8XC251SB BENCHMARK REPORT

OBJECTIVE

The purpose of this benchmark was to check the performance of the CPU power of 8XC251SB Vs 8XC51FX with different programs and different hardware.

SUMMARY OF BENCHMARK

A total of 8 programs were written for four experiments: four in pure 51 instructions and four in optimized 251 instructions. Out of the four types of programs, the first three programs are 64 bytes Data Transfer, Multiply and Accumulation, and 3x3 Matrix Multiplication. The last program is a combination of the first three programs to acquire the overall performance of the microcontrollers. The source for the first program, which performs the 64 bytes of Data Transfer, is included in Appendix A.

All programs were assembled in two modes: binary mode and source mode. Programs with 51 instructions assembled in binary mode are 8XC51FX compatible and can be run by all units. Programs with 51 instructions assembled in source mode and programs with 251 instructions in both binary and source mode can only be run by 87C251SB microcontrollers.

Experiment 1, 2 and 3 will be run on the EV80C51FX evaluation board while experiment 4 was conducted on a the 8XC51FX target board.

APPARATUS

Experiment 1, 2 and 3

- EV80C51FX evaluation board
- 12 MHz 87C51FB
- 12 MHz 87C251SB
- Texas Instruments TMS27C256 100ns 32K EPROM
- Digital stop watch

Experiment 4

- 8XC51FX target board
- 12 MHz 87C51FB
- 12 MHz 87C251SB
- Intel D27C256 120ns 32K EPROM
- Digital stop watch

PROCEDURE

- In each experiment, the test programs were programmed into external eprom (27C256) and internal eprom of the 87C251SB or the 87C51FB.
- For programs that run externally on eproms, all internal eproms of the microcontrollers were left blank.
- The configuration bytes of the 87C251SB microcontrollers were programmed accordingly. For example, addresses 80 & 81 of the 87C251SB controller were programmed as FE and EF to enable binary mode, 0 wait states, and non page-mode.
- Digital stop watch was switched on once the reset button of the target board was pressed, and stopped when the 5 of the display LEDs were turned on.
- 3 readings were taken to obtain the average processing time. The 87C51FB was taken as a reference to compare the processing speed of all other microcontrollers.
- Each program was analyzed thoroughly to obtain the exact state percentage of the type of the instructions. Graphs were plotted for comparison and to give a brief idea for readers to understand the advantage of new 251 instruction code.

EXPERIMENT 1

This experiment compares the processing speed of the 8XC51FX to the 8XC251SB by emulating data transfer from internal code memory to external data memory. The programs used are shown below:-

i) TMAC11B with pure 51 instructions assembled in binary mode.
ii) TMAC11S with pure 51 instructions assembled in source mode.
iii) TMAC12B with 251 new instructions assembled in binary mode.
iv) TMAC12S with 251 new instructions assembled in source mode.

The flow of the programs is shown below:-

1) LEDs of the evaluation board were displayed once (the pattern was 10000001)
2) The benchmarking routine was looped for 31 times
3) The LEDs were displayed at the end of every loop (the pattern was 100XXXXX)

In every loop of TMAC11B, TMAC11S, TMAC12B and TMAC12S, the following tasks were performed:-

	Task	Instruction Type
a.	Loop 3825 times. In each loop move 64 bytes of constant data from internal code memory to external data memory.	CPU
b.	Flashing the LED through port 1.	I/O

The results of the benchmarking are shown as follows:-

A) MCS 51 COMPATIBILITY

TMAC11B is written in 100% 51 instructions and assembled in Binary Mode

TMAC11S is written in 100% 51 instructions and assembled in Source Mode

Unit	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave	Ratio to FX
							1	2	3		
1	87C51FB	-	Ext	-	-	TMAC11B	3:40	3:40	3:40	3:40	1x
2	87C251SB	Bin	Ext	0	non-page	TMAC11B	1:18	1:18	1:18	1:18	2.82x
3	87C251SB	Bin	Ext	1	non-page	TMAC11B	1:51	1:51	1:51	1:51	1.98x
4	87C251SB	Src	Ext	0	non-page	TMAC11S	1:42	1:42	1:42	1:42	2.16x
5	87C251SB	Src	Ext	1	non-page	TMAC11S	2:27	2:27	2:27	2:27	1.50x
6	87C251SB	Bin	Int	0	non-page	TMAC11B	0:44	0:44	0:44	0:44	5x
7	87C251SB	Src	Int	0	non-page	TMAC11S	0:57	0:57	0:57	0:57	3.86x

TABLE 1.1

B) MCS 251 OPTIMIZATION

TMAC11B is written in 100% 51 instructions and assembled in Binary Mode

TMAC12B is written in optimized 251 instructions and assembled in Binary Mode

TMAC12S is written in optimized 251 instructions and assembled in Source Mode

Unit	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave	Ratio to FX
							1	2	3		
1	87C51FB	-	Ext	-	-	TMAC11B	3:40	3:40	3:40	3:40	1x
2	87C251SB	Bin	Ext	0	non-page	TMAC12B	0:31	0:31	0:31	0:31	7.10x
3	87C251SB	Bin	Ext	1	non-page	TMAC12B	0:43	0:43	0:43	0:43	5.12x
4	87C251SB	Src	Ext	0	non-page	TMAC12S	0:27	0:27	0:27	0:27	8.15x
5	87C251SB	Src	Ext	1	non-page	TMAC12S	0:37	0:37	0:37	0:37	5.96x

6	87C251SB	Bin	Int	0	non-page	TMAC12B	0:14	0:14	0:14	0:14	15.71x
7	87C251SB	Src	Int	0	non-page	TMAC12S	0:12	0:12	0:12	0:12	18.33x

TABLE 1.2

Device	Mode	Programs	Code Size	Code Size Ratio to 87C51FB
87C51FB	Binary	TMAC11B	78	1.00
87C251SB	Binary	TMAC11B	78	1.00
87C251SB	Source	TMAC11S	94	1.21
87C251SB	Binary	TMAC12B	81	1.04
87C251SB	Source	TMAC12S	84	1.08

TABLE 1.3: Code Size Difference

TMAC11B (In Pure 51 instructions)		
Types of Instructions	**Number of States**	**Percentage (%)**
I/O (Flashing LEDs)	46,326	0.12
Data Transfer (MOV etc.)	30,061,458	73.97
Branch (LJMP, LCALL etc.)	3,101,130	7.63
Arithmetic (ADD, DEC etc.)	7,428,588	18.28
Total	40,637,502	100

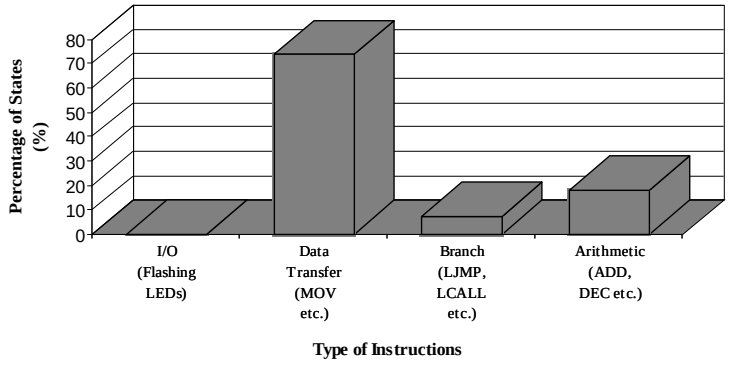
TABLE 1.4: Number and Percentage of States in One Complete Loop

TMAC12S (In Optimized 251 instructions)		
Types of Instructions	**Number of States**	**Percentage (%)**
I/O (Flashing LEDs)	7,722	0.39

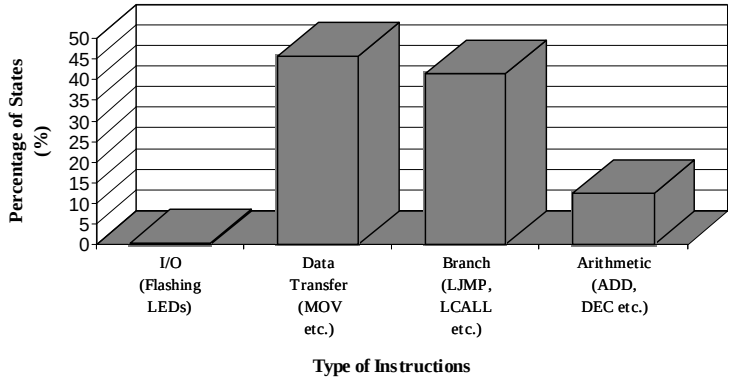
Data Transfer (MOV etc.)	894,858	45.58
Branch (LJMP, LCALL etc.)	813,782	41.45
Arithmetic (ADD, DEC etc.)	246,849	12.58
Total	1,963,211	100

TABLE 1.5: Number and Percentage of States in One Complete Loop

Graph 1.1: State Percentage of the Intructions Used in Data Transfer Program (In Pure 51 Instructions or TMAC11B)



Graph 1.2: State Percentage of the Instructions Used in Data Transfer Program (In Optimised 251 Instructions or TMAC12S)



DISCUSSION

Table 1.4 & Graph 1.1: State Percentage of the Instructions (In Pure 51 Instructions of TMAC11B)

Graph 1.1 shows the percentage of instruction states in one complete loop of the program to display the LED once (note that the LEDs were displayed 31 times in the experiment). The number states calculated for one complete loop of the program (which contains 3825 loops of moving 64 bytes of constant data from internal code memory to external data memory) is 40,637,502 states. For data transfer instructions, the percentage of states is 73.97%, branches is 7.63%, arithmetic operations take 18.28% while I/O instructions is only 0.12%. From this we know that data transfer instructions were mainly focused in this experiment.

