EECS 470, Winter 2024, Homework 2 answers.

1)

a) Main ones: There is a free reservation station. (You might also want to note that the instruction needs to have been fetched.)
b) Main ones: There is a free reservation station (not needed if mentioned above); any branch in front of the instruction has been completed.
c) Main ones: All of its arguments are ready, an execution unit is available, and the instruction gets selected (others might also be wanting the same execution unit).
d) Main ones: Execution is completed, CDB is available.

2)

Consider 100 instructions. Of those 100 instructions, 11 will be mispredicted Branches (20%*55%*100). There is no reason to suspect bias in the distribution of branches to which pipeline the data is in so half[1] will squash 6 instructions, half will squash 7. Thus, you have 11*6.5 = 71.5 "instructions" that are squashed, plus 100 slots utilized to execute the 100 instructions. ~171.5 slots take up 171.5/2 = 85.75 cycles. So, CPI= ~0.8575 and IPC= ~1.166.

3a)

```
P1=MEM[R2+0]
P2=MEM[R2+4]
Branch done if(P2==0)
P3=P1+R3
Branch start
P4=MEM[P2+0]
P5=MEM[P2+4]
Branch done if(P5==0)
P6=P4+P3
Branch start
P7=MEM[P5+0]
P8=MEM[P5+4]
Branch done if(P8==0)
Halt
```

3b)

It appears to be walking a linked list which includes a data element (MEM[R2+0]) as well as a pointer to the next element in the list. It sums up all the data elements but the last one.

4)

The gselect predictor uses a concatenation of bits from the branch PC and global history register as the index. Both bimodal and global history predictors are degenerate cases because gselect is bimodal when we use all the bits of PC and 0 bits from global history, and gselect is a global history predictor when we use all bits of global history register and 0 from PC. So, both bimodal and global history are manifestations of gselect. Gselect-

[1] Okay, there is a very slightly higher chance of being in the first, but we'll ignore it.

best is the combination of PC and global history bits that gives the best prediction rates. The authors in the paper run a set of simulations to find out the best combination of PC and global history bits. Since Gselect-best uses the best combination of PC and global history bits, it will perform at least as well as a bimodal or global history predictor.

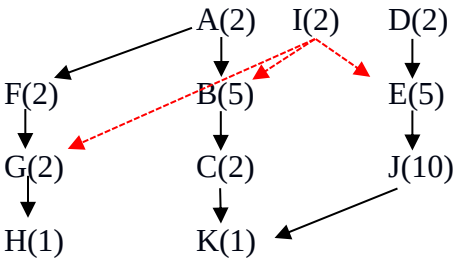
5)

a) 100% (no aliasing in the pattern)
b) 60% (mispredicts twice very loop since the PHT is only 1 bit. So 2/5 are predicted incorrectly)
c) 66% (still mispredicts twice every loop, but now there are 6 predictions per loop, so 2/6 are predicted incorrectly).

6)

Instruction	**FD**	**X**	**W**
`A: LD 0(R1),F2`	1	3	4
`B: MULTD F0,F2,F4`	4	9	10
`C: ADDD F2,F4,F6`	10	12	13
`D: LD 8(R1),F4`	13	15	16
`E: MULTD F0,F4,F10`	16	21	22
`F: ADDD F2,F2,F2`	22	24	25
`G: ADDD F0,F2,F2`	25	27	28
`H: SD 8(R1),F2`	28	29	30
`I: ADDDI F0, #16, F0`	30	32	33
`J: DIVDI F10, #3, F12`	33	43	44
`K: BNEQ F6,F12,A`	44	45	46

7a) Note: It is okay to omit the red arrows which indicate dependencies that go across loops. Putting I(2) by itself with no dependents is still correct.



7b)

Instruction	**Has all data**	**Start EX**	**End EX**	**Writeback**
`A: LD 0(R1),F2`	1	2	3	4
`D: LD 8(R1),F4`	1	2	3	4
`I: ADDDI F0, #16, F0`	1	2	3	4

`B: MULTD F0,F2,F4`	4	5	9	10
`E: MULTD F0,F4,F10`	4	5	9	10
`F: ADDD F2,F2,F2`	4	5	6	7
`G: ADDD F0,F2,F2`	7	8	9	10
`C: ADDD F2,F4,F6`	10	11	12	13
`H: SD 8(R1),F2`	10	11	11	12
`J: DIVDI F10, #3, F12`	10	11	20	21
`K: BNEQ F6,F12,A`	21	22	22	23

8)

a) First note that another (and better) way to do the same thing is to not let any instruction following a branch issues into the RSes until there is no unresolved branch in the RSes.

But either way, the issue is that the original Tomasulo's algorithm had no way of preventing an instruction younger than the branch from modifying architectural state (e.g a general-purpose register). If the branch were mispredicted, we'd have no clear way to undo that change.

b) Every branch instruction will stall following instructions and limit the number of instructions in the pipeline. Even if other instructions could start executing, they are not able to.

c) An exception is an uncommon or undesirable execution case which the program cannot handle and must control to some software handler to deal with it. Some examples of load exceptions include:
   a. Invalid address (ex: misaligned address)
   b. Permission error (current process is not allowed to access address)
   c. Page fault (need to go to disk to retrieve the necessary page)

d) If an exception in a load occurs, the code is going to end up branching to some handler. So the load is, in effect, a branch when an exception occurs.

e) A reorder buffer tracks the ordering of instructions and only commits them to the physical state once it knows that it is safe to do so. In the case of an exception, you can clear every instruction following the exception-causing one so that you do not execute anything you should not have. Since you have the ability to discard results before they are permanently committed to the architected state, you can start execution before the branch or load finishes.