SAVAGE16 – 16-BIT RISC ARCHITECTURE GENERAL PURPOSE MICROPROCESSOR

Andrei-Sorin Gheorghe, Corneliu Burileanu

University "Politehnica" of Bucharest
andrei.gheorghe@onsemi.com, **cburileanu@messnet.pub.ro**

Abstract
*This paper describes the architecture and the internal structure of "**Savage16**", a fully functional general purpose reduced instruction set microprocessor, with a modified Harvard, five stage pipeline architecture. The memory organization and key architecture elements are being described, as well as the hardware block diagram and the internal structure. A summary of the instruction set is presented, along with a brief description of the addressing modes.*
Keywords: RISC, pipeline, interrupts, memory organization, instruction set.

1. INTRODUCTION

The Savage16 microprocessor is a 16-bit general purpose RISC machine [1], based on the modified Harvard architecture, allowing constants to be stored in the program memory.

The internal structure is implemented as a five stage instruction execution pipeline providing better performances than classical sequential flow microarchitecture [3].

2. ARCHITECTURE

2.1. Memory Organization

The Savage16 microprocessor requires separate and independent memories for storage of the program code and the data.

Each memory is addressed on dedicated 16bit buses, accessing a maximum of 2 x 2^{16} memory locations. The program memory word has a width of 32bits, the length of a complete instruction. The data memory word has a width of 16bits, the same as of the internal data buses.

Without pagination and virtual addressing, the total memory size that can be accessed by the Savage16 microprocessor is 256kB of program code and 128kB of data. If needed, pagination can be implemented using external multiplexer and some output port pins used as page selectors.

The structure of the program memory is shown in figure 1.

2.2. Internal Register Set

The Savage16 microprocessor has 16 general purpose registers, named R_0 – R_{15}, each of them 16-bit wide. Any of the internal registers can be used as either data source or destination for any instruction.

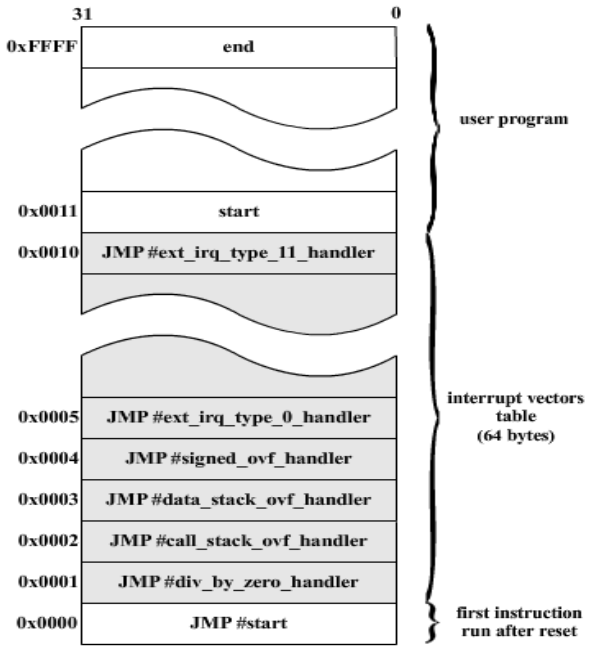

Fig. 1. Program Memory Structure

2.3. Flags

There are four status flags and four control flags in the flag register, as shown in figure 2.

The status flags are used to evaluate the last result of an arithmetic or a logic operation, indicating a null result (ZF), a carry or borrow bit (CF), a negative result (NF) or an overflow condition for signed operations (OF).

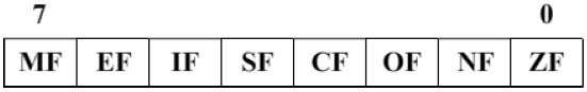

Fig. 2. The Flag Register

There is one control flag (SF) that switches the ALU between two modes of operation: unsigned integer and signed integer. When the SF flag is set, each ALU operation will interpret the two operands as signed integers and the operation will be done in two's complement format. The negative and overflow flags are set only in signed mode operations.

2.4. Interrupt System

A flexible and user configurable interrupt system is implemented in the Savage16 microprocessor.

There are 16 different types of interrupt requests (IRQs), each one having a designated interrupt handler with the starting address defined in the interrupt vector table. The first four interrupt requests are internally generated in case of a fault condition: division by zero, call stack overflow, data stack overflow and signed overflow. The remaining 12 interrupt requests are received on external dedicated pins.

The interrupt system can be configured using three dedicated control flags: IF – global interrupt enable flag, MF – masking enable flag and EF – signed overflow IRQ enable.

When the masking flag is set, each interrupt request can be disabled individually by means of the R_{15} register, which acts as a 16bit mask for the interrupt requests. When the masking flag is cleared, R_{15} returns to its general purpose state. The masking bit for each interrupt request is detailed in table 1:

Table 1. Savage16 Interrupt Requests

Type	Address	Generated by	Mask
0	0x0001	Division by zero	R_{15}[0]
1	0x0002	Call stack overflow	R_{15}[1]
2	0x0003	Data stack overflow	R_{15}[2]
3	0x0004	Signed overflow	R_{15}[3]
4	0x0005	IRQ_EXT$_0$ pin	R_{15}[4]
…	…	…	…
15	0x0010	IRQ_EXT$_{11}$ pin	R_{15}[15]

Type 0 request has the highest execution priority and type 15 has the lowest.

2.5. Hardware Stacks

Two hardware stacks have been implemented in the Savage16 architecture. This allows for user transparent stack operation and better memory organization and control.

The call stack is a 2kB fast SRAM memory core organized as 1024 locations x 16 bits and is used to store the value of the program counter before the execution of a subroutine or interrupt handler. A maximum of 1024 recursive calls are allowed, as a 1025th call will overflow the call stack and generate an overflow interrupt.

