EE3450 Computer Architecture

Project 1. MIPS Assembly Programming

Name: Yu-Hsiu Huang ID: 104061249 Department: EE

Introduction

In this project, five algorithms are provided to solve Fibonacci number. Our goal is to implement these algorithms with MIPS assembly code and verify it with C code. After that, the total number of instruction used and the ratio of each type of instruction are plotted versus the order of the Fibonacci number. Furthermore, the complexity is going to be analyzed based on these graphs.

The five algorithms are Iterative Method, Recursive Method, Tail Recursion, Q-Matrix Method, and Fast Doubling Method. Each method has its own properties and pros and cons. We will discuss the detail in the following content.

Analysis

Table 1. Fibonacci number calculated by C code

order	5	7	10	13	15	20	25	30
value	5	13	55	233	610	6765	75025	832040

Table 2. Iterative Method: instruct. count of different type vs. order

order	ALU	Jump	Branch	Memory	Other	R-type	I-type	J-type	Total
5	14	2	4	0	11	16	14	1	31
7	18	2	6	0	15	22	18	1	41
10	24	2	9	0	21	31	24	1	56
13	30	2	12	0	27	40	30	1	71

15	34	2	14	0	31	46	34	1	81
20	44	2	19	0	41	61	44	1	106
25	54	2	24	0	51	76	54	1	131
30	64	2	29	0	61	91	64	1	156
35	74	2	34	0	71	106	74	1	181
40	84	2	39	0	81	121	84	1	206
45	94	2	44	0	91	136	94	1	231

Table 3. Recursive Method: instruct. count of different type vs. order

order	ALU	Jump	Branch	Memory	Other	R-type	I-type	J-type	Total
5	61	30	27	42	19	41	123	15	179
7	165	82	74	120	45	106	339	41	486
10	709	354	320	528	181	446	1469	177	2092
13	3013	1506	1362	2256	757	1886	6255	753	8894
15	7893	3946	3569	5916	1977	4936	16392	1973	23301
20	87565	43782	39601	65670	21895	54731	181891	21891	258513

Table 4. Tail Recursive Method: instruct. count of different type vs. order

order	ALU	Jump	Branch	Memory	Other	R-type	I-type	J-type	Total
5	25	12	6	40	16	27	66	6	99
7	33	16	8	56	20	35	90	8	133
10	45	22	11	80	26	47	126	11	184
13	57	28	14	104	32	59	162	14	235
15	65	32	16	120	36	67	186	16	269

