Project 1

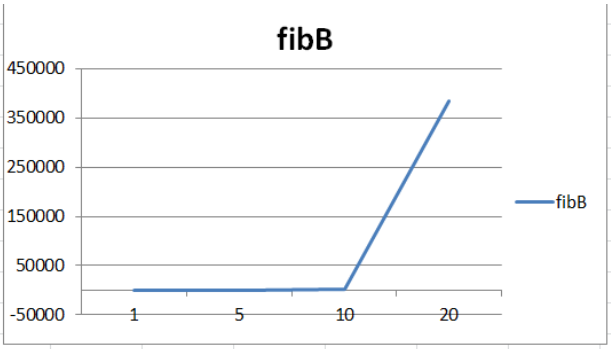
首先我先分析， complexity

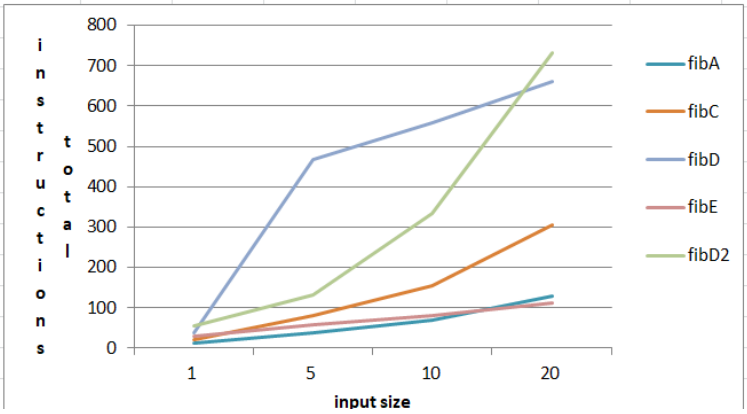
- 理論大體來說:

algorithm	complexity
A	O(N)
B	O(2^N)
C	O(N)
D	O(logN)
D2(not recur)	O(N)
E	O(logN(前半)+N/2(後半))

- relationship between input size and total instruction count:

1	5	10	20	input size
13	39	69	129	A
23	268	3103	383098	B
21	81	156	306	C
37	467	559	661	D
31	59	81	111	E
55	133	333	733	D2





- 接下來我 A~E 一一把理論跟實際去做比較解釋:

1. 演算法 A 是 Iterative Method:

因為就直接照 Fibonacci number 公式 $F_n = \begin{cases} 0 & (n=0) \\ 1 & (n=1) \\ F_{n-1} + F_{n-2} & (n>1) \end{cases}$ ，去做 n 次，所以很直觀的 complexity 是 O(N)，實際來說算相當線性成長。

2. 演算法 B 是 Recursive Method:

因為是在 Recursive terminate 條件達到以前一直是在叫分兩項的自己，Recursive terminate 條件達到開始回加，但這樣只就是分 2^N 個，所以 complexity 是 O(2^N)。實際來說可以看的出來成長是很快，看的出來也是指

數型成長，但是並不是以 2 為底的指數成長，後來跑指數回歸是 17*1.656^input size，我想可能是我跑的點不夠，但為了公平起見，我不能單為一個演算法多跑更多 input size，也可能是我除了分兩支以外還有很多 recursion 為了層層呼叫保護變數的 sw,lw 等動作而影響到 total instrutions。而且 input size=20 的跑了將近兩分鐘，有夠可怕的。

3. 演算法 C 是 Tail Recursion:

    他雖然是 recursion 但他只是利用 recursive 一層層減掉 n，每次 ` 減一並傳送下一階段的 a=b,b=a+b 這樣做 n 次本質上就是第一種 Iterative Method，故為 O(N)，實際上來看也算是當線性成長。

4. 演算法 D 是 Q Matrix:

    主要是 2X2 矩陣[[1,1],[1,0]]去做多次相乘，理論上 recursive 如果分支可同時做會是 O(logn)但是在演算法二的 recursive 不可平行，所以我也做單純的 multiplying with Q n times，此兩種理論上為 O(logN)跟 O(N)，實際上第一種(D)因為 1 就是直接 return Q1 所以不當基數不太好，5 以上雖然數字 467 起跳但成長像 logN，而第二種(D2)就相當線性成長。

5. 演算法 E 是 Fast Doubling Method:

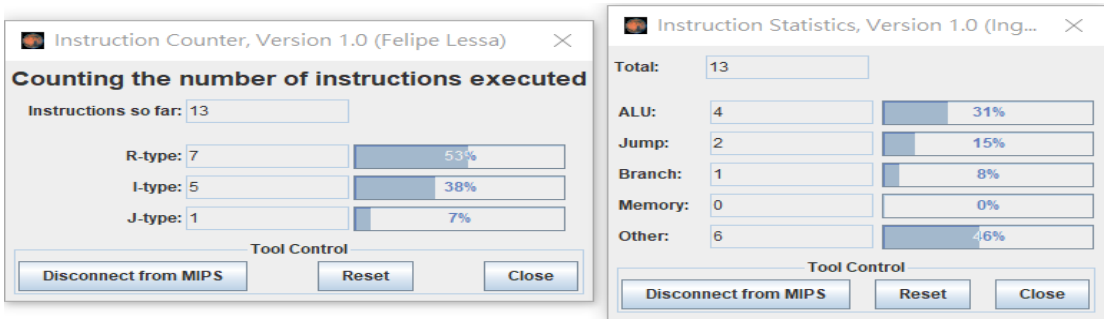
    這方法是此次最快，也在 input size 耗最少 instructions，他以 2 指數的速度成長，並確保每次相連的兩組解，這樣就可以確保一定超過一半的部分再乖乖線性成長，雖然這樣來講是 O(logN(前半)+N/2(後半))=O(N)，但是還是比正常的 O(N)快多了，實際上他也是跑最少的，看起來幾乎是線性的。

- 接下來分析 instruction distribution:

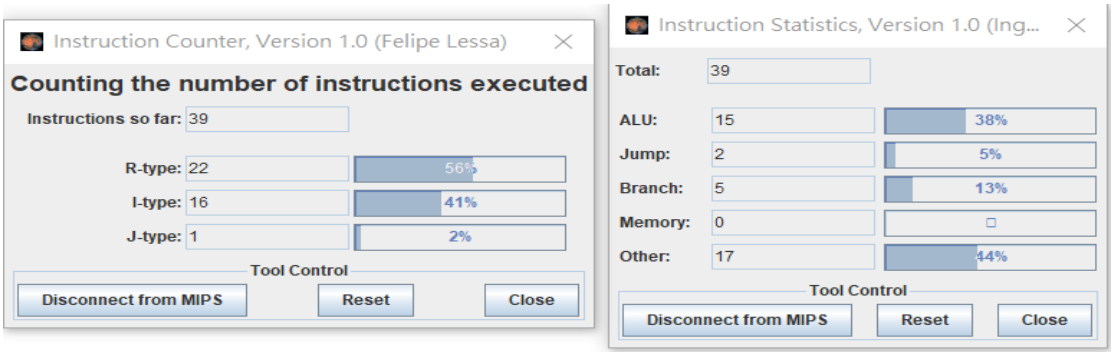
1.演算法 A 是 Iterative Method:

Iterative Method 基本上就大量的 R type: move & add，然後每 Iterate 要在 I type 的 slti & beq 所以 R type 最多次之 I type，實際觀察也是如此。而且有 input size 越大越明顯的趨勢。因為都沒 lw & sw 所以就沒用到 memory。J type 的 jal &jr 只有跳到 function 及跳回來所以 2。

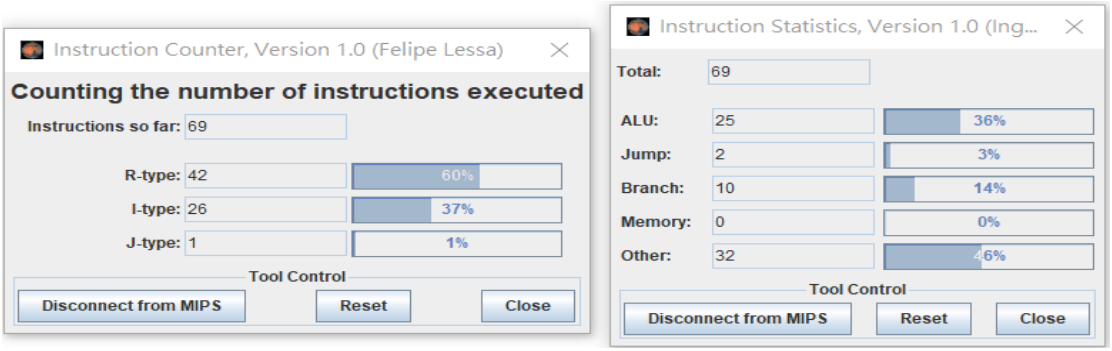
- Iterative Method input size=1:



