Nigerian Journal of Engineering
Faculty of Engineering
Ahmadu Bello University
Samaru - Zaria, Nigeria

Vol. 26, No. 3, December 2019
ISSN: 0794 - 4756

DEVELOPMENT OF A BEHAVIORAL MODEL FOR THE ARM 32-BIT PROCESSOR

I. J. Umoh

Department of Computer Engineering, Ahmadu Bello University, Zaria, Nigeria
Email: ime.umoh@gmail.com

ABSTRACT
ARM microcontrollers are being applied in the design of portable devices. This paper presents the implementation of a single processor ARM 32-bit microcontroller. The model of the processor developed using Verilog hardware description Language was tested by compiling a factorial program written in high level C programming language. The recursive factorial program implemented on the processor showed that the model properly references the stack memory. Results showed that the behavioural model was a true representation of the actual processor as the programs were compiled using industry standard tools.
Keywords: Single processor, portable devices, factorial, ARM microcontroller.

INTRODUCTION
This paper aimed at designing the behavioural model of a single core 32-bit ARM 11 processor to be used as a microcontroller. ARM processors support either a 16-bit or 32-bit instruction set (NXP, 2009). The proposed model was built around the ARM 32-bit instruction set. Implementation of the processor core isdone in high level Verilog Hardware Description Language (HDL). Verilog HDL is industry standard software for designing processors (IEEE, 2006).

All ARM processors are Reduced Instruction Set Computers (RISC) (ARM7TDMI-S, 2001).Other RISC based architecture include SPARC (SPARC, 1992), PowerPC (Wetzel *et al.*, 2005), MIPS (MIPS, 2014) while x86 (Intel, 2016) is a popular Complex Instruction Set Computer (CISC). This allows for fast program implementation and reduced code lengths. Other features of the processor include a 32-bit architecture that supports 32-bit (word), 16-bit (half word) and 8-bit (byte) data types (NXP, 2009). It is programmable as either little endian or big endian data alignment in memory. ARM processors have been implemented in both the Princeton memory architecture (ARM7TDMI) (ARM7TDMI-S, 2001) and the Harvard architecture (ARM920T) (ARM920T, 2001). The ARM7TDMI featured a 3-stage pipeline and a single interface to memory. The ARM920T featured a 5-stage pipeline, memory management unit, caches, single-cycle 32x16 multiplier and Jazelle technology. The Jazelle technology enables fast Java code execution. The ARM11 series (ARM11, 2008) featured up-to 8-stage pipeline with branch prediction. From the aspect of hardware design, ARM processors offer smaller die size, few transistors, and low power consumption which makes it a preferred processor for mobile technologies. Hence, ARM processors are extensively utilized as microcontrollers in embedded systems.

The highlight of this paper is the design of an ARM11 based on the Harvard architecture as shown in Figure 1. The scope of the proposed design will be to develop the behavioural model of a single ARM11 processor. The design will implement interrupts in the processor. The functionality of the proposed processor will be demonstrated using a recursive factorial program written in C programming language, complied with industry standard compliers for ARM processors and the resulting hex code run on the design.

PROGRAMMER'S MODEL
The proposed single core will be able to execute most of the instructions supported by the ARM 32-bit instruction set. The execution of these instructions from the fetch to execute will complete in one clock cycle for most of the supported instructions. Thus, the design does not implement pipelining. Equally, demonstration of the processor was done using C language and Assembly as such the Jazelle technology was not implemented. For high code density, 32-bit ARM processor allow a mixture of both the 32-bit ARM instruction set and the 16-bit Thumb instruction set (ARM920T, 2001) (ARM11, 2008) (ARM7TDMI-S, 2001). The processor fetches 2 bytes in Thumb mode in place of 4 bytes in the ARM mode. These enable a 32-bit performance to a 16-bit memory system.

