COMPUTER ORGANIZATION

Homework Two

(100 points equally distributed across all questions)

1. Consider this simple ISA:

Opcode	Register 1	Register 2
8 bits	4 bits	4 bits

What is the size of the register file (or how many registers can be addressed)? If each register's size is the same as the instruction's size and the memory address size, what is the size of the memory in bytes? How do you update the program counter in this case?

There are 16 addressable registers. The size of memory is 2 bytes. The Program Counter (PC) is updated by adding '2' to the current PC.

2. Complete this table with the content of the register file after each operation, considering the memory's state as shown bellow. This is a two-operand ISA and for auto decrement, d=2:

Register file:

	Initial state	Sub R0, R0	Add R2, #2	Add R3, (4)	Add R4,10(R2)	Add R5, (R4+R3)	Add R6,@ (R5)	Add R7, -(R2)
R0	1	0	0	0	0	0	0	0
R1	1	1	1	1	1	1	1	1
R2	1	1	3	3	3	3	3	1
R3	1	1	1	2	2	2	2	2
R4	1	1	1	1	3	3	3	3
R5	1	1	1	1	1	4	4	4
R6	1	1	1	1	1	1	5	5
R7	1	1	1	1	1	1	1	3

Memory content:

Address	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
Content	5	4	3	2	1	3	9	8	7	11	5	4	7	2	1	0

3. Write a procedure *Reduce* that takes a vector with n elements, adds all the values, and stores the result $v0. The address of the original vector is in $a1 and the number of elements is stored in $a0.

```
Reduce:
    move $t0, $a0          -- move n length into temp reg.
    move $t1, $a1          -- move vector address into temp reg.
    lw   $t2, ($t1) +      -- load value at current pointer, increment
    sub  $t0, $t0, #1      -- decrement number of vecs. left in array.
    add  $v0, $v0, $t2     -- add loaded value to total
    bne  $t0, 0, Reduce    -- repeat until done with whole array.
    jr   $ra               -- return to caller (necessary?)
```

4. The procedure below uses 3 numbers stored in $a0, $a1 and $a2.

```
addi   $sp,$sp,-4      --moving by a word on the stack (alloc): SP <- SP - 4
sw     $t0,0($sp)      --store value in $t0 at the stack pointer: Mem[0 + SP] <- T0
add    $t0,$a0,$a1     --add $a0 and $a1 and put result in $t0: T0 <- A0 + A1
add    $v0,$t0,$a2     --add $t0 and $a2 and put result in $v0: V0 <- T0 + A2
lw     $t0,0($sp)      --retrieve old $t0 from stack: T0 <- Mem[0 + SP]
addi   $sp,$sp,4       --de-allocate mem on stack: SP <- SP + 4
jr     $ra             --return to caller: (no RTN?)
```

   a. Write comments by the side of each instruction using RTN that describe its functionality.
   b. What is the code doing?
      This code first is allocating a word on the stack and storing some value from $t0 in that new allocation. The code then adds the values in $a0 and $a1, using $t0 as a temporary location. This half result is then added to $a2 and this final result is put in $v0. The value that was in $t0 and stored on the stack is now retrieved from the stack and placed back in $t0. The stack is reset and the procedure ends.
   c. Write a procedure Add4 that calls this one as a procedure to add 4 numbers. Use QTSPIM to test it and show the screen captures.

```
1           .globl main
2   main:
3           jal Add4
4
5   Add4:   move $v1, $ra     # save return address
6           li $a0, 3         # load 4 random numbers to add together
7           li $a1, 5
8           li $a2, 7
9           li $a3, 10
10          jal Sumn          # jump and link to given procedure
11          add $s1, $v0, $a3 # add fourth value since the procedure only adds three
12          jr $v1
13
14
15  Sumn:   addi    $sp, $sp, -4    # moving by a word on the stack (alloc): SP <- SP - 4
16          sw      $t0, 0($sp)     # store value in $t0 at the stack pointer: Mem[0 + SP] <- T0
17          add     $t0, $a0, $a1   # add $a0 and $a1 and put result in $t0: T0 <- A0 + A1
18          add     $v0, $t0, $a2   # add $t0 and $a2 and put result in $v0: V0 <- T0 + A2
19          lw      $t0, 0($sp)     # retrieve old $t0 from stack: T0 <- Mem[0 + SP]
20          addi    $sp, $sp, 4     # de-allocate word on stack: SP <- SP + 4
21          jr      $ra             # return to caller: (no RTN?)
22
23
```

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs | Int Regs [10] | Data | Text

Int Regs [10]

```
PC       = 4194372
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 805371664

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 15
R3  [v1] = 4194368
R4  [a0] = 3
R5  [a1] = 5
R6  [a2] = 7
R7  [a3] = 10
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 25
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

result

Text

```
                                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)           ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4        ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4         ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2           ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]   ; 188: jal main
[00400018] 00000000  nop                     ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10          ; 191: li $v0 10
[00400020] 0000000c  syscall                 ; 192: syscall # syscall 10 (exit)
[00400024] 0c10000a  jal 0x00400028 [Add4]   ; 3: jal Add4
[00400028] 001f1821  addu $3, $0, $31        ; 5: move $v1, $ra # save return address
[0040002c] 34040003  ori $4, $0, 3           ; 6: li $a0, 3 # load 4 random numbers to add together
[00400030] 34050005  ori $5, $0, 5           ; 7: li $a1, 5
[00400034] 34060007  ori $6, $0, 7           ; 8: li $a2, 7
[00400038] 3407000a  ori $7, $0, 10          ; 9: li $a3, 10
[0040003c] 0c100012  jal 0x00400048 [Sumn]   ; 10: jal Sumn # jump and link to given procedure
[00400040] 00478820  add $17, $2, $7         ; 11: add $s1, $v0, $a3 # add fourth value since the procedure only adds three
[00400044] 00600008  jr $3                   ; 12: jr $v1
[00400048] 23bdfffc  addi $29, $29, -4       ; 15: addi $sp, $sp, -4 # moving by a word on the stack (alloc): SP
[0040004c] afa80000  sw $8, 0($29)           ; 16: sw $t0, 0($sp) # store value in $t0 at the stack pointer: Mem[0 + SP]
[00400050] 00854020  add $8, $4, $5          ; 17: add $t0, $a0, $a1 # add $a0 and $a1 and put result in $t0: T0
[00400054] 01061020  add $2, $8, $6          ; 18: add $v0, $t0, $a2 # add $t0 and $a2 and put result in $v0: V0
[00400058] 8fa80000  lw $8, 0($29)           ; 19: lw $t0, 0($sp) # retrieve old $t0 from stack: T0
[0040005c] 23bd0004  addi $29, $29, 4        ; 20: addi $sp, $sp, 4 # de-allocate word on stack: SP
[00400060] 03e00008  jr $31                  ; 21: jr $ra # return to caller: (no RTN?)

                                    Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1        ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672          ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672          ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13           ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2          ; 96: srl $a0 $k0 2 # Extract ExcCode Field
```

Memory and registers cleared

SPIM Version 9.1.23 of December 4, 2021
Copyright 1990-2021 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

Memory and registers cleared

SPIM Version 9.1.23 of December 4, 2021
Copyright 1990-2021 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.