1. **Answer to Problem 1**

```assembly
.globl main
main:   add     $s7, $0, $ra    # save return address
.data
A:.word  4, 9, 25, 49, 121, 169, 289  # squares of first 7 primes
.text
################# Answer to Problem 2 ##########################
la$s1, A          # start address of A
addi    $s2, $0, 0      # running sum
addi    $s3, $0, 0      # array index of A
addi    $s4, $0, 7      # constant 7
Loop:  lw      $t1, 0($s1)     # $t1 = A[$s3]
add     $s2, $s2, $t1   # $s2 = sum of A[] so far
addi    $s1, $s1, 4     # $s1 += 4
addi    $s3, $s3, 1     # $s3 += 1
bne     $s3, $s4, Loop  # branch back to Loop until $s4 == 7
addi    $v0, $0, 1      # print the sum
add     $a0, $0, $s2
syscall
################# End of Answer to Problem 2 ###################
addi    $v0, $0, 4      # print a newline
la      $a0, Newln
syscall
add     $ra, $0, $s7    # restore return address
jr      $ra
.data
Newln:  .asciiz "\n"
################# Output #######################################
# ten42% spim -file exam1_2.s
# 666
################# End of output ################################
```

Another answer:

```assembly
.globl main
main:   add     $s7, $0, $ra    # save return address
.data
A:.word  4, 9, 25, 49, 121, 169, 289  # squares of first 7 primes
.text
################# Second Answer to Problem 2 ###################
la$s1, A          # start address of A
addi    $s2, $0, 0      # running sum
addi    $s3, $0, 0      # array index of A
addi    $s4, $0, 7      # constant 7
Loop:  mul     $t0, $s3, 4     # $t0 = array index * 4
add     $t2, $t0, $s1   # $t2 = start of A + offset
lw      $t1, 0($t2)     # $t1 = contents at start of A + offset
add     $s2, $s2, $t1   # $s2 = sum of A[] so far
addi    $s3, $s3, 1     # $s3 += 1
bne     $s3, $s4, Loop  # branch back to Loop until $s4 == 7
addi    $v0, $0, 1      # print the sum
add     $a0, $0, $s2
syscall
################# End of Second Answer to Problem 2 ############
```

2. **Answer to Problem 2**

```assembly
.globl main
main:   addu    $s7, $zero, $ra    # save return address
########## MAIN FOR PROB 2 #################
addi    $a0, $0, 12     # Param = 12
jal     F               # call F
add     $a0, $0, $v0    # $v0 = ret val
li      $v0, 1          # print it
syscall
jal     Newl            # print newline
# Finish main
addu    $ra, $zero, $s7 # normal end of main
jr      $ra             # return to system
########## END OF MAIN ####################
F:
add$v0, $a0, $a0
jr$ra
########## END OF FUNCTION F ##########
########## write newline ################
Newl:
li      $v0, 4
la      $a0, Newline
syscall
jr      $ra
########## DATE #########################
data
Newline: .asciiz "\n"
#########################################
# Output:
#   24
#########################################
```

3. Just the standard beq diagram for the singlecycle implementation.

4. Just the standard lw diagram for the singlecycle implementation, with FIVE cycles.

5. In the upper diagram, register 2 is forwarded from the sub and register 4 for the and instructions. In the lower diagram, register 4 would be forwarded from the and, except that register 4 is also the target of the or instruction, so it is forwarded from there.

6. A stall is needed after the beq instruction. See Section 6.6.

7. (a) Single-error correction and double error correction.

8. (a) Overflow and undefined instruction.
(b) The instruction after the one causing the error.

(i) Detect the exception.
(ii) Assuming the exception occurs during the EX stage (like overflow for example), must flush the instruction in the IF stage, by replacing it by all zeros (nop).
(iii) Must flush the instruction in the ID stage, but inserting zeros for control bits.
(iv) Must save the next instruction address in the EPC register.
(v) Must save the reason for the exception in the Cause register.
(vi) Must put address of start of the exception handler into the PC.

---

9. (a) Uses 10 bits for index (or cache address), so that the cache holds 1024 words of data.
(b) Since these are words the low-order bits of any address of a word are always zeros.
(c) The tag field holds the remaining 10 bits, after removing off the low-order 2 bits.
(d) The hardware extracts the low-order 10 bits (except for the lowest 2 bits), and uses these 10 bits as an address in the cache table, and uses these 10 bits as an address in the cache table. If the address is not in the cache, the processor will fetch the next instruction from memory.
(e) In case of a cache miss, see page 551 at the bottom.