Table 1.5 & Graph 1.2: State Percentage of the Instructions (In Optimized 251 Instructions of TMAC12S)

In the program written in 251 instructions and assembled in source mode, the total number states for one complete loop are greatly reduced to 1,747,219, which is 20 times less than the 51 instruction's states. Here, data transfer instructions take 45.58%, branch instructions take 41.45%, arithmetic operations is 12.58% while I/O instructions take only 0.39%. The states of data transfer instructions such as MOV are greatly reduced here. Take instruction MOV R0, A for example: it takes 6 states to run with 51 instructions but only 2 states with 251 instructions. This gives an approximately 3 times reduction of states in data transfer instructions. However, branch instructions such as LJMP and LCALL are not reduced as dramatically as data transfer instructions. For example, LCALL takes 12 states with 51 instructions but 9 states with 251 instructions. The reduction of states is only 1.3 times. Therefore, if we refer to graph 1.2, the state percentage of branch instructions (e.g. LJMP, LCALL) is quite high compared to the state percentage in graph 1.1.

In view of the reduction of states for every instruction type compared to 51 instructions, data transfer instructions have been reduced from 30,061,458 states to 894,858 states, which is approximately a 33 times reduction! For arithmetic instructions, the reduction is also 30 times. However, branch instructions only have a 4 times reduction of states. This means if the program is looped, 3825 loops to display an LED once in this case, the actual performance of the CPU is affected. As what have been mentioned, the total average number of states is only 20 times less when using 251 instructions. A better performance of the 87C251SB microcontroller is expected for a single loop of the program.

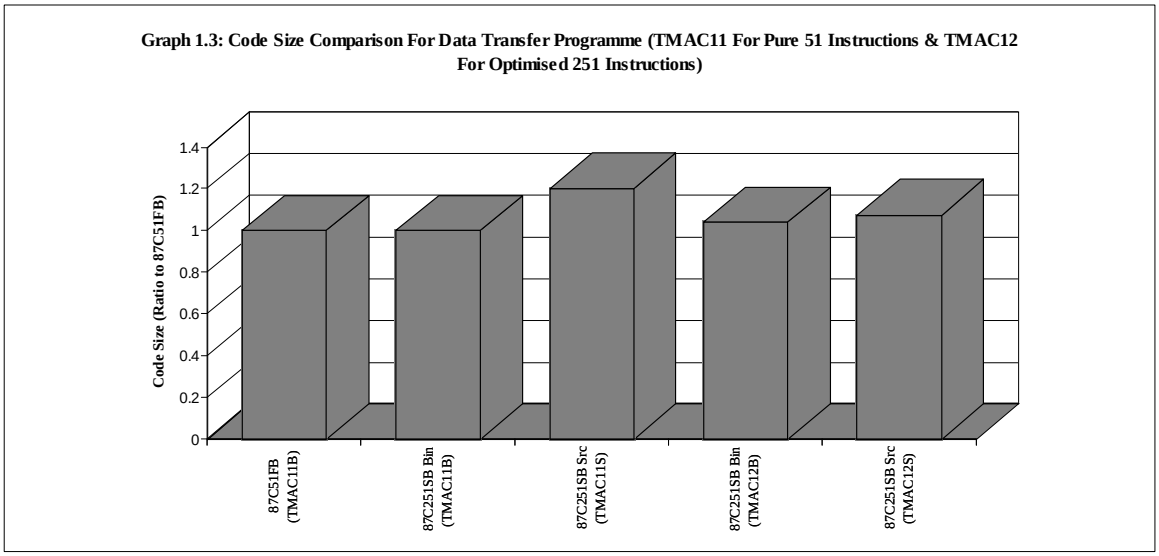


Table 1.3 & Graph 1.3: Code Size Comparison

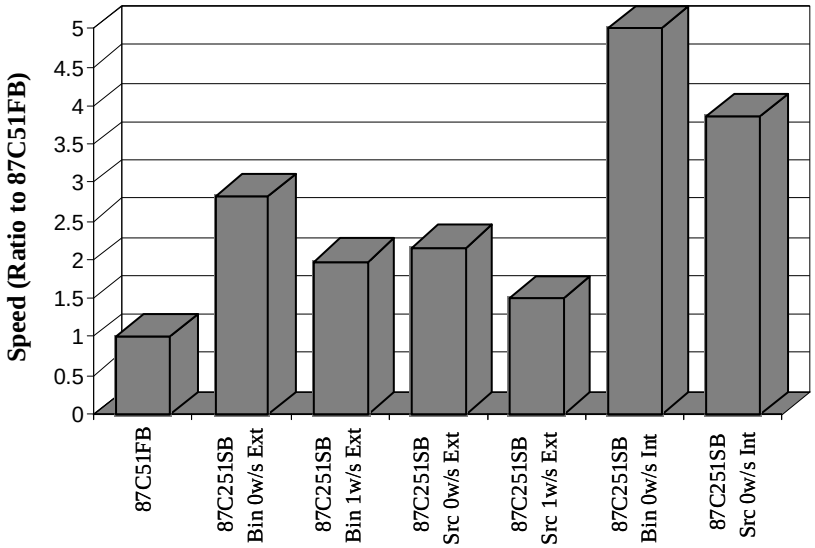
Code size is obtained by calculating the instruction code bytes in the program neglecting the definition of constants. Briefly, when a program is assembled in binary mode, code size of the 51 instructions will be less than the 251 instructions, where more bytes will be added to the 251 instructions. However, if a program is assembled in source mode, code size of the 51 instructions will be more than 251 instructions as more bytes will be added to the 51 instructions..

Graph 1.3 shows that the code size of all programs in pure 51 instructions that were run in experiment 1. From the graph, we can see that the code size of TMAC11S (51 instructions assembled in source mode) is 1.21 times larger than the TMAC11B (51 instructions assembled in binary mode), which is correct as more bytes were added to the same 51 instructions assembled in binary mode.

The code size of TMAC12B (251 instructions, binary mode) is expected to be less than TMAC11S and TMAC11B as most of the instructions were reduced in the program using the new 251 instructions. However, the graph shows the opposite result. This is actually due to the instructions used in the program. Although the number of instructions were reduced in TMAC12B, most of the 251 data transfer instructions are related to word register (WRj), which consume the code size. As the number of instructions related to word register increases, the code size is increased. This program, whose function is to transfer data, uses quite a number of word registers and therefore the code size is more than TMAC11B

For TMAC12S (251 instructions, source mode), an opposite result is obtained again. This is again due to the instructions. Registers were used in the program and the assembler takes the instructions related to the registers as 51 instructions. Therefore the code bytes are added and the code size is larger.