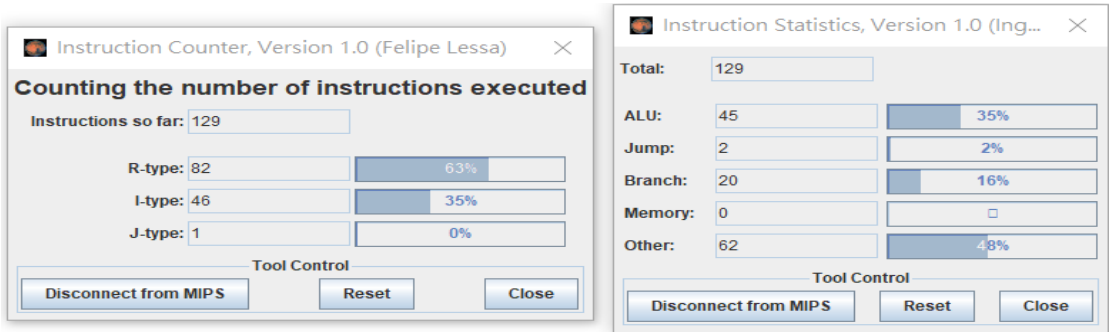
- Iterative Method input size=5



- Iterative Method input size=10:



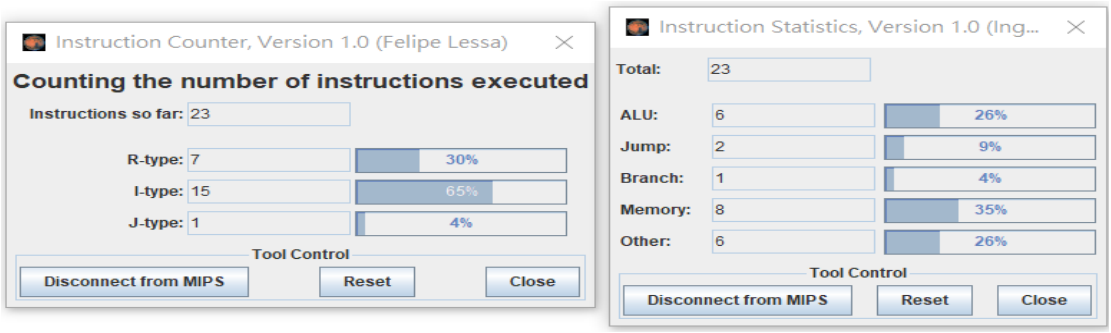
- Iterative Method input size=20:



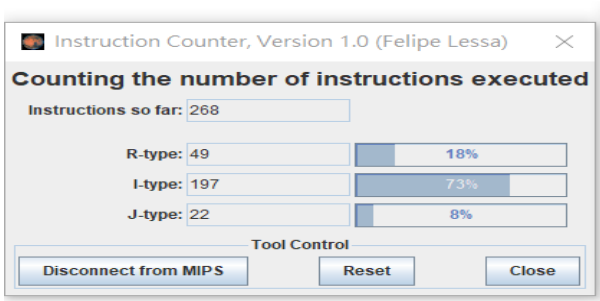
2. 演算法 B 是 Recursive Method:

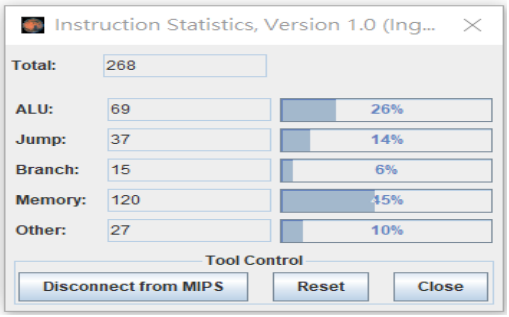
Recursive Method 基本上就大量的 I type: lw & sw 來保護變數，然後也每 Recursive R type 主要是 move 所以 I type 最多次之 R type，實際觀察也是如此。而且有 input size 越大越明顯的趨勢。Resursive 的特點是他很快分佈就會收斂到一個樣子，可以是 function 一直重複 call 自己的關係。

