International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)

http://ijcjournal.org/

Design of an ASIP Processor for Mathematic Functions

Mona Moradi*

Young Researcher and Elite Club, Roudehen Branch, Islamic Azad University, Roudehen, Tehran, Postal code: 3973188981, Iran

Email: mo.moradi@riau.ac.ir

Abstract

This paper presents new architecture for some instructions of Matlab Mathematic toolbar related to matrix such as Sort, Find max, Sind min, Sind size, Isempty, Isrow, Iscolumn, Isvector, Isscalar, Isfinite, Ismatrix, Isarray, , Isequal, Islogical, Findlength, NDims, Nheight, and member of array operations such as addition, subtraction, multiplication based on Application Specific Instruction Set(ASIP). These instructions can be considered as a part of ASIP processor for mathematics functions of Matlab software. Designing process for mentioned instructions is explained comprehensively. The basic structure is developed in order to reduce the required clock cycles for the mentioned instructions. The complete instruction set for each function is described in Register Transform Language.

Keywords: Matlab mathematics functions; ASIP; RTL; CPU; Register and ALU Configuration.

1. Introduction

Application Specific Instruction Set Processors (ASIPs) are the alternative programmable platforms comparing with conventional Application Specific Integrated Circuits (ASICs) [1]. ASIP makes compromise between ASICs and (Digital Signal Processing) DSPs. ASICs are very high-speed, and they need large consuming area and long time to reach the market; however DSPs utilize less area and are slower than ASICs. Meanwhile, ASIPs benefit from both advantages of ASICs and DSPs; it is faster than DSPs and uses less area than ASICs, so it can make sense of balance between costs And speed [2]. One of the ASIP remarkable features comparing with general purpose processor is its programmability, since it has its own specific instruction sets to execute a particular task, faster and reduce the programmer's mistakes [2, 3]. These features empower the software developers with more flexibility, easier designing and debugging of processes. In addition, ASIPs divides the entire task into both hardware and software aspects as an alternative of one side being principal [2]. Respecting these facts it can be concluded, ASIPs have the advantage of both programmability and efficiency at the same time.

---

* Corresponding author.

Matrix mathematical functions of Matlab software play important role in arithmetic operations and are utilized in many applications, thus designing an ASIP which is specified for these operations can reduce the delay time and power significantly.

In this paper, a new ASIP-based processor for some mathematics functions of Matlab package is proposed. The given architecture is designed to support both general purpose and mentioned instructions. Two structures called RC1 (register configuration), and RC2 are presented for each instruction. RC2 is the main contribution in this paper which benefits from low hardware complexity and less clock cycles.

The rest of the paper is organized as follows: Section 2 consists of a brief summary of each instruction functionality. The proposed register configurations and architecture are presented in section 3. Finally, section 4 concludes the paper.

2. Instructions Functionality

Mathematical operations for arrays and matrixes in Matlab software are essential functions for computations; hence considering ASIP advantages, designing an ASIP-based processor for them can present great advances in reducing computing delay time in variety applications related to computation such as genetic algorithm, weather forecasting, defending simulations and so on. For the first step the following instructions have been implemented. The complete instruction set is listed in Appendix.

Sort	Is-Finite
Fmax	Is-Matrix
Fmin	IS-Array
Size	Is-equal
Is-Empty	Is-logical
Is-Row	Length
Is-Column	Ndim
Is-Vector	Nheight
Is-Scalar	Is-Finite
Members additions	Members subtraction
Members multiply	