Graph 1.4: Speed Comparison For Data Transfer Program In 51 Instructions (TMAC11)



Graph 1.5: Speed Comparison For Data Transfer Program In Optimised 251 Instructions (TMAC12)

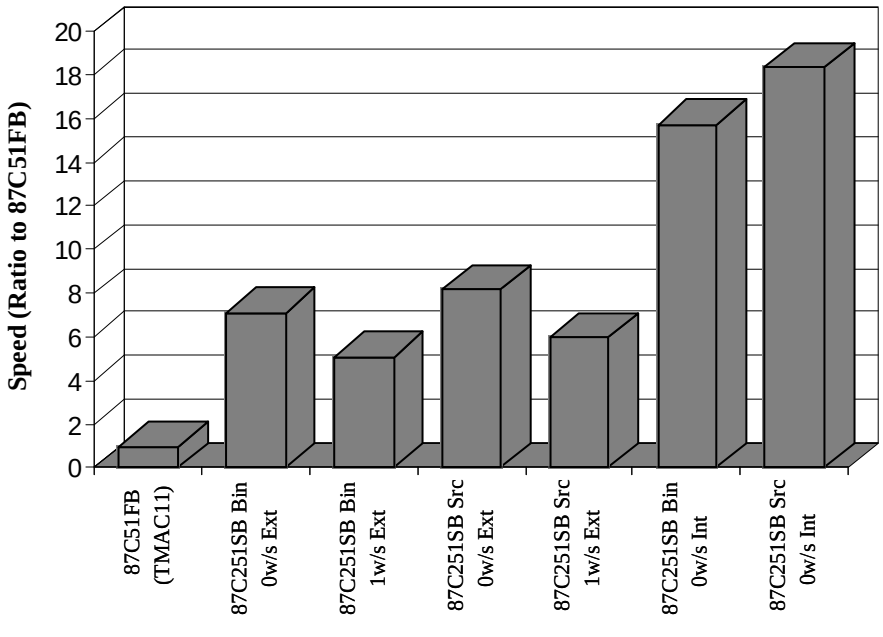


Table 1.1 & Graph 1.4: MCS 51 Compatibility

This part of the experiment is to compare the processing speed of a direct plug in of the 87C251SB microcontroller to the 87C51FB microcontroller.

In performing the data transfer program written in 51 instructions, the 87C251SB microcontroller is 2.82 times faster than the 87C51FB. Though, for programs that were run in the internal eprom of the 87C251SB, the processing speed of a binary mode 87C251SB is increased to 5 times that of the 87C51FB, and 3.86 times faster when in source mode. The 87C251SB microcontroller runs slower in source mode than binary mode due to the 51 instructions prefixed by extra code bytes, which extend the processing time.

Table 1.2 & Graph 1.5: MCS 251 Optimization

A better performance from the 87C251SB microcontroller is obtained from the optimized 251 instruction programs. From the results, running on external eprom, the 87C251SB microcontroller in source mode, 0 wait state was the fastest with processing speed 8.15 times faster than the 87C51FB microcontroller. For programs that were run on internal eprom of the 87C251SB microcontroller, configured in source mode with 0 wait states, the processing speed is 18.33 times faster than an ordinary 87C51FB! This has supported the analysis result obtained from the program that showed the total state reduction of approximately 20 times.

For programs that run on binary mode, the processing speed is slightly slower. This is again due to the additional code bytes that were added by the assembler to the 251 instructions and therefore leads to a delay of processing time.

CONCLUSION

From the experiment, we can conclude that the 87C251SB microcontroller can be optimized to maximum processing speed by using 251 instructions assembled in source mode. Minimum usage of instructions related to word registers and branching instructions will even lead to a faster processing speed.

EXPERIMENT 2

This experiment compares the processing speed of the 8XC51FX to the 8XC251SB by performing Multiplication and Accumulation (MAC) routines on 16 bits signed integer with 32 bits results. The programs used are shown below:

i) TMAC21B with pure 51 instructions assembled in binary mode.
ii) TMAC21S with pure 51 instructions assembled in source mode.
iii) TMAC22B with 251 new instructions assembled in binary mode.
iv) TMAC22S with 251 new instructions assembled in source mode.

The flow of the programs is shown below:-

1) LEDs of the evaluation board were displayed once (the pattern was 10000001) at the beginning
2) The benchmarking routine was looped 31 times
3) The LEDs were displayed at the end of every loop (the pattern was 100XXXXX)

In every loop of TMAC21B, TMAC21S, TMAC22B and TMAC22S, the following tasks were performed:-

	Task	Instruction Type
a.	Loop 65,025 times a 16 bit Multiplication and Accumulation (MAC)	CPU
b.	Flashing the LED through port 1.	I/O

The results of the benchmarking are shown as follows:-

A) MCS 51 COMPATIBILITY

TMAC21B is written in 100% 51 instructions and assembled in Binary Mode

TMAC21S is written in 100% 51 instructions and assembled in Source Mode

Unit	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave	Ratio to FX
							1	2	3		
1	87C51FB	-	Ext	-	-	TMAC21B	4:17	4:17	4:17	4:17	1x
2	87C251SB	Bin	Ext	0	non-page	TMAC21B	1:42	1:41	1:42	1:42	2.52x
3	87C251SB	Bin	Ext	1	non-page	TMAC21B	2:29	2:29	2:29	2:29	1.72x
4	87C251SB	Src	Ext	0	non-page	TMAC21S	2:12	2:12	2:12	2:12	1.95x
5	87C251SB	Src	Ext	1	non-page	TMAC21S	3:15	3:15	3:15	3:15	1.32x
6	87C251SB	Bin	Int	0	non-page	TMAC21B	0:57	0:57	0:57	0:57	4.51x

7	87C251SB	Src	Int	0	non-page	TMAC21S	1:12	1:12	1:12	1:12	3.57x

TABLE 2.1

B) MCS 251 OPTIMIZATION

TMAC21B is written in 100% 51 instructions and assembled in Binary Mode

TMAC22B is written in optimized 251 instructions and assembled in Binary Mode

TMAC22S is written in optimized 251 instructions and assembled in Source Mode

Unit	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave	Ratio to FX
							1	2	3		
1	87C51FB	-	Ext	-	-	TMAC21B	4:17	4:17	4:17	4:17	1x
2	87C251SB	Bin	Ext	0	non-page	TMAC22B	1:09	1:09	1:09	1:09	3.72x
3	87C251SB	Bin	Ext	1	non-page	TMAC22B	1:39	1:39	1:39	1:39	2.60x
4	87C251SB	Src	Ext	0	non-page	TMAC22S	1:00	1:00	1:00	1:00	4.28x
5	87C251SB	Src	Ext	1	non-page	TMAC22S	1:25	1:25	1:25	1:25	3.02x
6	87C251SB	Bin	Int	0	non-page	TMAC22B	0:36	0:36	0:36	0:36	7.14x
7	87C251SB	Src	Int	0	non-page	TMAC22S	0:32	0:32	0:32	0:32	8.03x

TABLE 2.2

Device	Mode	Programs	Code Size	Code Size Ratio to 87C51FB
87C51FB	Binary	TMAC21B	150	1.00
87C251SB	Binary	TMAC21B	150	1.00
87C251SB	Source	TMAC21S	198	1.32
87C251SB	Binary	TMAC22B	92	0.61
87C251SB	Source	TMAC22S	82	0.55

TABLE 2.3: Code Size Difference

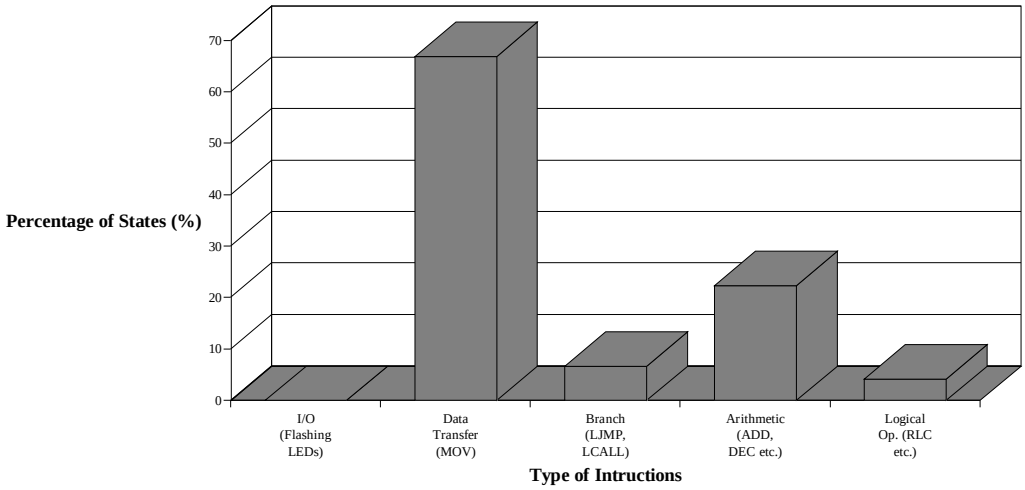
TMAC21B (In Pure 51 instructions)		
Types of Instructions	**Number of States**	**Percentage (%)**
I/O (Flashing LEDs)	30	6.28E-05
Data Transfer (MOV etc.)	31,976,448	66.94
Branch (LJMP, LCALL etc.)	3,161,112	6.62
Arithmetic (ADD, DEC etc.)	10,658,310	22.31
Logical Operations (RLC etc.)	1,973,760	4.13
Total	47,769,660	100

