**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.

1. 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 |

1. 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?)

1. 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?)

* 1. Write comments by the side of each instruction using RTN that describe its functionality.
  2. 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.

* 1. 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.