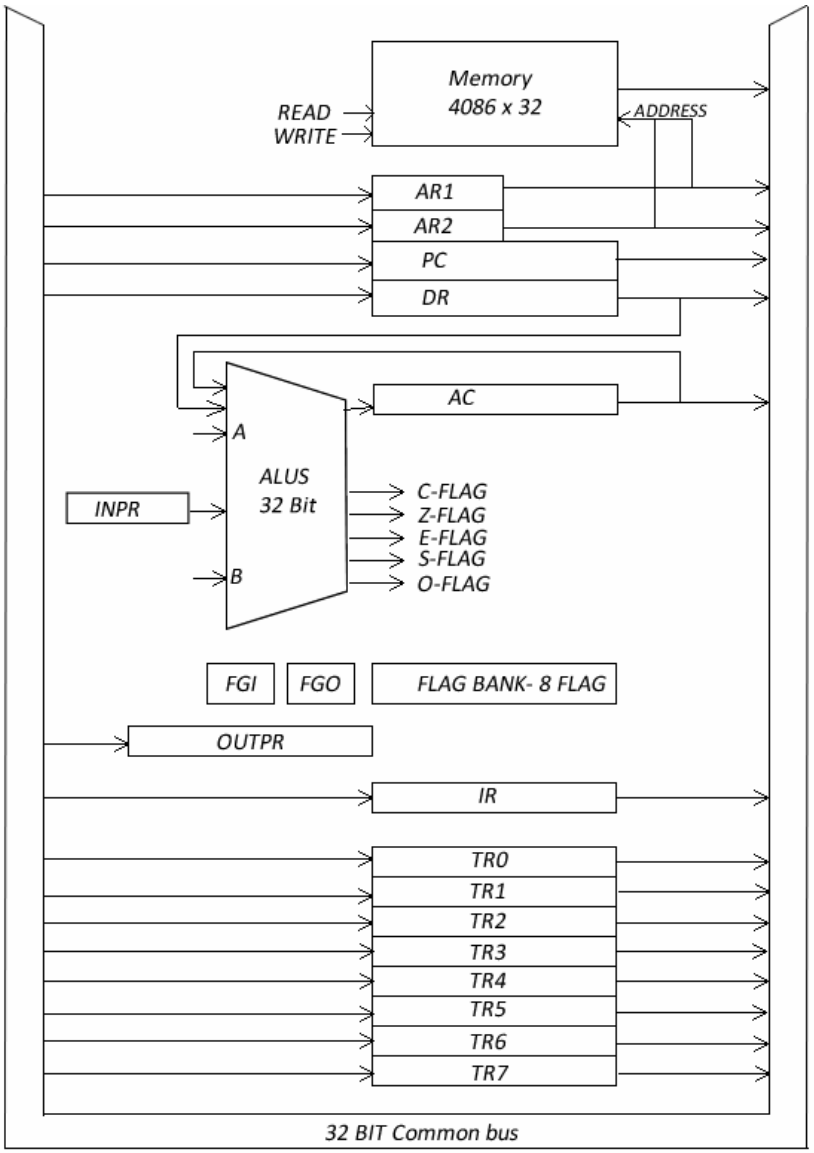

Figure 1: Register and ALSU configuration (RC2)

The selected architecture is based on what M. Mano has presented in [4, 5] which is called RC1, howeve it utilizes more (temporary registers) TR than conventional Mano architecture.

For implantation of RC2 the differences are the existence of more registers such as two (memory address register) MAR registers and one register bank which has 8 extra temporary registers ,and the existence of 7 bit sequence counter in Control Unit In order to generate 128 timing signals and also 7 extra flags(figure 1,2).

The comprehensive instruction set for each function is described in Register Transform Language (RTL) based on each register configuration, the required clock cycles is indicated in table.1.

As it is illustrated numbers of clock cycles have reduced due to registers configurations and register transfer optimizations in each clock cycle.