TABLE 2.4: Number and Percentage of States in One Complete Loop

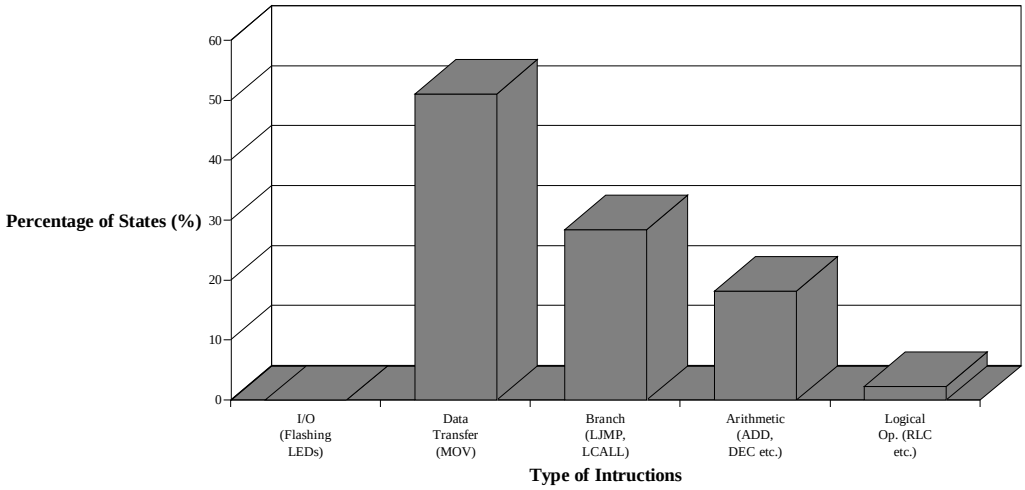
TMAC22S (In Optimized 251 instructions)		
Types of Instructions	**Number of States**	**Percentage (%)**
I/O (Flashing LEDs)	8	0.000138
Data Transfer (MOV etc.)	2,961,154	51.13
Branch (LJMP, LCALL etc.)	1,645,574	28.42
Arithmetic (ADD, DEC etc.)	1,052,672	18.18
Logical Operations (RLC etc.)	131,584	2.27
Total	5,790,992	100

TABLE 2.5: Number and Percentage of States in One Complete Loop

Graph 2.1: State Percentage of the Instructions Used in Multiplication and Accumulation Programme (In Pure 51 Instructions or TMAC21B)



Graph 2.2: State Percentage of the Instructions Used in Multiplication and Accumulation Programme (In Optimised 251 Intructions or TMAC22S)



DISCUSSION

Table 2.4 & Graph 2.1: State Percentage of the Instructions (In Pure 51 Instructions of TMAC21B)

From table 2.4, total counted states are 47,769,660 states in one complete loop. In these states, data transfer instructions dominated with a percentage of 66.94%, followed by arithmetic instructions with 22.31%. The state percentage of branch and logical operations is 6.62% and 4.13% respectively. I/O instructions can be neglected for its low percentage of states.

Referring to the exact program attached at the appendix, for multiply and accumulate routines, we can see that the data has to be transferred to the data pointer for arithmetic purposes. This is why data transfer instructions consume more states than arithmetic instructions in this program.

Table 2.5 & Graph 2.2: State Percentage of the Instructions (In Optimized 251 Instructions of TMAC22S)

For the multiply and accumulate program written in 251 instructions and assembled in source mode, total states have been reduced to 5,790,992. Percentage of data transfer is 51.13%, followed by branch instructions which is 28.42% and logical operations, 2.27%. I/O instructions is again neglected for a small percentage.

Total state reduction is 8.25 times that of the 51. This result is less reduction than in the previous experiment due to the way of the program was written. The number of states of arithmetic instructions have been reduced from 10,658,310 to 1,052,672 states, which is an approximate 10 times reduction. The same percentage of reduction is also obtained from the data transfer instructions. For logical operations, the percentage of state reduction is 15 times. However, the branch instructions only have a state reduction of 1.92 times. Due to the multiple loops of this test program—that is 65,025 loops done to display the LED once—the low reduction of states is greatly influenced by the preponderance of branch instructions, and thus the optimum performance of the 87C251SB microcontroller is affected.

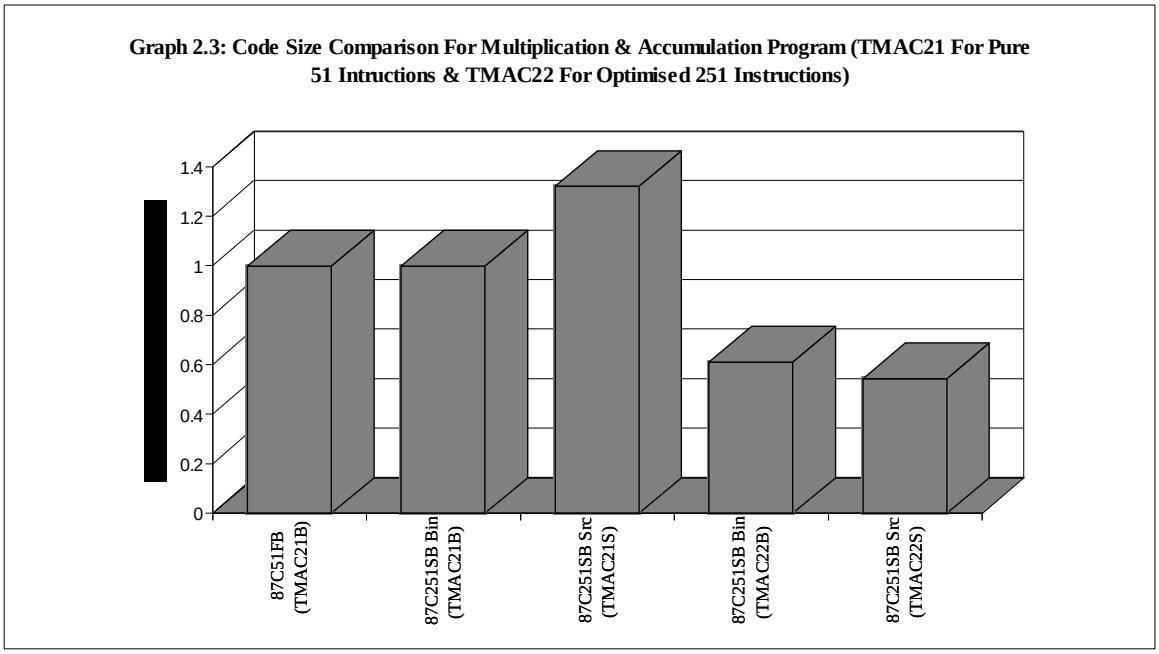
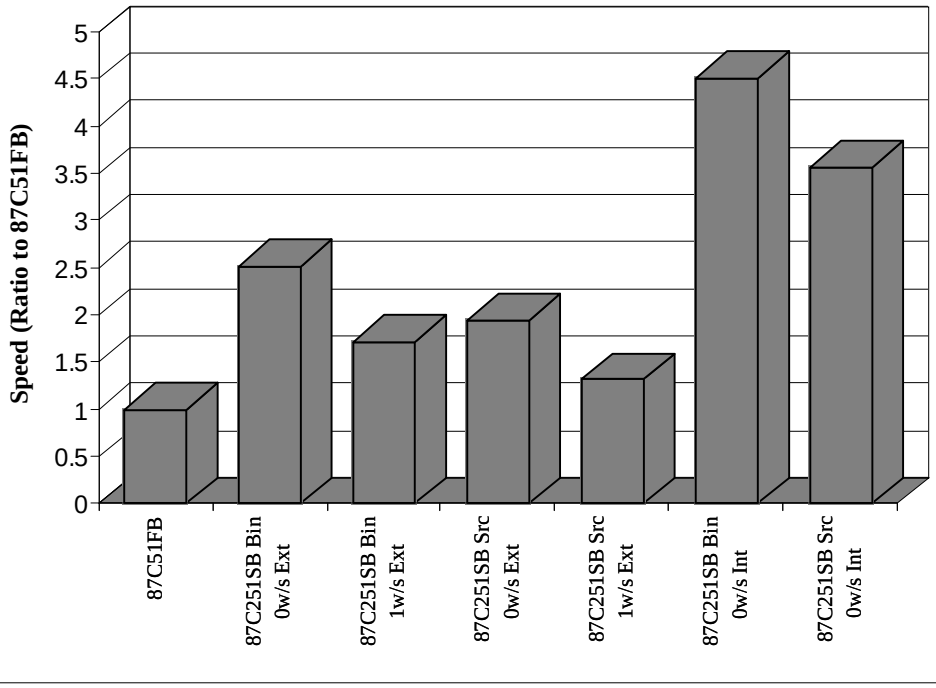


Table 2.3 & Graph 2.3: Code Size Comparison

From Graph 2.3, a significant reduction of code size is clearly shown here. Table 2.3 gives the exact figure of the code size in the test program.