In this paper, the seven modes defined in ARM architecture are implemented. As shown in Table 1, only one mode, User Mode (USR), is a non-privileged mode. The other six modes, Fast Interrupt request mode (FIQ), normal interrupt request mode (IRQ), abort mode (ABT), supervisor mode (SVC), undefined mode (UND) and system mode (SYS) are privileged modes. The processor will be in one these modes at a given point in time.

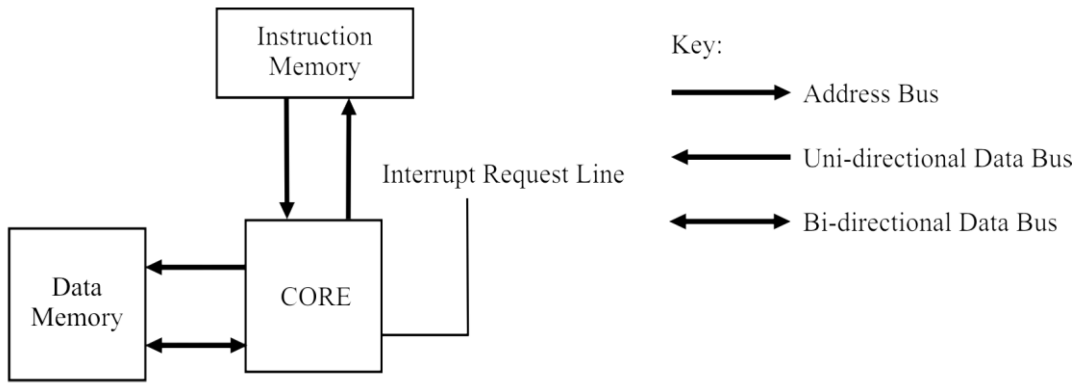

Figure 1: Diagram of a single core

Table 1: ARM modes and identification bits

Mode	Privileged	Mode bits, M [4:0]
USR	NO	10000
FIQ	YES	10001
IRQ	YES	10010
ABT	YES	10111
SVC	YES	10011
UND	YES	11011
SYS	YES	11111

(Sloss *et al.*, 2004)

The design implements a total of 31 general purpose registers. From Figure 2 there are 16 addressable registers in USR mode and a function specific register (Furber, 2000), Current Status Program Register (CSPR). All the flags generated by the Arithmetic Logic Unit are stored in the CSPR as shown in Figure 3. Also, the bits 0 to 4, M, indicated the mode the processor is currently operating. When M is set to 10000, the processor is in USR mode. In FIQ mode, M is set to 10001. When in FIQ mode, USR registers R0 to R7 and R15 are unbanked and can be changed in this mode. However, USR registers R8 to R14 are banked and the mode has alternative registers. FIQ mode has more banked registered than any other mode so as to reduce the latency in servicing an interrupt. Another mode for servicing interrupts is the IRQ mode with M set to 10010. In this mode only R13 and R14 are banked registers. Thus, the service routine will be required to push into stack other registers that are required. While running an Operating System, the processor is set to SVC mode (Berger, 2005) and M set to 10010. Here also, only R13 and R14 are banked registers. The other two modes UND and ABT modes have M set to 11011 and 10111 respectively. These respective modes are entered when the processor encounters an instruction it does not know how to execute and when there is an illegal access to memory (Berger, 2005). The SYS mode does not have any banked registers. This mode is the privileged equivalent of the USR mode. In this mode M is set to 11111.

Register Bits	User Mode	FIQ Mode	IRQ Mode	SVC Mode	UND Mode	ABT Mode
0000	R0					
0001	R1					
0010	R2					
0011	R3					
0100	R4					
0101	R5					
0110	R6					
0111	R7					
1000	R8	R8				
1001	R9	R9				
1010	R10	R10				
1011	R11	R11				
1100	R12	R12				
1101	R13	R13	R13	R13	R13	R13
1110	R14	R14	R14	R14	R14	R14
1111	R15					
	CPSR					
		SPSR	SPSR	SPSR	SPSR	SPSR

Figure 2: ARM modes and visible registers (Sloss *et al.*, 2004)