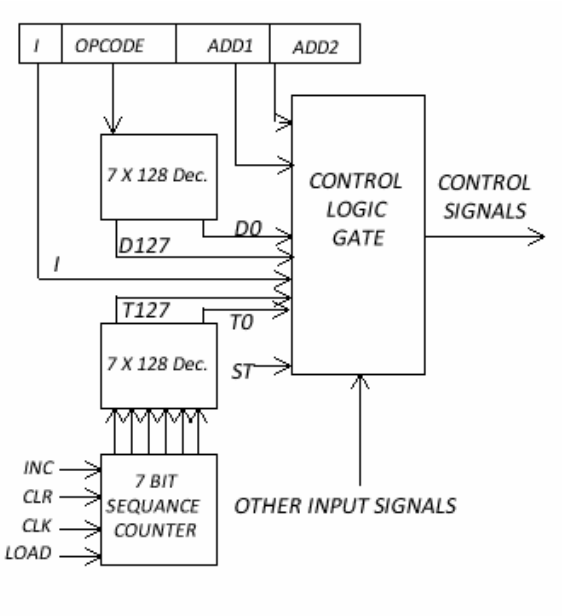

Figure 2: control unit configuration of rc2

Table 1: comparison result of RC1

Specific instructions	# clock cycle in RG1	# clock cycle in RG2
Sort		27
Fmax		29
Fmin		31
Size		30
Is-Empty		9
Is-Row		36
Is-Column		36
Is-Vector		40
Is-Scalar		39
Is-Finite		20
Is-Matrix		40
IS-Array		38
Is-equal		48
Is-logical		53
Length		47
Ndim		45
Nheight		40
Members additions		53

3. Conclusion

In this paper a new ASIP processor has been presented for some Mathematical operations in Matlab software. The main goal of the ASIP design has been to provide the required programmability, flexibility and speed for specific mentioned instructions. The instruction set consists of general purpose instructions together with eighteen specific instructions. Designing steps for all main components within the processor core have been explained. The first structure (RC1) has been modified by adding extra registers, flags, and CU modifications to boost the power of process. The execution of the each specific instruction, required clock cycles considering the

final structure illustrated in table1.

The presented architecture regarding to ASIP pros benefits from low hardware complexity and less clock cycles.

References

[1] K. Keutzer, S. Malik, and A.R. Newton, "From ASIC to ASIP: the next design discontinuity", In Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 84-90, 2002.

[2]. Reza Faghih Mirzaee, Mohammad Eshghi "Design of an ASIP IDEA Crypto Processor", Networked Embedded Systems for Enterprise Applications (NESEA), 2011 IEEE 2nd International Conference on

[3]. Yavar Safaei Mehrabani, Mohammad Eshghi "Desing of an ASIP processor for MD5 hash algorithm" 20th Telecommunications forum TELFOR 2012 Serbia, Belgrade, November 20-22, 2012.

[4]. M.M. Mano, Computer System Architecture, 3rd Edition, 1992.

Appendix (Instruction Set)

Di	Fi	Instruction	Name	Description	Ins. Reference	Opcode
D0	F0	RCV	Receive	ACL<—INPR	I/O	000000
D0	F1	SND	SEND	OUTR<—ACL	I/O	000001
D0	F2	SKI	Skip if FGI	FGI:PC<—PC+1	I/O	000010
D0	F3	SKO	Skip if FGO	FGO:PC<—PC+1	I/O	000011
D0	F4	ION	IEN ON	IEN<—1	I/O	000100
D0	F5	IOF	IEN Off	IEN<—0	I/O	000101
D0	F6	CLA	Clear Accumulator	ACL<—0	Register	000110
D0	F7	CAF	Clear Arithmetic Flags	S,Z,O,C<—0	Register	000111
D0	F8	CMA	Complement Accumulator	AC<—¬AC	Register	001000
D0	F9	CMC	Complement Carry Flag Register	C <—¬C	Register	001001
D0	F10	INC	Increment Accumulator	AC<—AC+1	Register	001010
D0	F11	DEC	De Increment Accumulator	AC<—AC-1	Register	001011