For the programs written in 51 instructions and assembled in source mode, due to the additional byte added by the assembler, the code size is 1.32 times more than the same program assembled in binary mode. For the programs written in optimized 251 instructions and assembled in source mode, the code size is smaller due to great reduction of the instructions. However, the assembler still added some additional bytes for certain 251 instructions, thus the code size is more than the same program assembled in source mode.

Graph 2.4: Speed Comparison For Multiplication and Accumulation Program In 51 Instructions (TMAC21)



Graph 2.5: Speed Comparison For Multiplication and Accumulation In Optimised 251 Instructions (TMAC22)

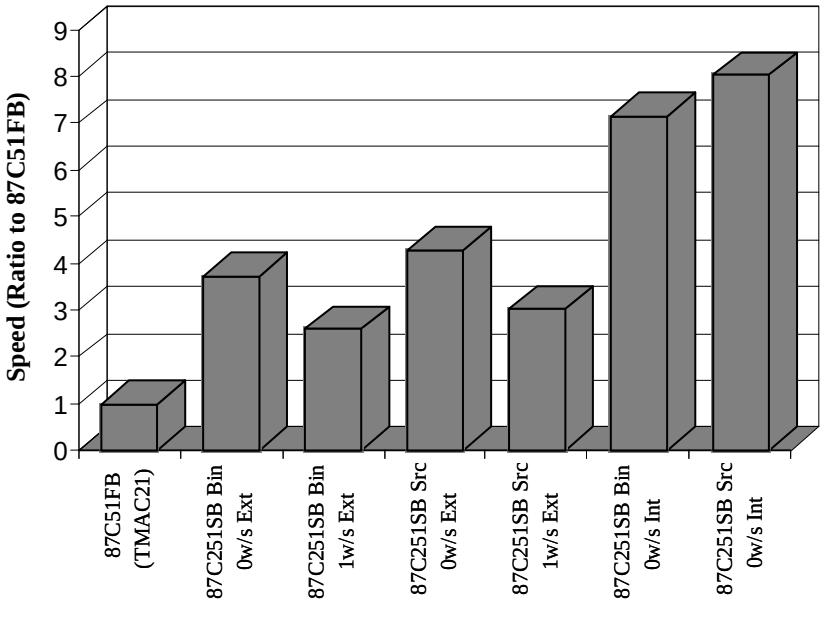


Table 2.1 & Graph 2.4: MCS 51 Compatibility

A same trend of speed performance as experiment 1 is obtained here.

Table 2.2 & Graph 2.5: MCS 251 Optimization

From the results, for programs executed in external eprom, the 87C251SB microcontroller executed fastest in source mode with 0 wait states -- 4.28 times faster than the 87C51FB microcontroller. For programs that were run on internal eprom of the 87C251SB microcontroller, the optimal configuration, the processing speed is 8.03 times faster than an ordinary 87C51FB.

The results from the program analysis correlates to the results obtained from the experiment. The recorded optimum execution speed of 87C251SB is 8.03 times, and the expected speed calculated in the program analysis is 8.25 times. The reason for a slower experimental execution time for 87C251SB in this experiment was explained earlier, which is mainly due to the multiple loops in the program.

CONCLUSION

From the experiment, it can be predicted that the 87C251SB will perform better in executing larger programs than executing multiple loops of one small program.

EXPERIMENT 3

This experiment compares the processing speed of the 8XC51FX and 8XC251SB microcontrollers of performing 3x3 Matrix Multiplication on 16 bit signed integers with 32 bit results. The programs used are shown below:-

i) TMAC31B with pure 51 instructions assembled in binary mode.
ii) TMAC31S with pure 51 instructions assembled in source mode.
iii) TMAC32B with 251 new instructions assembled in binary mode.
iv) TMAC32S with 251 new instructions assembled in source mode.

The flow of the programs is shown below:-

1) LEDs of the evaluation board were displayedonce (the pattern was 10000001) at the beginning
2) The benchmarking routine was looped 31 times.
3) The LEDs were displayed at the end of every loop (the pattern was 100XXXXX)

In every loop of TMAC31B, TMAC31S, TMAC32B and TMAC32S, the following tasks were performed:-

	Task	Instruction Type
a.	Loop 3825 times the 16 bit 3 x 3 Matrix Multiplication	CPU
b.	Flashing the LED through port 1.	I/O

The results of the benchmarking are shown as follows:-

A) MCS 51 COMPATIBILITY

TMAC31B is written in 100% 51 instructions and assembled in Binary Mode

TMAC31S is written in 100% 51 instructions and assembled in Source Mode

Unit	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave	Ratio to FX
							1	2	3		
1	87C51FB	-	Ext	-	-	TMAC31B	6:58	6:58	6:59	6:58	1x
2	87C251SB	Bin	Ext	0	non-page	TMAC31B	3:41	3:41	3:41	3:41	1.89x
3	87C251SB	Bin	Ext	1	non-page	TMAC31B	5:25	5:25	5:26	5:25	1.29x
4	87C251SB	Src	Ext	0	non-page	TMAC31S	3:41	3:41	3:41	3:41	1.89x
5	87C251SB	Src	Ext	1	non-page	TMAC31S	5:25	5:25	5:25	5:25	1.29x

6	87C251SB	Bin	Int	0	non-page	TMAC31B	2:00	2:00	2:00	2:00	3.48x
7	87C251SB	Src	Int	0	non-page	TMAC31S	2:01	2:01	2:01	2:01	3.45x

TABLE 3.1

B) MCS 251 OPTIMIZATION

TMAC31B is written in 100% 51 instructions and assembled in Binary Mode

TMAC32B is written in optimized 251 instructions and assembled in Binary Mode

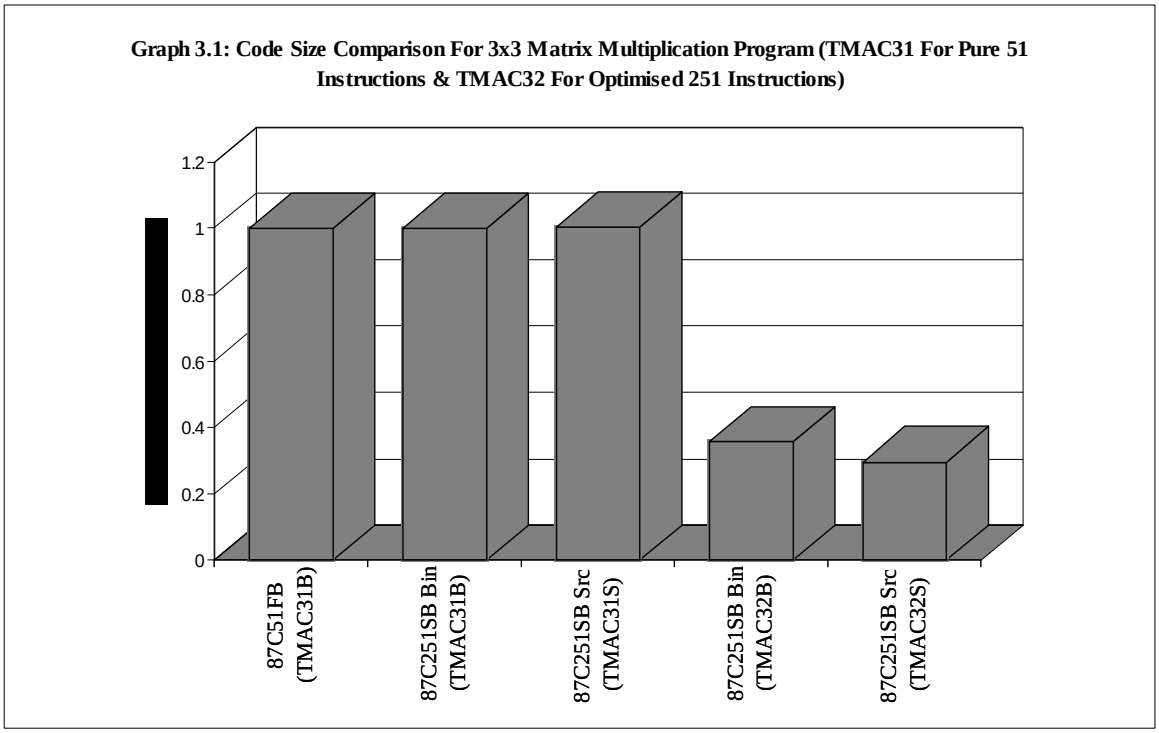
TMAC32S is written in optimized 251 instructions and assembled in Source Mode

