|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of instruction** | **Encoding** | | | | | | **Syntax** | **Operation** |
| **R-type** | opcode | s | t | d | sa | function |  |  |
| 000 | sss | ttt | ddd | 0 | 000 | add $d, $s, $t | $d ← $s + $t (add) |
| 000 | sss | ttt | ddd | 0 | 001 | sub d, $s, $t | $d ← $s - $t (subtract) |
| 000 | xxx | ttt | ddd | 1 | 010 | sll $d, $t, sa | $d ← $t << sa (shift left logical) |
| 000 | xxx | ttt | ddd | 1 | 011 | srl $d, $t, sa | $d ← $t >> sa (shift right logical) |
| 000 | sss | ttt | ddd | 0 | 100 | and $d, $s, $t | $d ← $s & $t (bitwise and) |
| 000 | sss | ttt | ddd | 0 | 101 | or $d, $s, $t | $d ← $s | $t (bitwise or) |
| 000 | sss | ttt | ddd | 0 | 110 | xor $d, $s, $t | $d ← $s ^ $t (bitwise xor) |
| 000 | sss | ttt | ddd | 0 | 111 | slt $d, $s, $t | If($s < $t) then $d ←1; PC←PC+1 else $d ← 0; PC ← PC + 1(set on less than) |
| **I-type** | opcode | s | t | imm/address | | |  |  |
| 001 | sss | ttt | iiiiiii | | | addi $t, $s, imm | $t ← $s + imm (add immediate) |
| 010 | sss | ttt | iiiiiii | | | lw $t, offset($s) | $t← M[$s + offset] (load word) |
| 011 | sss | ttt | iiiiiii | | | sw $t, offset($s) | M[$s + offset] ← $t |
| 100 | sss | ttt | iiiiiii | | | beq $t, $s, offset | If($s == $t) then PC ← PC + 1 + offset  else PC ← PC + 1 (branch on equal) |
| 101 | sss | ttt | iiiiiii | | | andi $t, $s, imm | $t ← $s & imm (bitwise and immediate) |
| 110 | sss | ttt | iiiiiii | | | ori $t, $s, imm | $t ← $s | imm (bitwise or immediate) |
| **J-type** | opcode | target address | | | | |  |  |
| 111 | ttttttttttttt | | | | | j target | PC ← PC[15:13] || target\_address |

A screenshot of a computer

Description automatically generated

Să se determine valoarea pară maximă dintr-un șir de N numere stocate în memorie începând cu adresa A (A≥12). A și N se citesc de la adresele 0, respectiv 4. Rezultatul se va scrie în memorie la adresa 8.

Instr0: lw $1, 0($0) // Load A from address 0 => 010\_000\_001\_0000000

Instr1: lw $2, 4($0) // Load N from address 4 => 010\_000\_010\_0000100

Instr2: xor $4, $4, $4 // Initialize the counter to 0 => 000\_100\_100\_100\_0\_110

Instr3: xor $5, $5, $5 // Initialize the max even number to 0 => 000\_101\_101\_101\_0\_110

Instr4: add $6, $0, $1 // Initialize address pointer to A => 000\_000\_001\_110\_0\_110

Instr5: beq $4, $2, end\_loop // Check if the counter has reached N => 100\_010\_100\_0010001

Instr6: lw $3, 0($6) // Load the next number from memory => 010\_110\_011\_0000000

Instr7: andi $7, $3, 1 // Check the least significant bit => 101\_011\_111\_0000001

Instr8: beq $7, $0, even // Branch if the number is even => 100\_000\_111\_0001011

Instr9: addi $4, $4, 1 // Increment counter => 001\_100\_100\_0000001

Instr10: jump 14 // go to the next address => 111\_0000000001110

Instr11: even: slt $7, $3, $5 // Check if the number is less than the current maximum => 000\_011\_101\_111\_0\_111

Instr12: beq $7, $0, max // Branch if the number is not less than the current maximum => 100\_000\_111\_0001111

Instr13: addi $4, $4, 1 // Increment counter => 001\_100\_100\_0000001

Instr14: add $6, $6, 4 // Move to the next address => 000\_110\_100\_110\_0\_110

Instr15: max: sw $3, 0($5) // Store the current number as the new max => 011\_101\_011\_0000000

Instr16: jump Instr5 Jump back to the loop start => 111\_0000000000101

Instr17: end\_loop: sw $5, 8($0) // Write the max even number to memory at address 8 => 011\_000\_101\_0001000