31	30	29	28	27	26 25	24	23 20	19 16	15 10	9	8	7	6	5	4 0
N	Z	C	V	Q	RES	J	RES	GE	RES	E	A	I	F	T	**M**

Figure 3: Status register

By transitioning from the USR mode to other privileged modes the CPSR can be stored in the Saved Program Status Register SPSR of the respective mode. In this design, the processor uses the following flags shown in Figure 2 for decision making; N, Z, V, C, Q, GE and E bits. N, Z, V and C represent the negative, zero, overflow and carry flags. A write to other bits will be ignored by the processor (Heath, 2003).

THE PROCESSOR DESIGN

The status registers enable the processor to operate in a deterministic and predictable way. A block diagram of the processor design can be seen in Figure 5. This design uses the Harvard Architecture where the instruction memory uses a separated data-bus from the data memory. In this design, almost all the instructions are fetched, decoded and executed in one clock cycle and pipelining is not implemented. This is done to model the behavior of the ARM processor. Where such a design attempted to be fabricated on chip, it will result a large chip area.

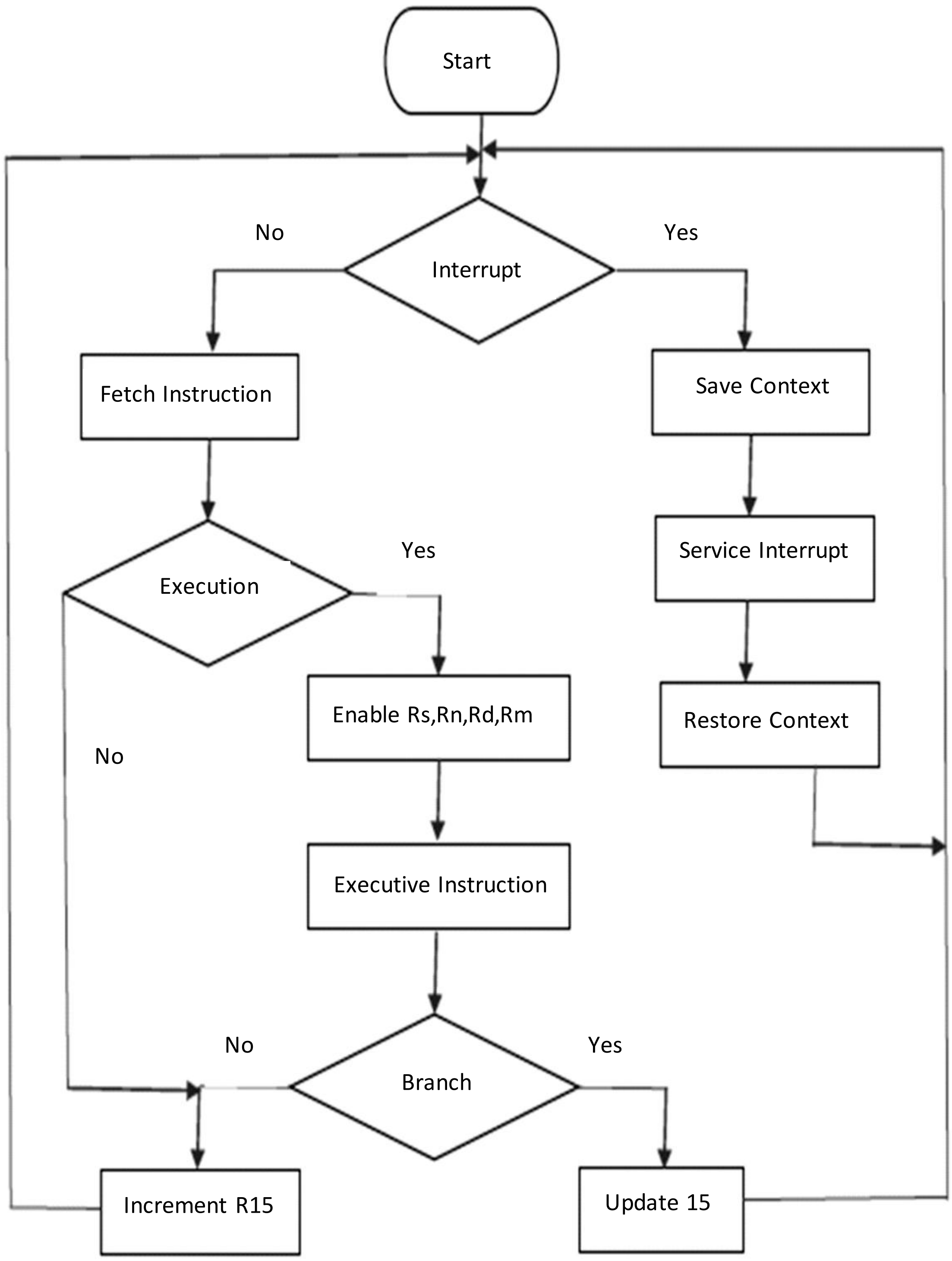

Figure 4: Flowchart of sequence of operation

A flowchart shown in Figure 4 illustrates the processor executes an instruction. Register R15 is the designs program counter, PC, and it points to the address of the instruction to be fetched from the instruction memory. In the fetch state, the processor first tests the presence of an interrupt signal. Where an interrupt is present, a context switch is made to either IRQ or FIQ mode. The processor then executes the service routine for the interrupt. In the absence of an interrupt, the processor tests the condition for execution. All the instructions are conditionally executed (Goodacre and Sloss, 2005). Where the instruction does not satisfy the condition for execution, the processor increments the program counter and fetches the next instruction. Where the instruction does satisfy the condition for execution the processor enters the decoding state.

In this state, the instruction is then decoded in the Register Read block. Respective registers represented by Rm, Rs, Rn and Rd, shown in Figure 5, are enabled as destination registers (Rn and Rd) or as source registers (Rs and Rm). All instructions update the register addressed by the Rd bits except multiplication which updates the register addressed by the Rn bit.

The final state the processor enters is the execute state. In this state the proposed processor design implements the following functions; swap memory, multiplication and multiplication and accumulate, MRS and MSR, Branch and Branch and Link, load and store, load and store multiple, arithmetic and logical processing.

