Logika Cyfrowa - Lista 12

Zadanie 1

```
add x5, x6, x7
addi x5, x5, -5
```

Zadanie 2

Załóżmy, że w rejestrze `x1` mamy zmienną `x`, w `x2` zmienną `a`, w `x3` zmienną `b`, a w `x4` zmienną `c`.

Po instrukcji `add x1, x2, x3` mamy
`x = a + b`
Po instrukcji `add x1, x4, x1` mamy
`x = x + c`

Zatem wyrażenie odpowiadające tym instrukcjom to

```
x = a + b + c;
```

Zadanie 3

```
sub x28, x28, x29 // zapisujemy w x28 i-j
add x28, x28, x10 // dodajemy do x28 (i-j) adres tablicy A
lb x5, 0(x28) // zapisujemy w rejestrze x5 wartość A[i-j] (czyli 1 bajt)
sb x5, 8(x11) // zapisujemy wartość x5 do B[8] (czyli 1 bajt)
```

Zadanie 4

Załóżmy że:
`x12` - adres tablicy `A`
`x5` - zmienna `x`
`x6` - zmienna `i`

`lb x30, 8(x12)` → w `x30` zapisujemy wartość `A[8]`

`add x30, x30, x5` → do `x30` (A[8]) dodajemy wartość `x`

```
add x31, x12, x6
sb  x30, 0(x31)
```

→ zapisjemy wartość `x30` w `A[i]`

Zatem to wyrażenie możemy zapisać jako

```
A[i] = A[8] + x;
```

Zadanie 5

`opcode` = 3_{16}=0000011_2 - czyli rodzaj instrukccji to LOAD
`funct3` = 2_{16}=010_2 - czyli instrukcja to `lw`
`rs1` = 27_{10}=11011_2
`rd` = 3_{10}=00011_2
`imm` = 4_{16}=000000000100_2

Instrukcja:
`lw x3, 4(x27)`

Kod binarny instrukcji:

`000000000100 11011 010 00011 0000011`
(`imm` `rs1` `funct3` `rd` `opkod`)

Zadanie 6

`sw x5, 32(x30)`

`imm` = 32_{10}=000000100000_2
`imm[11:5] = 0000001`
`imm[4:0] = 00000`
`rs1` = 30_{10}=11110_2
`rs2` = 5_{10}=00101_2
`funct3` = 010_2
`opcode` = 0100011_2

Instrukcja w kodzie binarnym:

`0000001 00101 11110 010 00000 0100011`
(`imm[11:5]` `rs2` `rs1` `funct3` `imm[4:0]` `opkod`)

Zadanie 7

`0000 0000 0001 0000 1000 0000 1011 0011`

`opcode` = 0110011_2, więc jest to instrukcja o formacie R

Kod po odpowiednim podzieleniu w celu łatwiejszego odczytu wartości:
`0000000 00001 00001 000 00001 0110011`

`rd` = 00001_2=1_{10}
`funct3` = 000_2
`rs1` = 00001_2=1_{10}
`rs2` = 00001_2=1_{10}
`funct7` = 0000000_2

Instrukcja:

`add x1, x1, x1`

Zadanie 8

`x5 - 0x0000aaaa = 0000 0000 0000 0000 1010 1010 1010 1010`
`x6 - 0x12345678 = 0001 0010 0011 0100 0101 0110 0111 1000`

a)
`slli x7, x5, 4` (przesunięcie logiczne w lewo o 4)
`x7 - 0000 0000 0000 1010 1010 1010 1010 0000`

`or x7, x7, x6`

```
    0000 0000 0000 1010 1010 1010 1010 0000
 or 0001 0010 0011 0100 0101 0110 0111 1000
 ------------------------------------------
    0001 0010 0011 1110 1111 1110 1111 1000
```

Wartość `x7` to `0001 0010 0011 1110 1111 1110 1111 1000`, czyli `0x123efef8`

b)
`srli x7, x5, 3` (przesunięcie logiczne w prawo)
`x7 - 0000 0000 0000 0000 0001 0101 0101 0101`

`andi x7, x7, 0xfef`
`0xfef` = `1111 1110 1111`, ale to liczba ujemna, więc musimy dopełnić jedynkami

```
     0000 0000 0000 0000 0001 0101 0101 0101
 and 1111 1111 1111 1111 1111 1111 1110 1111
 -------------------------------------------
     0000 0000 0000 0000 0001 0101 0100 0101
```

Wartość `x7` to `0000 0000 0000 0000 0001 0101 0100 0101`, czyli `0x00001545`