Homework 2 Solutions

1. [Instruction executions]
   a) 0x0000D113
   b) 0xFFFF2FD3
   c) Illegal. Can't have two immediate operands
   d) Illegal. Register can't serve as offset
   e) ADD gets replaced with ADDI on compilation - $t0 = 0xFFFFFEF6
   f) 0xFFFFFFAB
   g) 0x000000A9
   h) 0x0235C4AC
   i) 0x0003D808
   j) 0x008D3BC0
   k) SRA keeps the sign → 0xFFFFFF80

2. [Memory Declarations]

1. LA $t0,a Loads the address of a into $t0, $t0=0x00000000
2. LW $s0,d Loads the word at d into $s0, $s0=0xB1B0A9A8 (since we assume little endian)
3. SB $t0,13($zero) Stores the least significant byte of $t0 to MEM[13], MEM[13]=0x00
4. LA $t3,f Loads the address of f into $t3, $t3=13=0x0000000D
5. LB $t4,0($t3) Loads the byte stored at MEM[$t3]=MEM[13], with sign extension, $t4=0x00000000
6. SW $t0,i Stores the word stored in $t0 to address i, MEM[16]= MEM[17]= MEM[18]= MEM[19]=0x00
7. LH $t1,0($t3) Illegal instruction, unaligned address
8. SH $t1,g Stores the lower half word of $t1 into the location starting at g (overwriting g and g+1), MEM[14]=0x00, MEM[15]=0xB4
9. LW $t5,17($zero) This is an illegal instruction, since the offset is not word aligned
10. LBU $t5,f Loads the byte at MEM[f]=MEM[13] to $t5, without sign extension, $t5=0x00000000

REGISTERS CONTENTS
$t0=0x00000000 (instruction 1)
$s0=0xB1B0A9A8 (instruction 2)
$t3=0x0000000D (instruction 4)
$t4=0x00000000 (instruction 5)
$t1=? (instruction 7)
$t5=0x00000000 (instruction 10)

LABEL	ADDRESS	FINAL CONTENTS

a:	0	0xA0
	1	0xA1
b:	2	0xA2
	3	0xA3
c:	4	0xA4
	5	0xA5
	6	0xA6
	7	0xA7
d:	8	0xA8
	9	0xA9
	10	0xB0
	11	0xB1
e:	12	0xB2
f:	13	~~0xB3~~ 0x00 (instruction 3)
g:	14	~~0xB4~~ 0x00 (instruction 8)
h:	15	~~0xB5~~ 0xB4 (instruction 8)
i:	16	~~0xB6~~ 0x00 (instruction 6)
	17	~~0xB7~~ 0x00 (instruction 6)
	18	~~0xB8~~ 0x00 (instruction 6)
	19	~~0xB9~~ 0x00 (instruction 6)

3. [Translating C code fragments to MIPS assembly]

```
.data

ia:  .word 7
ib:  .word 0x23
ic:  .word
id:  .word
ie:  .word
ig: .word

.text

main:
        li      $t0, 0x1234     #store 0x1234 in ia
        sw      $t0, ib         #store ia into ib

        add     $t0,$t0,$t0     #add ia and ib
        sw      $t0, ic         #store in ic

        lw      $t2, ib                 #load ib
        andi    $t2, $t2, 0x11  #and takes precedence over or
        or      $t2, $t2, $t0   #execute or
        sw      $t2, id         #store t2 into id
        lw      $t0, ig         #load ig
        xori    $t0, 0xFFFFFFFF #invert $t0
```

```
            sw     $t0, ie            #save to ie

            lw     $t0, ia
            lw     $t1, ib
            lw     $t2, ic
            lw     $tw, id

            sub    $t0, $t0, $t1      #ia – ib
            add    $t2, $t2, $t3      #ic + id
            xor    $t0, $t0, $t2      #(ia – ib) ^ (ic + id)
            sw     $t0, ig            # store ig
done:
```

4. [Translating C to MIPS assembly – Conditionals]

```
.data
a:  .word 4
b:  .word 30
c:  .word 20
d:  .word 10

main:
    lw      $s1, a
    lw      $s2, b
    lw      $s3, c
    lw      $s4, d

if:
    bne     $s1, $s1, elif   #branch if a != b
    li      $t0, 33
    sw      $t0, c
    j       Done

elif:
    beq     $s2, $s3, el     #branch if b != c
    li      $t0, 20
    sw      $t0, a
    j       Done

el:
nested_if:
    slt     $t0, $s1, $s2
    bgtz    $t0, nested_elif
    li      $t0, 10
    sw      $t0, b
    j       Done
```

```
 nested_elif:
    subi    $t0, $s3, 0x0A
    bgtz    $t0, nested_el
    li      $t0, 12
    sw      $t0, c
    j       Done
nested_el:
    li      $t0,5
    sw      $t0, a

Done:
```

5. [Memory and data references]
   a) Load Immediate – loads an immediate value into the register
      Load Address – loads the address into a register
      Load Word – loads the word that is at the specified address

   b)

```
.data
VarA:  .space 4          # allocate a 4 byte words for each each Var A,B,C,E,F
VarB:  .space 4
VarC:  .space 4
VarE:  .space 4
VarF:  .space 4
VarD: .word 10           # allocate 4 byte word for VarD, but initialized to 10
ArrayA: .space 400       # 100 words for ArrayA

main:

li      $t0,20           #load immediate value of 20 into $t0
sw      $t0,VarA         #save contents of $t0 at VarA

lw      $t0,VarA         #load word stored at VarA
sw      $t0,VarB         #save word in $t0 at VarB
la      $t0,VarA         #get address of ArrayA – needed since we want ArrayA[1]
lw      $t1,VarB         #get word stored at VarB
sw      $t1,4($t0)       #Save to location of ArrayA + offset of 4

la      $t0,VarE         #get address of VarE
sw      $t0,VarF         #put address of VarE into VarF
```

6. [Loops]

```
.data
A   .word 1,2,3,4,5,6
B   .word 0,0,0,0,0,0
size .word 6
```

```
.text
main:
    li      $t0,0               #offset counter
    lw      $t6, size($zero)    #remaining elements
for:
    blez    $t6, done
    lw      $t1, A($t0)         #load A[$t0] to $t1
    sw      $t1, B($t0)         #save $t1 into address B + offset $t0
    addi    $t0, $t0, 0x04      #increase to next word
    subi    $t6, $t6, 0x01      #one fewer elements left
    j       for
```

7. [Shifts and logicals]

```
a)  xori    $s0, $s0, 0xFFFFFFFF
b)  xori    $s1, $s1, 0xFFFFFFFF
    addi    $s1,$s1,0x1
c)  or      $s2, $s2, 0x2400C080
d)  and     $s3, $s3, 0xFFFFF7DB
e)  li      $t0, 0x00003002
    and     $t1, $s4, $t0
    beq     $t0, $t1, target
f)  sra     $s5, $s5, 6
g)  and     $s6, $s6, 0xE0
    sra     $s6, $s6, 5
```