Unit	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave	Ratio to FX
							1	2	3		
1	87C51FB	-	Ext	-	-	TMAC31B	6:58	6:58	6:59	6:58	1x
2	87C251SB	Bin	Ext	0	non-page	TMAC32B	0:59	0:59	0:59	0:59	7.08x
3	87C251SB	Bin	Ext	1	non-page	TMAC32B	1:23	1:23	1:23	1:23	5.04x
4	87C251SB	Src	Ext	0	non-page	TMAC32S	0:53	0:53	0:53	0:53	7.89x
5	87C251SB	Src	Ext	1	non-page	TMAC32S	1:13	1:13	1:13	1:13	5.73x
6	87C251SB	Bin	Int	0	non-page	TMAC32B	0:35	0:35	0:35	0:35	11.94x
7	87C251SB	Src	Int	0	non-page	TMAC32S	0:32	0:32	0:32	0:32	13.06x

TABLE 3.2

Device	Mode	Programs	Code Size	Code Size Ratio to 87C51FB
87C51FB	Binary	TMAC31B	1723	1.00
87C251SB	Binary	TMAC31B	1723	1.00
87C251SB	Source	TMAC31S	1731	1.005
87C251SB	Binary	TMAC32B	616	0.36
87C251SB	Source	TMAC32S	512	0.30

TABLE 3.3: Code Size Difference



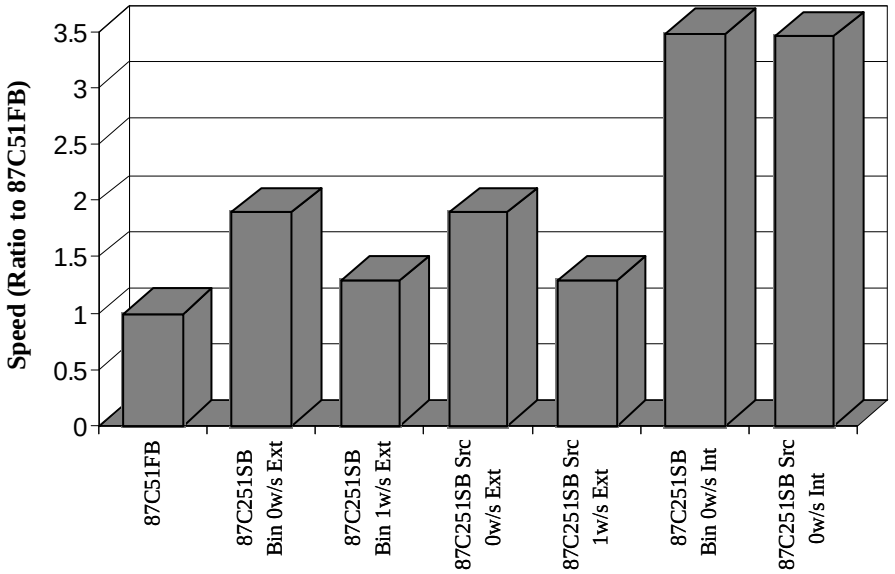
DISCUSSION

Table 3.3 & Graph 3.1: Code Size Comparison

From table 3.3, we see that the code size of the program written in 251 instructions assembled in source mode is significantly reduced up to 70% of the same program in 51 instructions. This tells that most of the 51 instructions was removed or replaced with only a few 251 instructions.

Other results indicated by Graph 3.3 are almost the same as previous experiments. Programs written in 51 instructions and assembled in source mode have larger code size than the same program assembled in binary mode and vice versa. The code size for TMAC31 in source mode is 1.005 larger than TMAC31 in binary mode, and the code size for TMAC32 in binary mode is 1.2 times larger than TMAC31 in source mode.

Graph 3.2: Speed Comparison For 3x3 Matrix Multiplication Program In 51 Instructions (TMAC31)



Graph 3.3: Speed Comparison For 3x3 Matrix Multiplication Program In Optimised 251 Instructions (TMAC32)

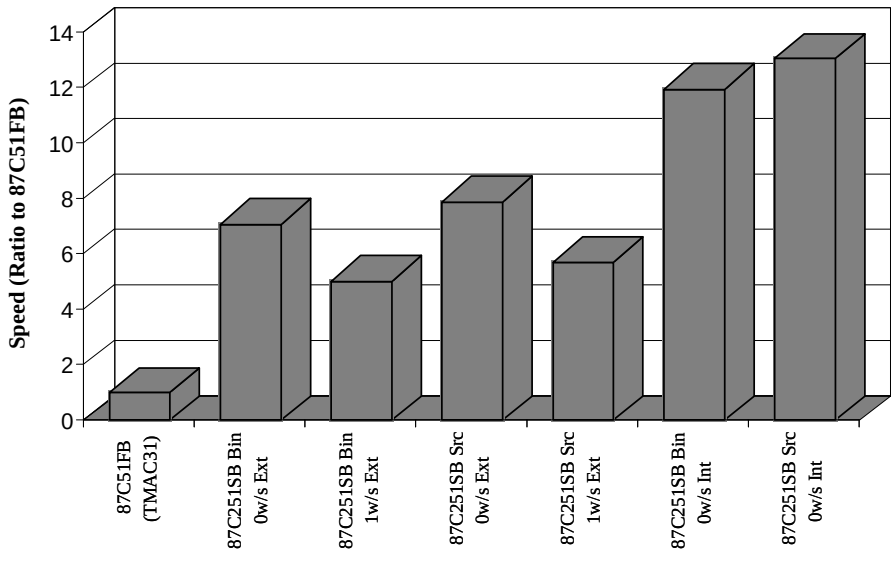


Table 3.1 & Graph 3.2: MCS 51 Compatible

A larger program with arithmetic operation and data transfer is tested in this experiment. For a direct plug-in, the 87C251SB microcontroller is still slightly faster than the 87C51FB by 1.89 times. For programs running in internal eprom of the 87C251SB, the speed is increased to 3.48 times faster the 87C51FB.

From table 3.1, there is no time difference between a source mode and a binary mode. This is due to the additional bytes added by the assembler. Since the assembler only add additional bytes to certain 51 instructions when assemble in source mode, less byte will be added if less of these instructions were used. This is what exactly happening in this program, where the time spent for executing the added bytes is little, perhaps in milliseconds. Therefore, the time difference is little between binary and source mode and it is not detected in the experiment.

Table 3.2 & Graph 3.3: MCS 251 Optimization

The size of this test program is slightly larger than the ones in experiment 1 and 2. The performance of the 87C251SB microcontroller is expected to be better as less loops were executed compared to program in experiment 2.

From the results, the optimum execution speed running on external eprom for 87C251SB is 7.89 times faster than the 87C51FB. For program executed in internal eprom of the 87C251SB, the processing speed is 13.06 times faster than the 87C51FB.

These results show that the 87C251SB performs as well as expected.

CONCLUSION

State percentage analysis was not done to the test program here as the instructions in this program are mainly the same as the programs in experiment 1 and 2.

From the results obtained from this experiment and the first two experiments, we can conclude that the execution speed for 87C251SB range from 2 to 3 times faster than the 87C51FB for a direct plug in running on external eprom and up to 5 times for running program at internal eprom.

For optimized case, the execution speed for 87C251SB range from 4 to 8 times faster than the 87C51FB and can be up to 18 times faster for program running at internal eprom.

EXPERIMENT 4

This experiment is to compares the processing speed of the 8XC51FX and 8XC251SB microcontrollers on performing a combined program consisting of emulation of Data transfer from internal code memory to external data memory, Multiplication and Accumulation (MAC) and 3x3 Matrix Multiplication on 16 bits signed integer. Experiment 4 was done with a different hardware: the 8XC51FX target board with page mode capability was used to test the performance of the new feature in 87C251SB. The programs used are shown below:-

i) TMAC1 with pure 51 instructions assembled in binary mode.
ii) TMAC2 with pure 51 instructions assembled in source mode.
iii) TMAC7 with 251 new instructions assembled in binary mode.
iv) TMAC8 with 251 new instructions assembled in source mode.

