# Lista 14

## zadanie 1

| Rodzaj | Cykle na instrukcję | instrukcje na 100 instrukcji | cykle na 100 instrukcji |
|---|---|---|---|
| OP | 4 | 24 | 96 |
| typ I (bez LOAD) | 3 | 20 | 60 |
| LOAD | 5 | 25 | 125 |
| STORE | 4 | 10 | 40 |
| BRANCH | 3 | 11 | 33 |
| JAL | 3 | 2 | 6 |
| typ U | 3 | 8 | 24 |
| | | ***Razem*** | ***384*** |

Mamy zatem średnio **384 cykle** na **100 instrukcji**, czyli **3,84 cykla na instrukcję**.

## zadanie 2

W zadaniu przyjąłem zgodnie z wykładem 12, że w rejestrze x0 jest *zawsze zero*. Program oblicza sumę liczb od 0 do 9 włącznie i zapisuje ją w rejestrze x9.
Po rozwinięciu pętli, w programie znajdują się łącznie 23 instrukcje OP i OP-IMM (3 na początku i 10 x 2 w ciele pętli), które wykonywane są w ciągu czterech cykli oraz 11 instrukcji rodzaju BRANCH (skok bge z porównaniem kolejno liczb 0 … 10 z dziesiątką), które wykonywane są przez trzy cykle.
Daje nam to łącznie **125 cykli procesora** i **34 instrukcje**, z czego możemy wywnioskować, że średnia ilość cykli na instrukcję to w przybliżeniu **3,68**.

## zadanie 3

***a)***
Nowe elementy ścieżki danych nie będą nam potrzebne. Do obliczenia sumy RS1 i RS2 potrzebnej do określenia adresu wystarczy użyć ALU, podobnie jak robi to instrukcja LOAD.

***b)***
Nie trzeba nic modyfikować.

***c)***
Nie trzeba nic dodawać.

***d)***
Nowe sygnały sterujące nie będą potrzebne.

***e)***
Żadne poważne modyfikacje nie będą potrzebne. Dodana nowa instrukcja lwiw wykonywana by była analogicznie do instrukcji load z różnicą pojawiającą się na etapie MEM_ADDR, gdzie wygenerowany sygnał sterujący alu_bsel musiałby wybrać RS2 zamiast IMM.