Jackson Myers

02/24/2023

HW02

1. ENCODE:

   srl $t0, $s0, 4 - Initial Instruction

   op 00000 R16 R8 4 SRL - Format instruction and resolve registers for R-type encoding

   000000 00000 10000 01000 00100 000010 - Further resolve instructions

   0000 | 0000 | 0001 | 0000 | 0100 | 0001 | 0000 | 0010 Split for hex conversion

   0x00104102 - Convert to hex for final answer

   sub $t0, $t1, $s1 - Initial Instruction

   000000 $9 $17 $8 00000 100010 - Format instruction and resolve registers for R-type

   000000 01001 10001 01000 00000 100010 - Further resolve instruction to binary

   0000 | 0001 | 0011 | 0001 | 0100 | 0000 | 0010 | 0010 - Split for hex conversion

   0x01314022- Convert to hex for final answer

2. DECODE:

   0x29010064 - Encoded instruction

   2 | 9 | 0 | 1 | 0 | 0 | 6 | 4 - Split for conversion to binary

   001010 | 01000 | 00001 | 0000 0000 0110 0100 - I-type format, convert end to hex

   SLTI R1 R8 0x0064 - Convert binary to register numbers, resolve opcode

   SLTI $at, $t0, 0x0064 - Final answer instruction

0x21290011 - Encoded instruction

2 | 1 | 2 | 9 | 0 | 0 | 1 | 1 - Split for conversion to binary

001000 | 01001 | 01001 | 0000 0000 0001 0001 - I-type formatting, Addi opcode

ADDI R9, R9, 17 - Resolve registers and constant

ADDI $t0, $t0, 17 - Final instruction

3. Translation:

   i = $s0

   j = $s1

   Base of A[] = $s2

   Base of B[] = $s3

   3.1:

```
        bne $s0, $s1, else:
        addi $s1, $s1, 1
        add $s1, $s0, $s1
else:
        addi $s0, $s0, -1
        add $s1, $s0, $s1
```

3.2:

```
sll $s4, $s1, 2          - this line defines and stores a register for offset for j
add $s4, $s2, $s4        - add to A base address
sll $s5, $s0, 2          - offset for i, i * 4
add $s5, $s3, $s5        - add offset to B base address
lw $s6, 4($s5)           - load the base address of  B + i + 1
lw $s7, 4($s3)           - load the base address of B + 4 bytes
add $s6, $s6, $s7        - add loaded values together
sw $s6, 0($s4)           - store the value into the address of A + j
```

3.3:

```
        addi $s0, $zero, 0      -zero out i
loop:   sll $s4, $s0, 2         -shift i twice to *4, because we will use as index
        add $s4, $s2, $s4       -add i to A base for indexed value
        lw $s4, 0($s4)          -load index address into register
        beq $s1, $s4, end_while:        -branch if values are equal to end_while
        addi $s0, $s0, 2        - add 2 to i
        j loop                  - jump back to start of loop
end_while:
```

4.

```
main:
        jal test                    - jump and link to test function

test:
        addi $sp, $sp, -8           -create stack
        sw $s0, 0($sp)              -store values in stack
        sw $s1, 4($sp)              -store values in stack
        jal lookup                  -call lookup function
        lw $s1, 0($sp)              -resolve registers back from stack
        lw $s0, 4($sp)
        jr $ra

lookup:
        addi $sp, $sp, -8           -create stack
        sw $s0, 0($sp)
        sw $s1, 4($sp)
        addi $t0, $zero, 0          -make temporary variable i = 0
        loop:   beq $t0, $a1, end_loop          -loop under the condition that i<size
                sll $s1, $t0, 2             -store offset of i in $s1
                add $s1, $s0, $s1           -add base address and index
                beq $s1, $a2, return        -if equal return the value with return branch
                addi $t0, $t0, 1
                j loop

return:                             -if value is found go here
        move $v0, $t0
        lw $s1, 0($sp)              -resolve registers back from stack
        lw $s0, 4($sp)
        jr $ra

end_loop:                           -if loop is invalid, go here
        li $t1, -1
        move $v0, $t1
        lw $s1, 0($sp)              -resolve registers back from stack
        lw $s0, 4($sp)
        jr $ra
```

5.
```
int i = 10
while (i > 0)
{
        A[i] = i + 2;
}
```