20	85	42	21	160	46	87	246	21	354
25	105	52	26	200	56	107	306	26	439

Table 5. Q-Matrix Method: instruct. count of different type vs. order

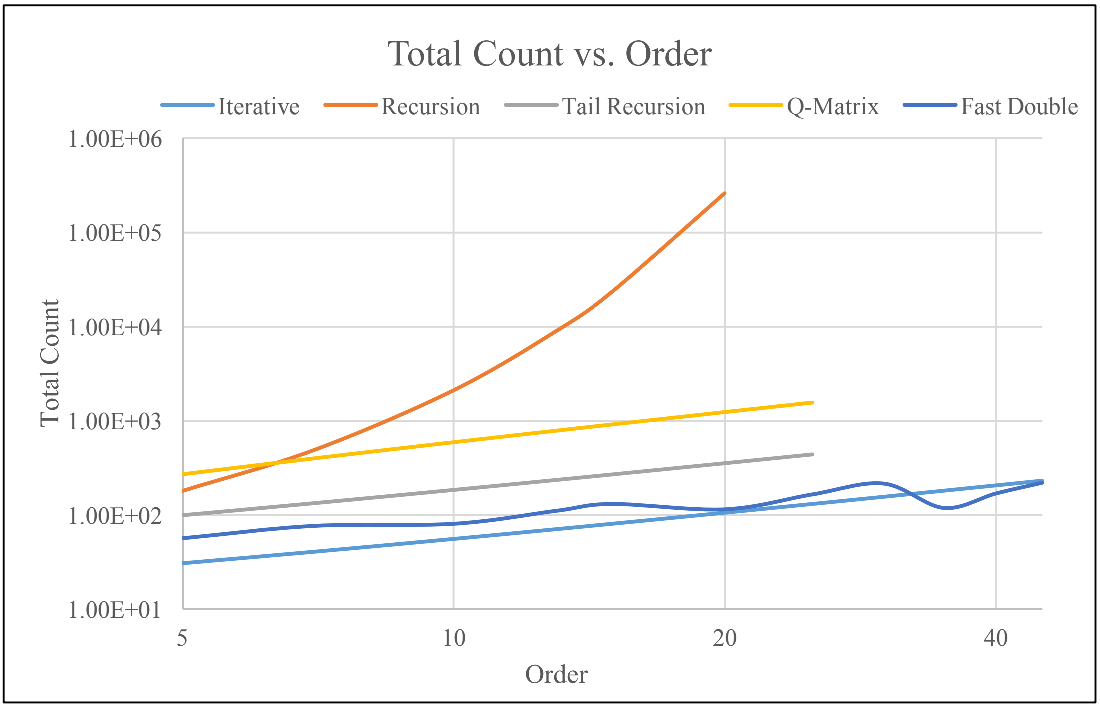
order	ALU	Jump	Branch	Memory	Other	R-type	I-type	J-type	Total
5	82	26	10	89	64	113	145	13	271
7	120	38	14	133	94	167	213	19	399
10	177	56	20	199	139	248	315	28	591
13	34	74	26	265	184	329	417	37	783
15	272	86	30	309	214	383	485	43	911
20	367	116	40	419	289	518	655	58	1231
25	462	146	50	529	364	653	825	73	1551

Table 6. Fast Doubling Method: instruct. count of different type vs. order

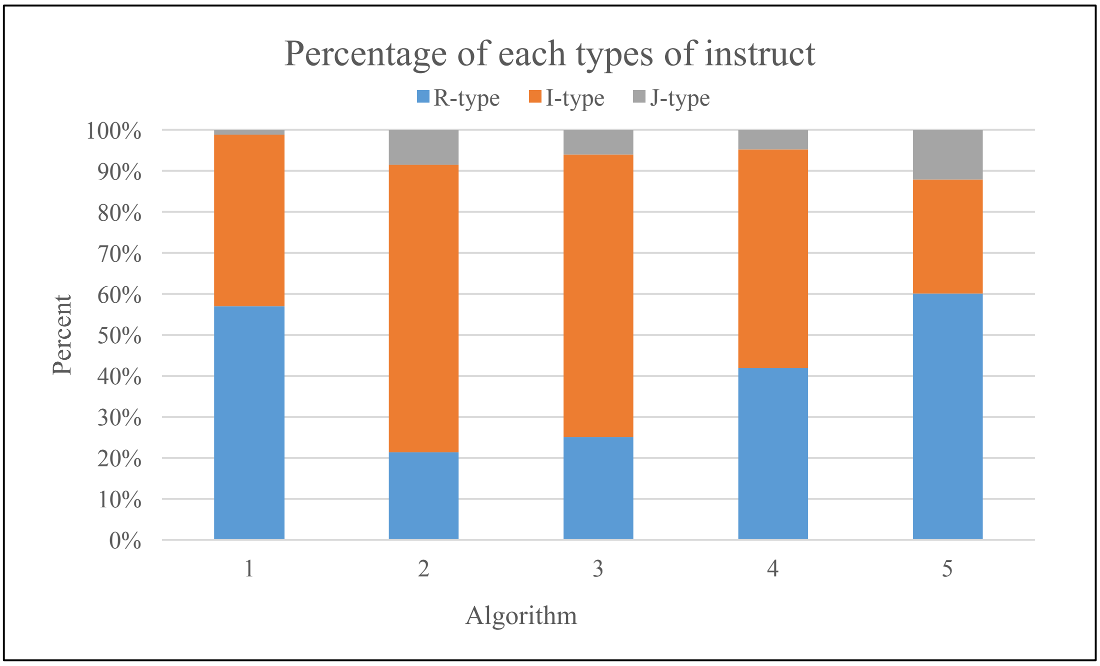
order	ALU	Jump	Branch	Memory	Other	R-type	I-type	J-type	Total
5	15	4	9	0	29	36	17	4	57
7	19	8	13	0	37	46	23	8	77
10	19	7	13	0	42	52	22	7	81
13	25	13	19	0	54	67	31	13	111
15	29	17	23	0	62	77	37	17	131
20	25	12	19	0	59	73	30	12	115
25	35	22	29	0	79	98	45	22	165
30	45	32	39	0	99	123	60	32	215
35	25	11	19	0	64	79	29	11	119