- Recursive Method input size=1

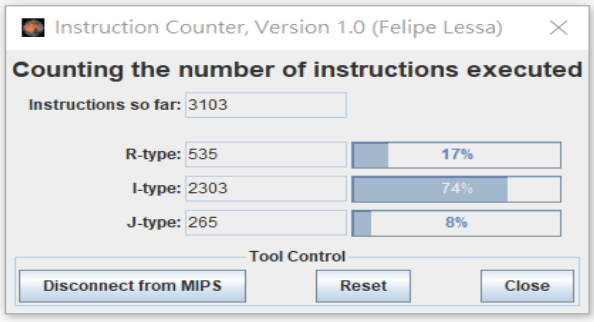


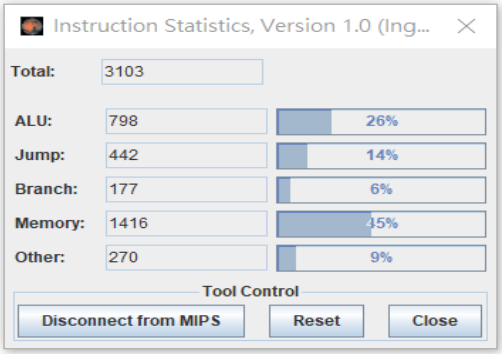
- Recursive Method input size=5



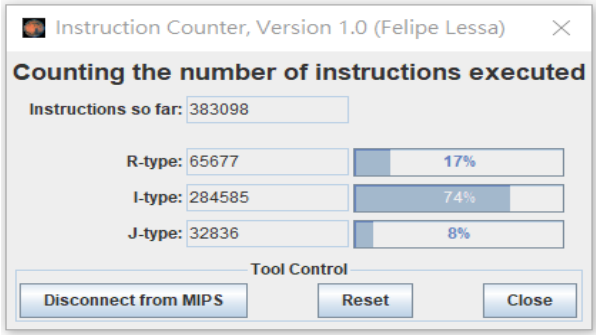


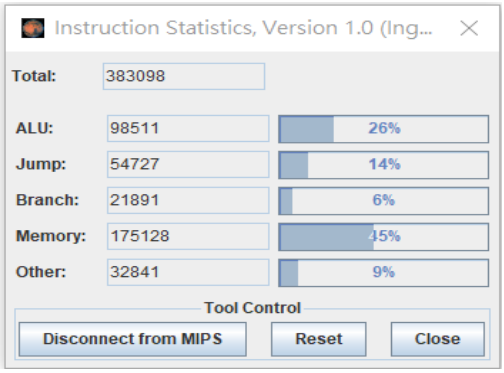
- Recursive Method input size=10





- Recursive Method input size=20

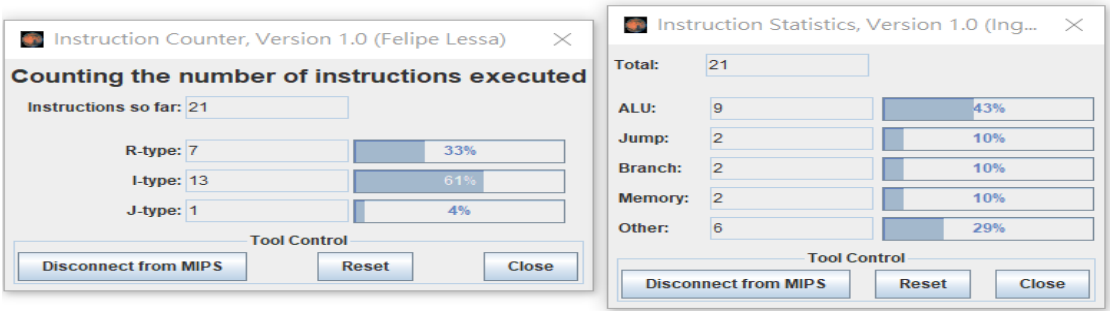




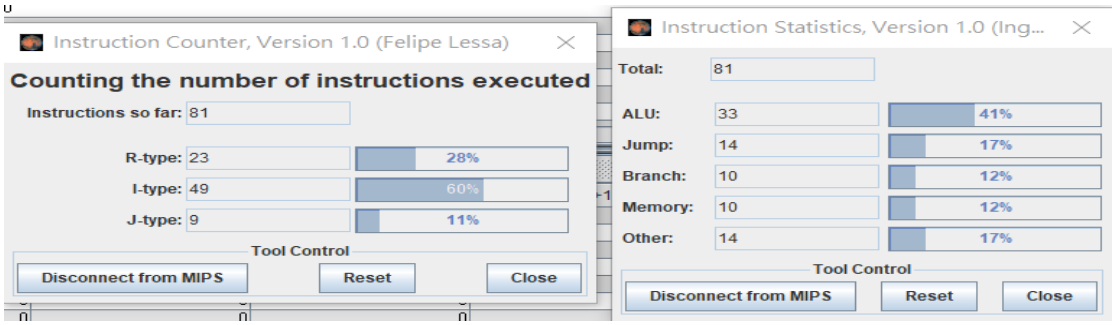
3. 演算法 C 是 Tail Recursion Method:

Tail Recursion Method 基本上雖然是 Recursion 實際上 Recursion 只是模仿著 iterate 所以就大量的 I type: lw & sw 來保護變數，然後也每 Recursive R type 主要是 move 所以 I type 最多次之 R type，實際觀察也是如此。而且有 input size 越大越明顯的趨勢。Resursive 的特點是他很快分佈就會收斂到一個樣子，可以是 function 一直重複 call 自己的關係。注意到他本上個 Recursion Method 少更多的 I type 是因為我減少要 lw sw 保護的變數，因為他模仿 iterate，不會同時一些暫存，所以我不保護。