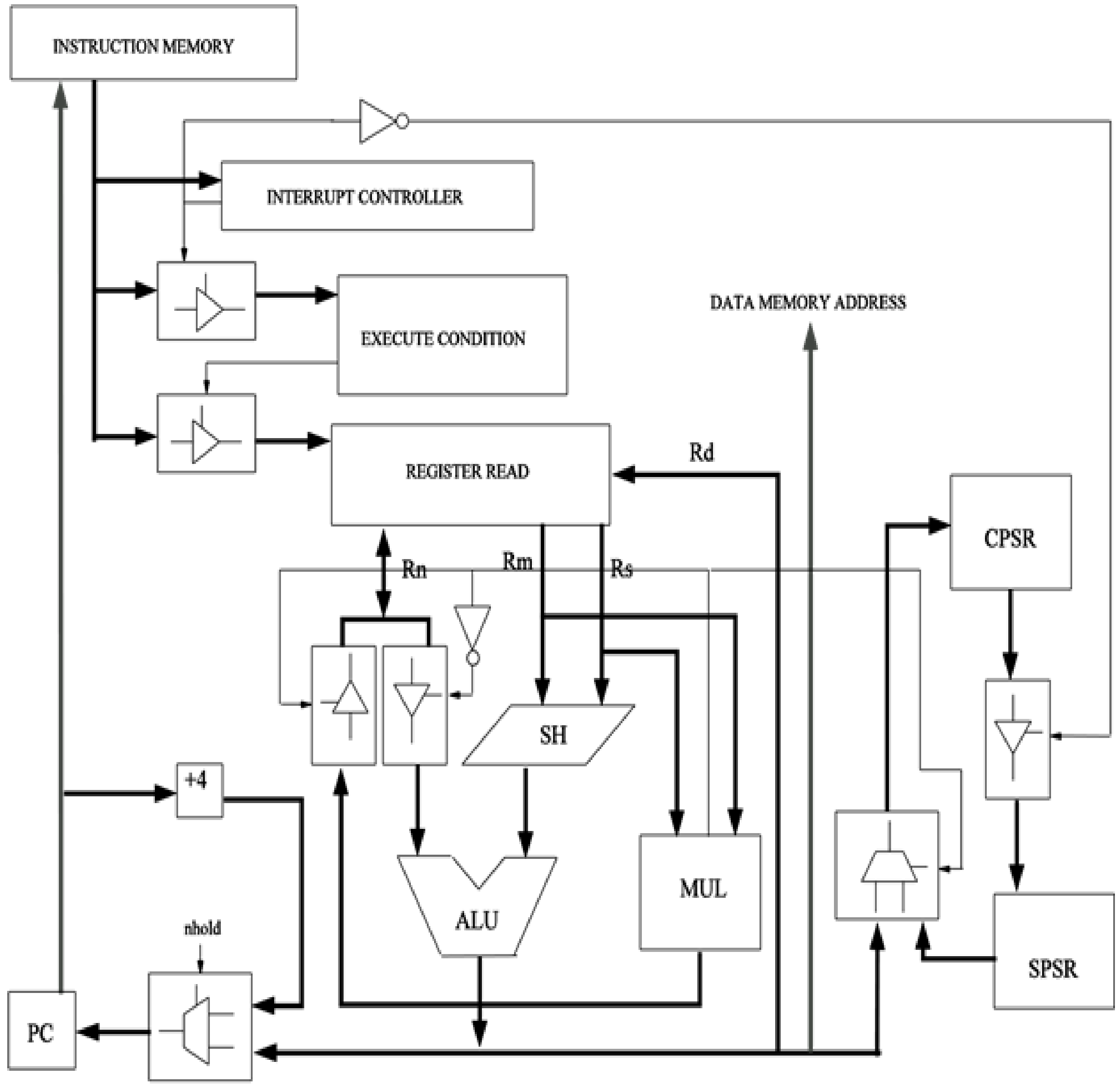

Figure 5: Data path of processor design

Key:
ALU: Arithmetic Logic Unit
MUL: multiplication unit
S_H: barrel shifter

Table 2: Instruction set format

	31 28	27 26	25	24	23	22	21	20	19 16	15 12	11 8	7 4	3 0
1	COND	0 0	I	OPCODE				S	Rn	Rd			Rm
2	COND	0 0	0	0	0	0	W	S	Rn	Rd	Rs	1 0 0 1	Rm
3	COND	0 1	I	P	U	B	W	L	Rn	Rd	ADDRESS MODE		
4	COND	1 0 1		L	SIGNED WORD OFFSET								

First, in swap memory the processor swaps data between the data memory and a register. Bits required to decode this instruction are; 27 to 23, 21 to 20 and 7 to 4. The entire process takes two clock cycles to implement.

Second, the multiplication and multiplication and accumulate instruction is required to perform all multiplications between two operands. In row 2 of Table 2, the bits decoded for this instruction are 27 to 22 and 7 to 4. A block in the processor handles the multiplication of 2 operands. The result is written to the destination register.

Third, when a context is required to be stored or retrieved, the MSR and MRS allow the status register to be copied to an addressed register and vice versa. The bits decoded to carry out this function include; 27 to 23, 21 to 20 and 7 to 4. Bit 22 being high or low determines if the SPSR or the CPSR is to be updated.

Fourth, the branch and branch and link instructions handle the flow control of the processor. These instructions are decoded from bits 27 to 24 as shown in row 4 of Table 2. The branch and link differs from branch by storing the return address in the link register, R14. Setting bit 24 in the instruction register high implies a branch and link operation, and a branch instruction otherwise. Branch statements are PC relative therefore; the processor adds the offset value to PC and stores the result in PC. These instructions prevent auto increment of PC.