40	35	21	29	0	84	104	44	21	169
45	45	31	39	0	104	129	59	31	219

Graph 1. Total Count vs. Order



Graph 2. Percentage of Instruction type vs. algorithm



The Fibonacci number we used in this analysis is solved in Table 1. By comparing the result obtained from MIPS code and C code, we verify that answers from all the MIPS programs are correct. Thus, we use the statistic tool and count tool provided by MARS to count the number of instruction used in each algorithm and the type of instruction as well. The results are shown as tables from Table 2 to Table 6, where each table belong to an algorithm. For each table, we record the instruction number corresponding to the order of the Fibonacci number. The orders are basically chosen to be 5, 7, 10, 13, 15, 20, and 25. However, the recursion algorithm passes 25 since it took too much time to calculate the number of instruction and the number while the program was terminated reached up to more than 40,000. It is significantly larger than all the other algorithm under the same order condition. Therefore, the remaining term is discarded to save time. As for algorithm A and algorithm E, i.e. Iterative Method and Fast Doubling Method, are tested up to order of 30. The result is that both of them took less execution instruction than others. In addition, the total counts of Fast Doubling Method sometimes drop dramatically. To see the overall trend, we plot to the order of 45 since the order larger than this value will overflow.

To view the result more intuitively, we plot the count vs. order as Graph1. The y-axis is displayed in logarithm scale. From this graph, it is obvious that the complexity of Recursive Method, or say algorithm B, is much worse than all the other methods. The Iterative Method, Tail Recursion Method, and Q-Matrix Method have the same order of complexity because their slopes in Graph 1 are almost the same as each other with only the y-shifting value difference. The weird performance of Fast Doubling Method is clear in Graph 1 as well. The dark blue line ripples as the order increases, which allows the algorithm sometimes performs better than Iterative Method.

The ripple of Fast Doubling Method arises from the way we calculate iterative step. It doubles the iteration index until it approaches to the order we are going to solve. Thus, it saves a lot of instructions compared to algorithm B, C, and D.

Graph 2 shows the percentage of each type of instruction of different algorithm in average. Iterative Method applies J-type instruction less than 1% in average since it only needs J-type instruction to jump back to the main function in this design. Algorithm B and E use the most J-type algorithm in average due to change of function set. I-type instructions are used most often in both recursive methods. Recursion-type methods need to call itself multiple times during each execution. Hence, it needs plenty of load and store instruction, which belong to I-type instruction. In addition, due to a lots of memory-related instructions used in recursive methods, the execution time must increase since the memory-related instructions often have higher CPI than all the other instructions.

Conclusion

From the above analysis, we verify two things. First, the Recursive Method takes much more time and memory complexity than Iterative Method does. Although the Recursive Method looks more elegant in visual. Second, the Fast Doubling Method does help reduction of instruction when the order of Fibonacci number increases. In brief, the best way to calculate Fibonacci number is Iterative Method and Fast Doubling Method.

This project enforces me to get familiar with MIPS assembly code. By comparison, C code is much easier to read and write. Not surprising that we, human, developed instruction interface to help us convert high-level language like C into assembly language to save programming time and enhance readability of the overall program.