1.
   a.)
      Sign-Extend Output: 00000000000000000000000000101000
      Jump: 00011000100000000000000001010000
   b.)
      ALU control unit: 00
      Instruction: 010100
   c.)
      New Address: PC + 4
      Path: PC->ADD(PC+4)->Branch Mux->Jump Mux->PC
   d.)
      ALU: 20
      MEM: x
      Jump: PC+4
      Branch:PC+4
      WrReg:0
   e.)
      ALU: 20 & -3
      ADD(PC+4): PC & 4
      ADD(Branch): PC+4 & 20*4
   f.)
      Reg1:3
      Reg2:2
      RegWrite:0
      WriteReg:0
      WriteData:x
2.
   a.)
      Non-Pipelined Processor: 1250ps
      Pipelined Processor: 350ps
   b.)
      Non-Pipelined Processor: 1250ps
      Pipelined Processor: 1750ps
   c.)
      We always want to split the longest stage, which would be ID, New speeds are:
      Pipelined Processor: 1500ps
   d.)
      Data Memory Being Used Only By LW & SW: 35%
   e.)
      Data Memory Being Used Only By LW & ALU: 65%
   f.)
      Multi-Cycle Organization = Pipelined Organization (in terms of speed)
      Single-Cycle = Pipelined (in terms of speed)

To get how fast Multi-Cycle is:
(5 x .2) + (4 x (.45 + .20 + .15)) = 4.2
So, Multi-Cycle is 4.2x faster than Pipeline.
To get how fast Single-Cycle is:
Cycle time(Non-Pipelined) / Cycle time(Pipelined)
1250 / 350 = 3.5
So, Single-Cycle is 3.5x faster than Pipeline

3.

a.)
- Read after Write on r1 from the 1st instruction to the other two instructions
- Read after Write on r2 from 2nd instruction to the 3rd instruction
- Write after Read on r2 from 1st instruction to the 3rd instruction
- Write after Read on r1 from 2nd instruction to 3rd instruction
- Write after Write on r1 from 1st instruction to 3rd instruction

b.)
There is dependency between all three instructions and assuming that will cause some harm, then we have to add two NOP between the instructions in order for there to be no errors so the result will be:

```
or r1,r2,r3
NOP
NOP
or r2,r1,r4
NOP
NOP
or r1,r1,r2
```

c.)
Since there is only ALU instructions there will be no hazards

d.)
Without Forwarding: (7+4) x 250ps = 2750ps
With Forwarding: (7+0) x 300ps = 2100ps
Speed-up: w/o Forwarding / w/ Forwarding = 2750/2100 = 1.31

e.)
After the second instruction it cannot go forward because it will move from memory to execute so you have to add NOP there:

```
or r1,r2,r3
or r2,r1,r4
NOP
NOP
or r1,r1,r2
```

f.)
Without Forwarding: (7+4) x 250ps = 2750ps
ALU-ALU Forwarding: (7+2) x 290ps = 2610ps
Speed-up: w/o Forwarding / w/ ALU-ALU Forwarding = 2750/2610 = 1.054

4.

a.)

Since during the IF stage the ADD and SLT instruction will access the memory at the same time we need to wait 2 instructions to make sure this doesn't happen so the total execution time is 11 clock cycles.

Yes you can add NOPS to avoid this because as long as you delay the instructions ADD and SLT you can avoid this hazard.

b.)

New Code:

sw r16,r6

lw r16,r7

beq r5,r4,lb1

add r5,r1,r4

slt r5,r15,r4

Before it was 9 and now it is 8 cycles so the speed-up is: 9/8 = 1.13

c.)

Originally the cycle would take 11, but using stall-on-branch it will reduce it by 1 cycle, so the speed-up will now be: 11/10 = 1.1

d.)

Using the results of part b we had 9-cycles / 8-cycles, but with EX/MEM in a single cycle or can be done in parallel so the new speed up will be: 9(200) / 8(210) = 1800 / 1680 = 1.07

e.)

In part c we had 11-cycles / 10-cycles,but we are now taking into account the change in clock cycle time assuming the ID stage increases by 50% and EX stage decreases by 10ps when branch outcome resolution is moved from EX to ID resulting in a speed-up of: 11(200) / 10(200) = 1.1

This is due to the highest latency still being 200 so there is no change in speed-up.

f.)

If EX is reduced, then we get 130ps, but the cycle time doesn't change at all so we still have a cycle time of 11 cycles. Having an execution time of 2200ps. Since The branch of MEM would be increased by one, we now get a cycle time of 12 cycles. Having now an execution time of 2400ps. Calculating for the speed-up we get: 2200ps/2400 = 0.92