decode cycle the instruction is in the ROB). Instructions write their output in the physical register file when the data is generated, and other instructions directly get that data from the physical register file when needed. The ROB does not contain a “data” field. Stores calculate their address as an integer operation. Branches are also considered integer operations and take two cycles to execute. You can assume integer operations take two cycles to execute, and floating point four cycles. You can also assume one integer and one floating point functional unit, both fully pipelined that latch their operands on their first pipeline stage. “Use” means that that corresponding ROB entry is valid and “exec” means the instruction is executing. “Pd” is 1 when the instruction completes.

Ins #	Use?	Exec?	Op	P1	Src1	P2	Src2	Pd	Dest

- Without branch prediction, how many cycles penalty do we incur? Next let's assume that we design a simple predictor that always predicts that a branch is not taken. Show the state of the ROB as soon as the first branch of our code (number 4) completes and the processor realizes that it mis-speculated, and what it should do to recover?