The flow of the TMAC1 is shown below:-

1) LEDs of the target board were displayed once (the pattern was 10000001) at the beginning
2) The benchmarking routine was looped for 31 times
3) The LEDs were displayed at the end of every loop (the pattern was 100XXXXX)

In every loop of TMAC1, the following tasks were performed:-

	Task	Instruction Type
a.	Loop 3825 times moving 64 bytes of constant data from internal code memory to external data memory	CPU
b.	Loop 65,025 times a 16 bit Multiplication and Accumulation (MAC)	CPU
c.	Loop 3825 times a 16 bit 3 x 3 matrix Multiplication	CPU
d.	Flashing the LED through port 1	I/O

The results of the benchmarking are shown as follows:-

A) MCS 51 COMPATIBILITY

TMAC1 is written in 100% 51 instructions and assembled in Binary Mode

TMAC2 is written in 100% 51 instructions and assembled in Source Mode

Unit #	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave.	Ratio to FX
							1	2	3		

1	87C51FB	-	Int	-	-	TMAC1	14:53	14:53	14:53	14:53	1x
2	87C51FB	-	Ext	-	-	TMAC1	14:53	14:53	14:53	14:53	1x
5	87C251SB	Bin	Ext	0	non-page	TMAC1	6:40	6:40	6:40	6:40	2.23x
6	87C251SB	Bin	Ext	1	non-page	TMAC1	9:45	9:45	9:45	9:45	1.53x
7	87C251SB	Src	Ext	0	non-page	TMAC2	7:35	7:35	7:35	7:35	1.96x
8	87C251SB	Src	Ext	1	non-page	TMAC2	11:07	11:07	11:07	11:07	1.34x
9	87C251SB	Bin	Ext	0	page	TMAC1	4:06	4:06	4:05	4:06	3.63x
10	87C251SB	Bin	Ext	1	page	TMAC1	6:57	6:57	6:57	6:57	2.14x
11	87C251SB	Src	Ext	0	page	TMAC2	4:35	4:35	4:35	4:35	3.25x
12	87C251SB	Src	Ext	1	page	TMAC2	7:52	7:51	7:51	7:51	1.90x
13	87C251SB	Bin	Int	0	non-page	TMAC1	3:41	3:41	3:41	3:41	4.04x
14	87C251SB	Src	Int	0	non-page	TMAC2	4:10	4:10	4:10	4:10	3.57x

Table 4.1

B) MCS 251 OPTIMIZATION

TMAC1 is written in 100% 51 instructions and assembled in Binary Mode

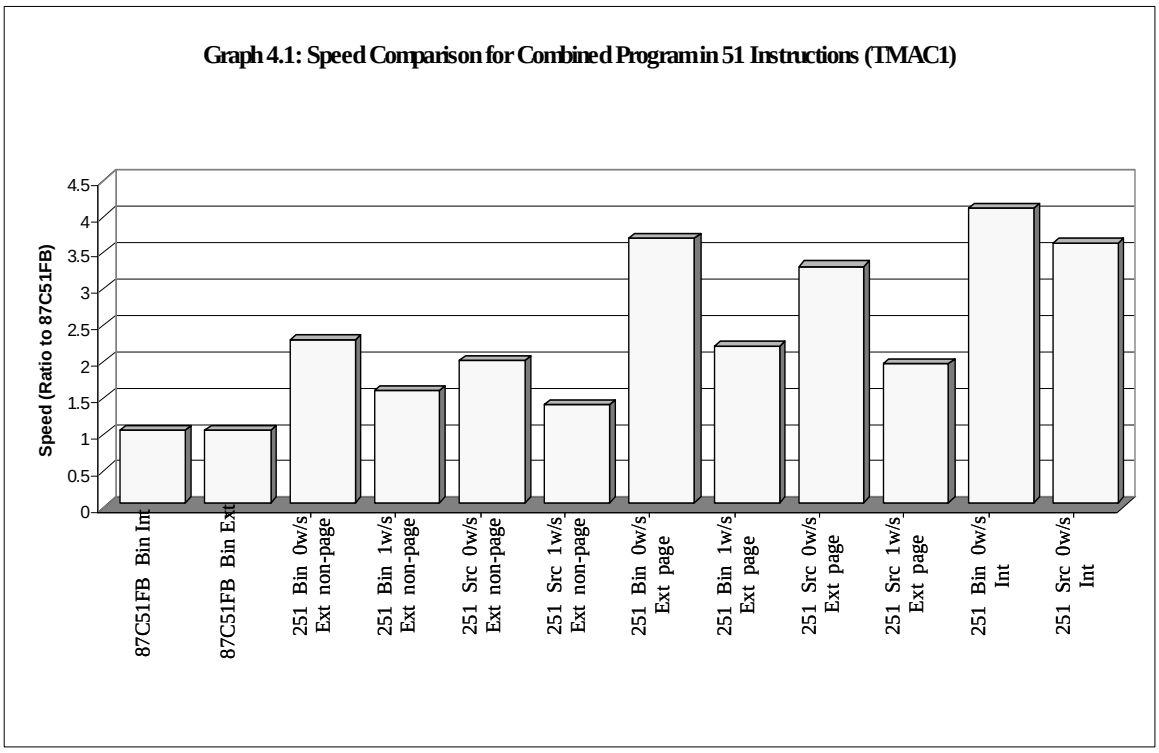
TMAC7 is written in optimized 251 instructions and assembled in Binary Mode

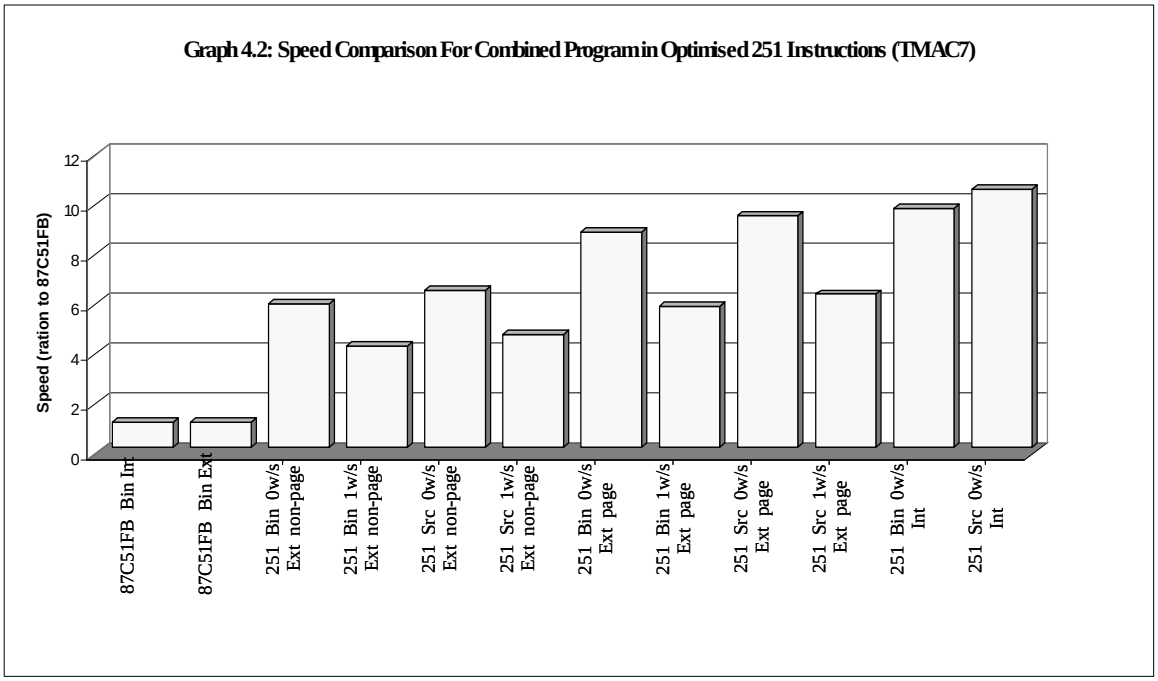
TMAC8 is written in optimized 251 instructions and assembled in Source Mode