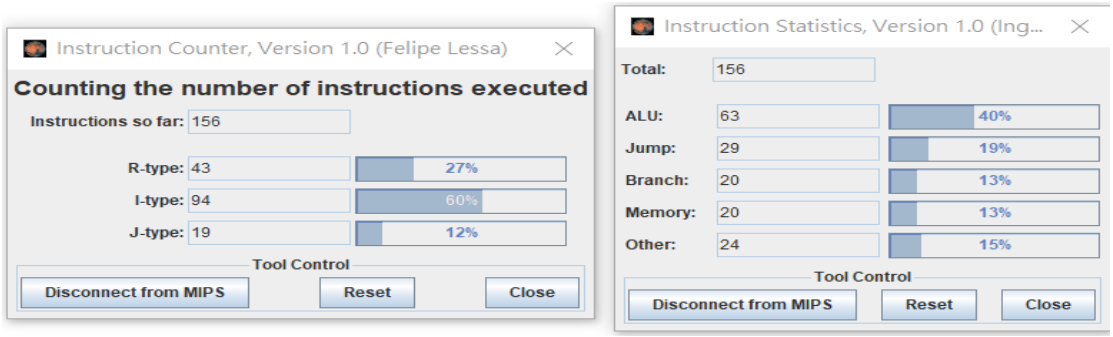
- Tail Recursion input size=1



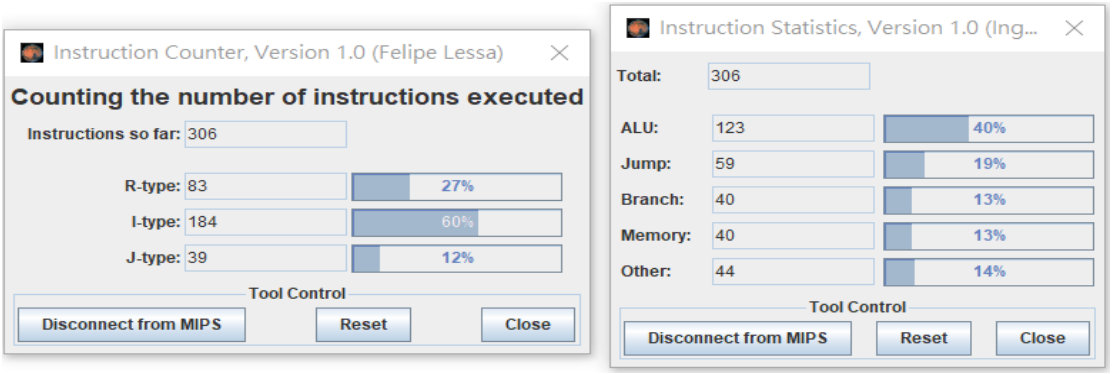
- Tail Recursion input size=5



- Tail Recursion input size=10



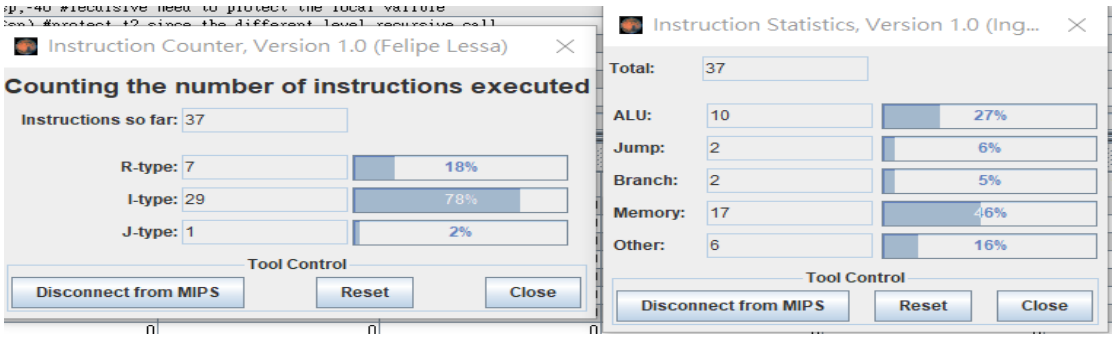
- Tail Recursion input size=20



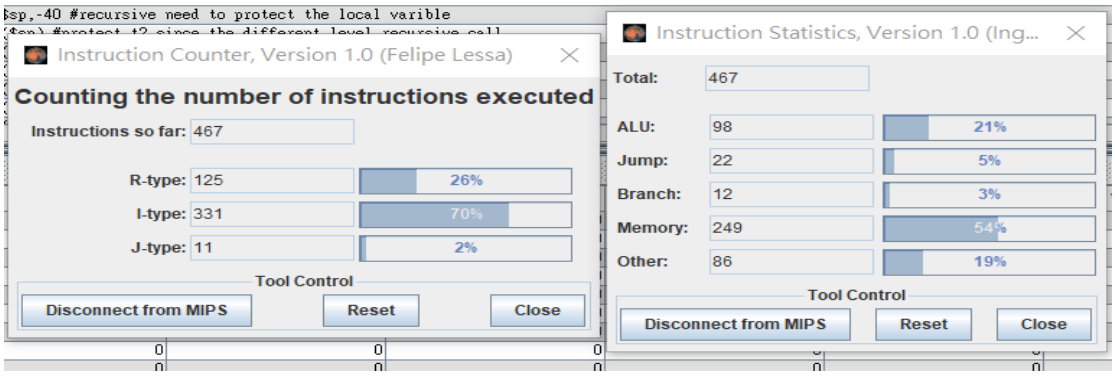