Fifth, to save data in the memory from the register uses a store instruction while to store data in the register from the memory uses a load instruction. Both instructions can be decoded from bits; 27 to 26 as shown in row 3 of Table 2. The store instruction takes one clock cycle to execute while the load instruction takes two clock cycles to complete. The processor supports updating registers a byte, half-word or word.

Sixth, a variation of the load and store instruction is the load and store multiple. This is a single instruction that can load or store multiple registers from or to the memory. The instruction is decoded from the instruction register using the bits 27 to 25.The load multiple takes additional clock cycles, compared to just the load and store, to set the memory address latch before loading the registers.

Final, the proposed processor can execute arithmetic and logic operations between registers or between a register and an immediate value embedded in the instruction. Data processing instructions are decoded by the bits 27 to 26, 25, and 24 to 21 as shown in row 1 of Table 2.

All instructions except the branch statements increments the program counters by 4. If an instruction is not executed in any of the block mentioned above, the instruction will be treated as an undefined instruction. Also, for cases where the condition of executing an instruction is not satisfied. The program counter will be incremented to point to the next instruction.

TESTING AND EVALUATION

To demonstrate the behavioral model a factorial program was written in C language as shown in Algorithm 1.0. The recursive property of the factorial program forces the processor to utilize a stack. This program tests how the processor pushes data into the stack and removes the data. The C language code is compiled with a standard C compiler with the target processor set as ‘arm4’. The resulting object file is shown in Algorithm 2.0, the extracted Hex file was ran on the processor.

Algorithm 1.0: A recursive Factorial subroutine

```
int factorial (int a){if (a <= 1) return 1;
else return a * factorial (a - 1);}
```

Algorithm 2.0: Factorial generated Assembly code

```
factorial.o: file format elf32-littlearm
Disassembly of section .text:
00008000 <factorial>:
8000: e92d4800 push {fp, lr}
8004: e28db004 add fp, sp, #4 ; 0x4
8008: e24dd004 sub sp, sp, #4 ; 0x4
800c: e50b0008 str r0, [fp, #-8]
8010: e51b3008 ldr r3, [fp, #-8]
8014: e3530001 cmp r3, #1 ; 0x1
8018: ca000001 bgt 8024 <factorial+0x24>
801c: e3a03001 mov r3, #1 ; 0x1
8020: ea000006 b 8040 <factorial+0x40>
8024: e51b3008 ldr r3, [fp, #-8]
8028: e2433001 sub r3, r3, #1 ; 0x1
802c: e1a00003 mov r0, r3
8030: ebfffff2 bl 8000 <factorial>
8034: e1a03000 mov r3, r0
8038: e51b2008 ldr r2, [fp, #-8]
803c: e0030392 mul r3, r2, r3
8040: e1a00003 mov r0, r3
8044: e24bd004 sub sp, fp, #4 ; 0x4
8048: e8bd8800 pop {fp, pc}
```

When the processor is run, the respective registers are first initialized. In this processor, the stack pointer, R13, is set to address locations, h'200000ac, h'20000044, h'20000078, h'20000270, h'200000e0 and h'200003f8 for modes SVC, ABT, UND, IRQ, FIQ and USR respectively. Figure 6 shows the order in which the initialization is done. After the initialization is done, then the factorial subroutine is run.