D0	F12	SHL	Shift Left Accumulator	AC<—SHL(ACL)	Register	001100
D0	F13	SHR	Shift Right Accumulator	AC<—SHR(ACL)	Register	001101
D0	F14	ROL	Rotate Left Accumulator	AC<—ROL(ACL)	Register	001110
D0	F15	ROR	Rotate Right Accumulator	AC<—ROR(ACL)	Register	001111
D0	F16	SNA	Skip if Negative Accumulator	S:PC-PC+1	Register	010000
D0	F17	SNA	Skip if Zero Accumulator	Z:PC<—PC+1	Register	010001
D0	F18	SEA	Skip if Even Accumulator	E:PC<—PC+1	Register	010010
D0	F19	SZC	Skip if Zero Carry flag	¬C:PC<—PC+1	Register	010011
D0	F20	CME	Complement E	E<—¬E	Register	001100
D0	F21	CLE	Clear E	E<—0	Register	001101
D0	F22	MOVDA	Move DR to AC	AC<—DR	Register	001110
D0	F22	HLT	Hult	SC<—Disable	Register	001111
D1	F0	QLD	Quick Load Accumulator	ACL<—Data	Quick Memory	000000
D1	F1	QCM	Quick Complement	Data<—¬Data	Quick Memory	000001
D1	F2	QAN	Quick And	Data<—Data^ACL	Memory	000010
D1	F3	QOR	Quick OR	Data<—Data ∨ ACL	Memory	000011
D1	F4	QNA	Quick NAND	Data<—¬(Data ^ ACL)	Memory	000100
D1	F5	QNO	Quick NOR	Data<— ¬(Data ∨ ACL)	Memory	000101
D1	F6	QXR	Quick XOR	Data<—Data ⊕ACL	Memory	000110
D1	F7	QXN	Quick XNOR	Data<— ¬(Data ⊕ ACL)	Memory	000111
D1	F8	QAD	Quick Addition	Data<—Data +ACL	Memory	001000
D1	F9	QSB	Quick Subtraction	Data<—Data -ACL	Memory	001001
D1	F10	QSB	Quick Multiplication	Data<—Data ×ACL	Memory	001010
D2 | D3	F0	COM	Complement	ACL<—¬M[AR]	Memory	000000
D2 | D3	F1	AND	ADD	ACL<—M[AR] ^ACL	Memory	000001

D2 | D3	F2	OR	OR	ACL<—M[AR] ∨ ACL	Memory	000010
D2 | D3	F3	NAND	NAND	ACL<—¬(M[AR] ^ACL)	Memory	000011
D2 | D3	F4	NOR	NOR	ACL<—¬(M[AR] ∨ACL)	Memory	000100
D2 | D3	F5	XOR	XOR	ACL<—M[AR] ⊕ACL	Memory	000101
D2 | D3	F6	XNOR	XNOR	ACL<—¬(M[AR] ⊕ACL)	Memory	000110
D2 | D3	F7	ADD	Addition	ACL<—M[AR] +ACL	Memory	000111
D2 | D3	F8	SUB	Subtraction	ACL<—M[AR] -ACL	Memory	001000
D2 | D3	F9	MUL	Multiplication	ACL<—M[AR] ×ACL	Memory	001001
D2 | D3	F10	LDA	Load Accumulator	ACL<—M[AR]	Memory	001010
D2 | D3	F11	STA	Store Accumulator	M[AR] <—ACL	Memory	001011
D2 | D3	F12	JMP	Jump	PC<—AR	Memory	001100
D2 | D3	F13	JSR	Jump & Save Return Address	M[AR] <— PC , PC<— AR	Memory	001101
D2 | D3	F14	DSZ	Decrement and Skip if Zero	ACL/M[AR] -1 , Z:PC<— PC+1	Memory	001111
D2 | D3	F15	SRT	Sort	Sorting	Specific / Memory	010000
D2 | D3	F16	FMAX	Fmax	Finding Maximum	Specific / Memory	010001
D2 | D3	F17	FMIN	Fmin	Finding Minimum	Specific / Memory	010010
D2 | D3	F18	SIZE	Size	Finding Size	Specific / Memory	010011
D2 | D3	F19	ISEMP	Is-Empty	Finding being empty	Specific / Memory	010100
D2 | D3	F20	ISSCAL	Is-Scalar	Finding being scalar	Specific / Memory	010101
D2 | D3	F21	ISFIN	Is-Finite	Finding being finite	Specific / Memory	010110
D2 | D3	F22	ISEQ	Is-equal	Finding all members are equal	Specific / Memory	010111
D2 | D3	F23	ISLOG	Is-logical	Finding all members are	Specific / Memory	011000

				logical		
D2 | D3	F24	LNG	Length	Finding length	Specific / Memory	011001
D2 | D3	F25	NDIM	Ndim	Finding Number of Dimension	Specific / Memory	011010
D2 | D3	F26	MEMADD	Members additions		Specific / Memory	011011
D2 | D3	F27	MEMSUB	Members subtraction		Specific / Memory	011100
D2 | D3	F28	MEMMUL	Members multiply		Specific / Memory	011101