4-1. Q Matrix recursive:

Q Matrix recursive 主要是把每個 Q^N 一直除 2recursive 分治到 call find 直到 Q^1，所以大量的 I 是必要的只是 mmul 內也有大量需要 ALU 的運算，大抵上屬於 R type mul & add，所以 I type 最多次之 R type，接下來要跟 Q Matrix 單純 iterate 比較下。

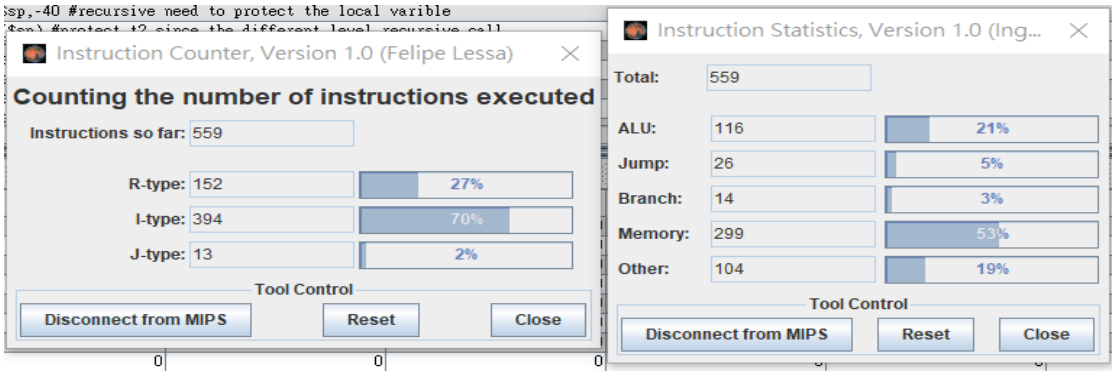
- Q Matrix input size=1:



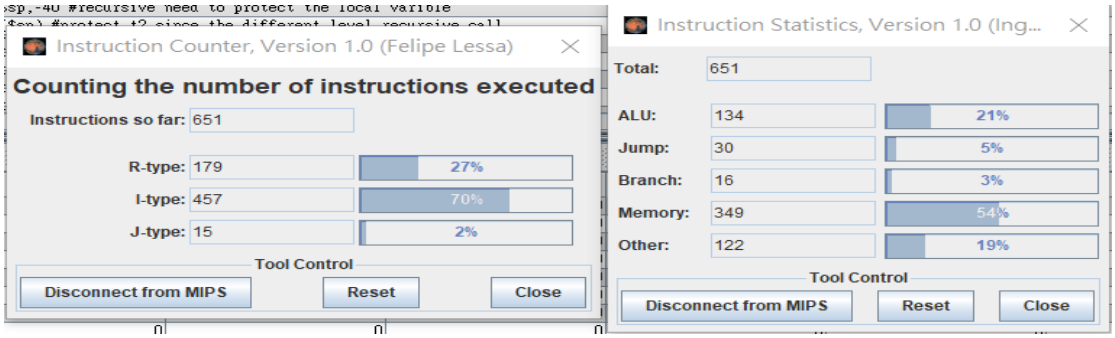
- Q Matrix input size=5



- Q Matrix input size=10



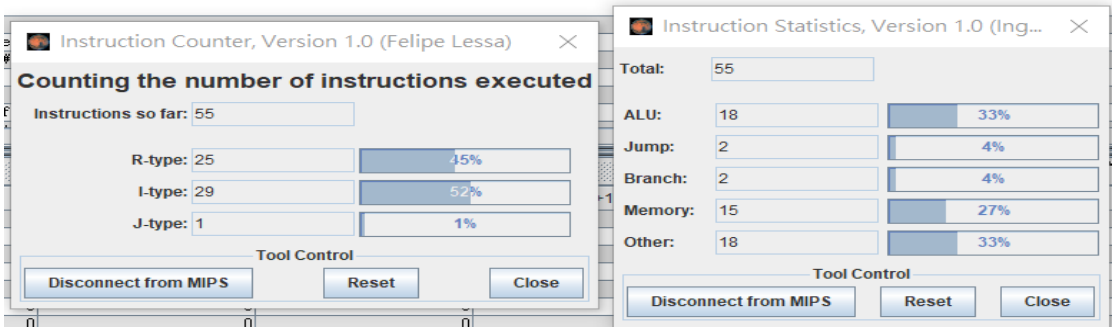
- Q Matrix input size=20



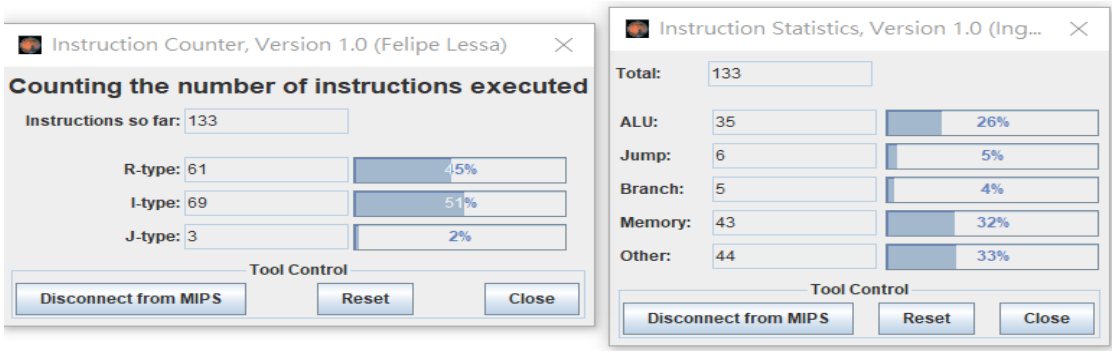
4-2. Q Matrix 單純 iterate:

Q Matrix 單純 iterate 主要是很直接去做需要次數的 mmul，是 mmul 內也有大量需要 ALU 的運算，大抵上屬於 R type mul & add，但其實為了記住計算得值到矩陣個位置也伴有大量的 I type lw &sw，單純 iterate 也放大了 I type 比例，所以更 I type 最多次之 R type。

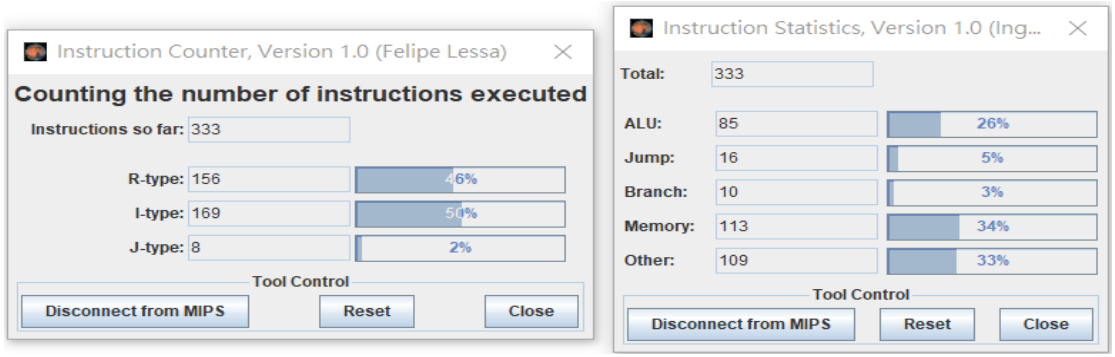
- Q Matrix input size=1:



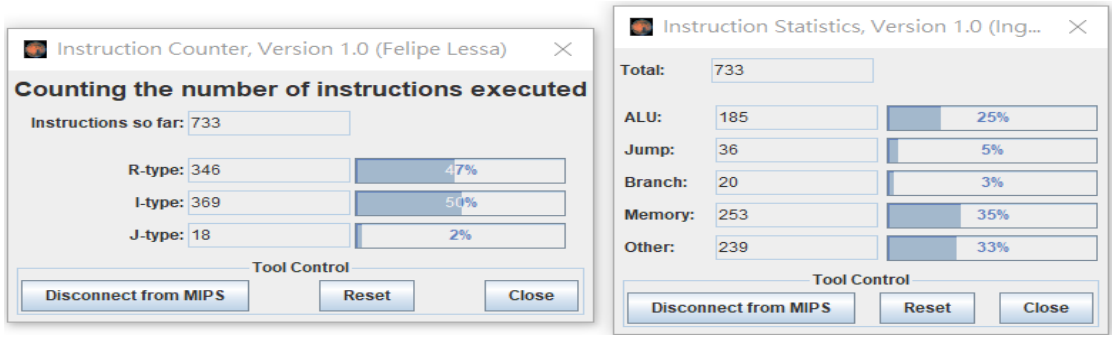
- Q Matrix input size=5



- Q Matrix input size=10



- Q Matrix input size=20

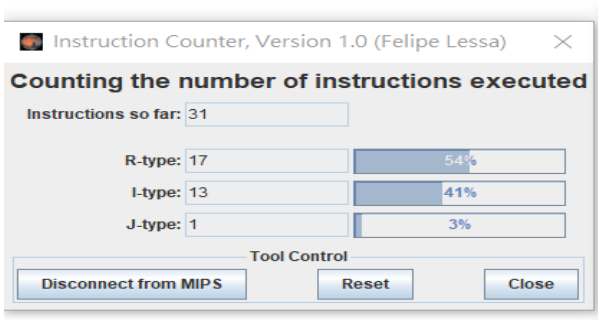


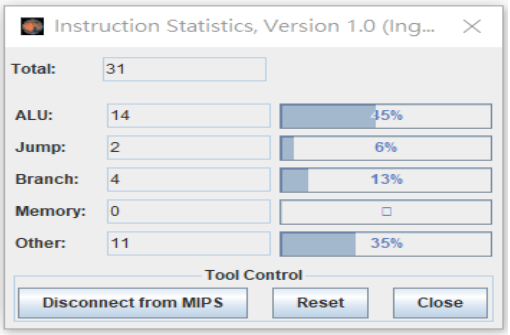
5. Fast Doubling Method:

Fast Doubling Methodg 是前半倍數成長後半在一步步來，所以分布是相對其他的演算法不穩定些。照理來說比二的次方數大一點點會很像倍數成長的樣子，小一點點則是倍數成長跟一步步參半。實際的分步卻比想像中穩定多，大抵上 R

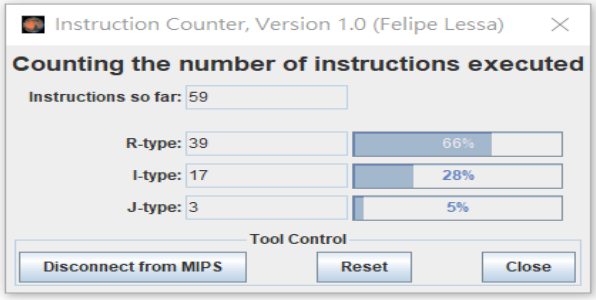
tpye 因為 mul 及 move 及 add 居多，然後 I type 主要不是來自 lw 及 sw 而是來自 addi 及 li。所以 R type 最多次之 I type。就像之前講的，分布是相對其他的演算法不穩定些，所以個比例是微微在一個小範圍震盪。

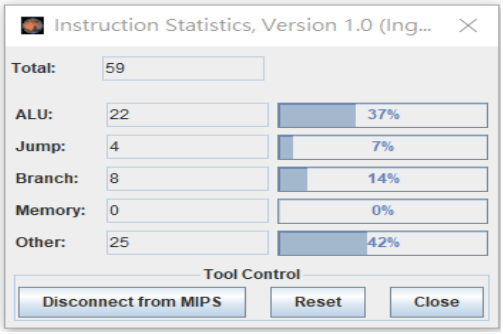
- Fast Doubling Method input size=1



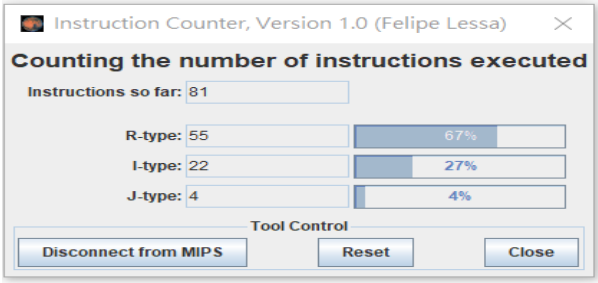


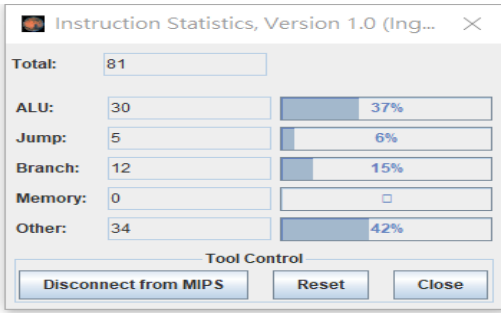
- Fast Doubling Method input size=5





- Fast Doubling Method input size=10





- Fast Doubling Method input size=20