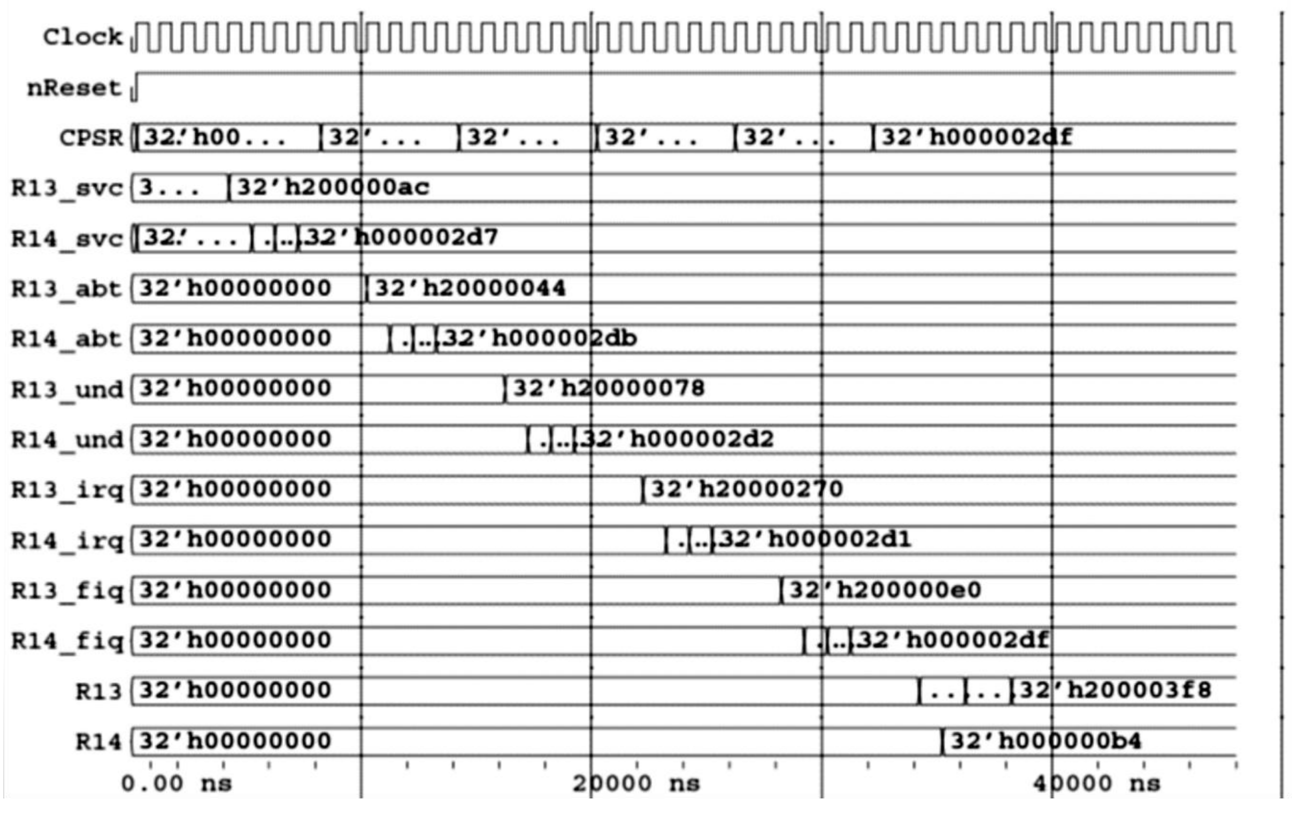

Figure 6: Initialization of special registers for the various modes of operation

To execute the factorial program the compliers uses 5 addressable registers, R0, R2, R3, R11 and R13. R11 is used as a frame pointer, FP. Since, a recursive function calls itself; the return address is stored in the stack, R13

Table 3: Data in registers in execution time

time (ns)	instruction	R2	R3	R11 (fp)	R13 (sp)	R14
167504	0x00000814	0xe000e000	0x1	0x200003d0	0x200003c8	0x00000818
189499	0xe1a00003	0x2	0x2	0x200003d0	0x200003c8	0x00000818
198597	0xe1a00003	0x3	0x6	0x200003dc	0x200003d4	0x00000818
207503	0xe1a00003	0x4	0x18	0x200003e8	0x200003e0	0x00000818
216488	0xe1a00003	0x5	0x78	0x200003f4	0x200003ec	0x00000818
220488	0xe8bd8800	0x5	0x78	0x200003fc	0x200003f8	0x00000818