Unit #	Device	Mode	Mem	W/s	Page	Prog	Time (Min:Sec)			Ave.	Ratio to FX
							1	2	3		
1	87C51FB	-	Int	-	-	TMAC1	14:53	14:53	14:53	14:53	1x
2	87C51FB	-	Ext	-	-	TMAC1	14:53	14:53	14:53	14:53	1x
3	87C251FB	Bin	Ext	0	non-page	TMAC7	2:34	2:34	2:34	2:34	5.80x
4	87C251FB	Bin	Ext	1	non-page	TMAC7	3:39	3:39	3:39	3:39	4.08x
5	87C251FB	Src	Ext	0	non-page	TMAC8	2:21	2:21	2:21	2:21	6.33x
6	87C251FB	Src	Ext	1	non-page	TMAC8	3:17	3:18	3:17	3:17	4.53x

7	87C251FB	Bin	Ext	0	page	TMAC7	1:43	1:43	1:43	1:43	8.67x
8	87C251FB	Bin	Ext	1	page	TMAC7	2:38	2:39	2:38	2:38	5.65x
9	87C251FB	Src	Ext	0	page	TMAC8	1:36	1:36	1:36	1:36	9.30x
10	87C251FB	Src	Ext	1	page	TMAC8	2:25	2:25	2:25	2:25	6.16x
11	87C251FB	Bin	Int	0	non-page	TMAC7	1:33	1:33	1:33	1:33	9.60x
12	87C251FB	Src	Int	0	non-page	TMAC8	1:26	1:26	1:26	1:26	10.38x

Table 4.2





DISCUSSION

Table 4.1 & Graph 4.1

The results of the 15 tests are shown in graph 4.1. The test program were written in pure 51 instructions.

From the graph, for program running externally at external eprom, the 87C251SB is 2.23 times faster than the 87C51FB. However, with the hardware modification to enable page-mode capability of the 87C251SB microcontroller, the performance rose up to 3.63 times faster than the 87C51FB. The page mode actually speeds up the execution time of the 87C251SB microcontroller on external eprom processing by approximately 1.63 times. It is even faster than program assembled in source mode and running at internal eprom, which only provides a 3.57 times speed increase over the 87C51FB.

Table 4.2 & Graph 4.2

This graph shows a better performance of the 87C251SB after a modification done to the test program converting it to optimized 251 instructions. Program assembled in source mode performs better here. For 87C251SB running externally at external eprom without hardware modification, the execution time is 6.33 times faster than the 87C51FB. With hardware modification and running in page mode, the speed performance rose up to 9.3 times faster than the 87C51FB. From the graph, the performance of 87C251SB running at external eprom in page mode can actually compete with program running in internal eprom of the same microcontroller.

CONCLUSION

Code size and state analysis was not carried out in this experiment as they were analyzed in detail in the previous experiments. This experiment is mainly to show the performance of the 87C251SB in larger program with hardware modification to enable it to run in page mode.

From experiment 4, it is obvious that more benefits are to be obtained with hardware modification if a program were to run on external eprom. The performance of the 87C251SB microcontroller can be greatly increased and can even match the speed of internal memory of the same program.

FINAL CONCLUSION

All the tests mainly exercised the multiplication, addition, moving and branching instructions. It may not show the most optimized code, but it will give a brief idea on the actual performance of the microcontrollers.

Overall, the test programs in experiment 1, 2, 3 and 4 consist of small programs looping multiple times and they may not show the ideal performance of the 87C251SB microcontroller. In real life, larger programs with less loops will be used and the performance of the 87C251SB microcontroller will be better.

APPENDIX A: PROGRAM LISTINGS FOR TMAC11 AND TMAC12

Listing 1: TMAC11 application routine base on Data Transfer by using 51 instruction only

```
;*************************************************************
; FUNCTION OF ROUTINE:
; Perform moving data from internal code memory location 0100 to
; external data memory location 01:0030. Total of 64 bytes data will
; be moved. The destination is at data memory starting from location
;  01:0030 (XDATA)
; Rev 3.0 Jan. 5th 1995
; File: TMAC11.ASM
;*************************************************************

      NAME TMAC11

ACC   EQU   0E0H
DPL   EQU         82H
DPH   EQU         83H

DSEG    AT        01:0030H
      DST:        DS    40

CSEG    AT        0000H
      LJMP        MAIN

ORG 0100H
      SRC0:    DB
0H,1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH
      SRC10:  DB
10H,11H,12H,13H,14H,15H,16H,17H,18H,19H,1AH,1BH,1CH,1DH,1EH,1FH
```

```
      SRC20:  DB
20H,21H,22H,23H,24H,25H,26H,27H,28H,29H,2AH,2BH,2CH,2DH,2EH,2FH
      SRC30:  DB
30H,31H,32H,33H,34H,35H,36H,37H,38H,39H,3AH,3BH,3CH,3DH,3EH,3FH
      TOTAL:  DB  40H

ORG 0200H
MOVE1:
      PUSH        ACC
      MOV     DPL,       #LOW (TOTAL)
      MOV     DPH,      #HIGH (TOTAL)
      CLR      A
      MOVC   A,          @A+DPTR
      MOV     R5,        A
      MOV     R3,        #LOW (DST)
      MOV     R2,        #HIGH (DST)
      MOV     DPL,       #LOW (SRC0)
      MOV     DPH,       #HIGH (SRC0)

TABLE:
      CLR       A
      MOVC   A,          @A+DPTR
      MOV     R6,        DPH
      MOV     R7,        DPL
      MOV     DPL,       R3
      MOV     DPH,       R2
      MOVX    @DPTR,    A
      INC         DPTR
      MOV     R3,        DPL
      MOV     R2,        DPH
      MOV     DPL,       R7
      MOV     DPH,       R6
      INC         DPTR
      DJNZ    R5,        TABLE
      POP         ACC
      MOV     DPTR,      #0FF80H
      RET

MAIN:
      MOV     DPTR,      #0FF80H     ; display LED's while waiting
      MOV     A,  #07EH              ;

START:
      MOVX  @DPTR,  A
      LCALL       MOVE1
      MOV   R1,   #0FH
DELAY2:
      MOV   R0,   #0FFH
DELAY3:
      LCALL MOVE1
      DJNZ  R0,   DELAY3
      LCALL       MOVE1
      DJNZ  R1,   DELAY2
      DEC   A
      AJMP  START

      END
```

Listing 2: TMAC12 application routine base on Data Transfer by using 51 and 251 instruction

```
;*************************************************************
```

```
; FUNCTION OF ROUTINE:
; Perform moving data from internal code memory location 0100 to
;   external data memory location 01:0030. Total of 64 bytes will be
;   moved. The destination is at data memory starting from location
;   01:0030. The 16 bit registers being used.
; Rev 4.0 Jan. 5th 1995
; File: TMAC12.ASM
;*************************************************************
;
      NAME TMAC12

ACC         EQU   0E0H
P1    EQU     90H

DSEG    AT        01:0030H
      DST:        DSW   1CH

CSEG    AT        0000H
      LJMP    MAIN

      ORG     0100H
      SRC0:   DW  0001H,0203H,0405H,0607H,0809H,0A0BH,0C0DH,0E0FH
      SRC10:  DW        1011H,1213H,1415H,1617H,1819H,1A1BH,1C1DH,1E1FH
      SRC20:  DW        2021H,2223H,2425H,2627H,2829H,2A2BH,2C2DH,2E2FH
      SRC30:  DW        3031H,3233H,3435H,3637H,3839H,3A3BH,3C3DH,3E3FH
      TOTAL:  DB        20H

      ORG     0200H
MOVE1:
      PUSH    ACC
      MOV     A,  R0
      PUSH    ACC
      MOV     A,  R1
      PUSH    ACC
      MOV     WR0,      #TOTAL
      MOV     R5,       #20H
      MOV     WR8,      #SRC0
      MOV     WR12,     #DST

TABLE:  MOV    WR16,    @WR8
      INC         WR8,  #2
      MOV     @WR12, WR16
      INC         WR12, #2
      DJNZ    R5, TABLE
      POP         ACC
      MOV     R1,       A
      POP         ACC
      MOV   R0, A
      POP         ACC
      MOV     DPTR,     #0FF80H
      RET

MAIN:
      MOV     DPTR,     #0FF80H     ; display LED's while waiting
      MOV     A,   #07EH            ;

START:
      MOVX  @DPTR, A
      LCALL  MOVE1
      MOV   R1,   #0FH
DELAY2:
      MOV   R0,   #0FFH
DELAY3:
      LCALL       MOVE1
```

```
DJNZ  R0,   DELAY3
LCALL       MOVE1
DJNZ  R1,   DELAY2
DEC   A
AJMP  START

END
```