Table 3 shows the working of the processor as it retrieves already stored addresses in the stack. During the calculation of 5!, R3 holds the result of the multiplications. At 189499 ns, R3 holds the result of multiplication of 2 and 1 which are the initial data in R2 and R3 respectively. From Algorithm 2.0 the processor is executing instruction at line 803c and the program counter is current pointing to the next instruction. By 198597 ns, 207503 ns and 216488 ns, R3 held the values, 6, 24 and 120 respectively. By 220488 ns, after correctly calculating 5! As 120, the stack pointer, R13, has a value of 0x200003f8 which is the same value the stack pointer was initialized to as shown in Figure 6. Thus, the processor can correctly manage the stack memory.

CONCLUSIONS

This paper presents a design of behavioural model of a single processor ARM 32-bit microcontroller. In the design, most of the ARM instructions are implemented. As the microcontroller is aimed at being programmed using C Language. Jazelle an ARM technology for Java execution is not implemented. All data processing instructions are designed to be executed in one clock cycle while instructions that require memory access may require additional clock cycles per instruction. The proposed behavioural model is tested using a high level recursive factorial program written in C Programming Language. The factorial program was compiled and the generated HEX file was run on the processor. The processor not only correctly calculated the given factorial it also properly utilized the stack memory as shown in Table 3.

REFERENCES

ARM7TDMI-S, (2001). “ARM7TDMI-S Technical Reference Manual”, ARM, ch. 1, pp. 1-25, url. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0234b/DDI0234.pdf. Last accessed 16/01/2014.

ARM920T, (2001). “ARM920T Technical Reference Manual”, ARM, ch. 1, pp. 1-4, url. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0151c/ARM920T_TRM1_S.pdf Last accessed 16/01/2014.

ARM11 (2008). ARM11 MPCore Processor Technical Reference Manual”, ARM, ch. 1, pp. 1-4, url. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0360e/DDI0360E_arm11_mpcore_r1p0_trm.pdf. Last accessed 16/01/2014.

Berger, A. (2005).“Hardware and Computer Organisation: The Software Perspective, Elsevier, ch. 11.

Furber, S. (2000). “ARM System-on-chip Architecture”, Addison Wesley ch.2 pp 35 – 46.

Goodacre, J. and Sloss, A. N. (2005). "Parallelism and the ARM instruction set architecture," in *Computer*, vol. 38, no. 7, pp. 42-50, doi: 10.1109/MC.2005.239.

Heath, S. (2003). “Embedded systems design” Newness, ch1, ch6 and ch 7.

IEEE (2006). “IEEE Standard for Verilog Hardware Description Language”, in *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, vol., no., pp.1-590, 7 April 2006 doi: 10.1109/IEEESTD.2006.99495
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1620780&isnumber=33945.

Intel (2016). "Intel 64 and IA-32 Architectures Software Developer's Manual" Santa Clara, California: Intel Corporation.

MIPS (2014). "MIPS Architecture for Programmers Volume I-A: Introduction to the MIPS32 Architecture" Campbell, Califonia: MIPS Corporation

NXP (2009). "Get Better Code Density than 8/16 bit MCU's, NXP LCP1100 Cortex M0", NXP, pp 1-66, https://www.nxp.com/wcm_documents/techzones/microcontrollers-techzone/Presentations/cortex.m0.code.density.pdf last accessed 16/01/2014.

Sloss, A. N. Dominic, S. and Chris, W. (2004). "ARM System developers guide designing and optimizing system software", Morgan Kaufmann, ch. 1-2, pp. 1-43.

SPARC (1992). "The SPARC Architecture Manual" Califonia: SPARC International Inc

Wetzel, J. Silha, E. May, C. Frey, B. Furukawa, J. and Frazier, G (2005). "PowerPC User Instruction Set Architecture" Austin, Texas: IBM Corporation.