The data stack is used to store the internal registers, transfer arguments to subroutines or save the flag register. The data stack has been implemented as four times the size of the call stack to allow more than one register saves per each recursive call.

2.6. Input / Output Ports

There are four 16bit input ports named PINA through PIND and another four 16bit output ports named PORTA through PORTD.

Input data is read using the IN instruction and data is send to the output ports using the OUT instruction. No other port operations are possible.

3. INSTRUCTION SET

The Savage16 microprocessor is a three operand machine, each instruction requiring two source operands and one destination for the result. The source operands can only be found in the register file or in the program memory as a constant. The result will always be stored in the register file. There is no designated accumulator.

Most of the instructions are executed in one clock cycle, but there are some exceptions to this rule, as shown in table 2:

Table 2. Clocks per Instruction

Instruction Type	CPI
Arithmetic and Logic	1
Data Transfer	1
CPU Control	1
Branch	2
Call, Ret	2
Multiplication	11
Division	19

3.1. Instruction Summary

There are a total of 61 instructions in the Savage16 instruction set. These instructions can be split into five major categories:

a) CPU Control – instructions like NOP, STOP or SET do not generate a numeric result but alter the microprocessor's state. The SET and CLR instructions allow setting and clearing of any status or control flag. Multiple flags can be modified in the same clock cycle.
   Ex: `SET IF, MF, EF`
b) Arithmetic and Logic – instructions like ADD, NEG or XOR generate a numeric result as a function of two source operands, in unsigned integer or two's complement mode, depending of the state of the SF flag.
   Ex: `MOV R0, R1`
c) Data Transfer – instructions like MOV, LOAD or PUSH copy the content of an

internal register to another register, a memory location, a port or the data stack, or load the data from these sources to the register file. The OUT instruction is a very versatile one, allowing logic operations with the current value of the output port.

Ex: `OUT PORTA, XOR 0x0101`

d) Branch and Subroutine – instructions like JMP, CALL or RET, as well as the hardware implicit call of an interrupt handler alter the value of the program counter and access the call stack.

Ex: `CALL _delay`

e) Multiplication and Division – these are instructions based on finite automata and require more clock cycles for execution. These instructions generate two results per execution. The multiplication will output a 32bit result, with the last significant byte stored in the destination register and the most significant byte stored in the destination index increment register. The division will output a quotient and a remainder, with the quotient being stored in the destination register and the remainder being stored in the destination index increment register.

Ex: `MULT R1, R1, R4`

3.2. Addressing Modes

There are four addressing modes used in the Savage16 instruction set:

a) Immediate addressing – the data is a constant stored in the program memory.

Ex: `MOVI R0, 0x1234`

b) Register addressing – the data is stored in one of the internal registers.

Ex: `MOV R0, R1`

c) In memory immediate addressing – the data is stored in the external data memory and is accessed using a direct read or write address stored in the program memory.

Ex: `LOAD R0, 0x00a4`

d) In memory through register addressing – the data is stored in the external data memory and is accessed using an address stored in one of the internal registers.

Ex: `LOAD R0, (R1)`

All data transfers with the external memory are performed using the LOAD and STORE instructions. This is a fundamental characteristic of Savage16's RISC architecture [1].

4. INTERNAL STRUCTURE

The block diagram of the Savage16 microprocessor is represented in figure 3.

4.1. 1st Pipeline Stage – FETCH

The 1st pipeline stage deals with instruction fetch and interrupt request control. There are two major design blocks in this stage:

e) Instruction Fetch – this block includes the program counter and the circuitry for incrementing and loading the counter. Also, if an interrupt request is being served, a virtual CALL instruction is inserted into the pipeline, to start the execution of the corresponding interrupt handler.

f) IRQ Control – the interrupt request signals are sampled and filtered. A request buffer stores the requests and the execution of the interrupt handlers is done starting with the highest priority one and advancing towards the lowest priority one. All requests are served as long as the signaling pulse width is longer than the master clock period and no more than one request of the same type occurs during the execution of a higher priority interrupt handler.

4.2. 2nd Pipeline Stage – DECODE

The 2nd pipeline stage deals with instruction decoding, branch instructions control, and jump prediction. Also, the call stack is part of this pipeline stage. It includes two major blocks:

a) Decoder – a fully combinational block which generates all the control signals in the microprocessor based on the instruction code and arguments.

b) Fetch Control – this is the block that implements the 2bit jump predictor [2] and the call stack. It generates the program counter control signals and it decides when an interrupt request is served.

4.3. 3rd Pipeline Stage – READ

In the 3rd pipeline stage the two operands are read from the register file. There are three major design blocks in this stage: the Register File, the Flag Register and the Hazard Control.

The hazard prevention circuit [2] detects dependencies between consecutive instructions and activates the data forwarding paths which bypass the register file, speed up execution time and avoid the need for the programmer to assure enough NOP instructions between dependent instructions.

4.4. 4th Pipeline Stage – EXECUTE

In the 4th pipeline stage the result of the operation is computed. This is where all data sources like input ports, data stack or data memory are multiplexed.

Also, it is here that the actual branch instruction condition is evaluated and the correct decision is compared to that of the jump predictor, and in case of a mismatch the pipeline has to be flushed and CPU cycles are lost.

This pipeline stage includes large design blocks as the arithmetic and logic unit, the data stack, the multiplication and division circuits and the SRAM controller.

4.5. 5th Pipeline Stage – WRITE-BACK

The 5th pipeline stage implements the output ports and the circuitry that allows logic operations with their current value.

Also, the write-enable control signals for the register file, the flag register and the data memory are generated in this final pipeline stage.

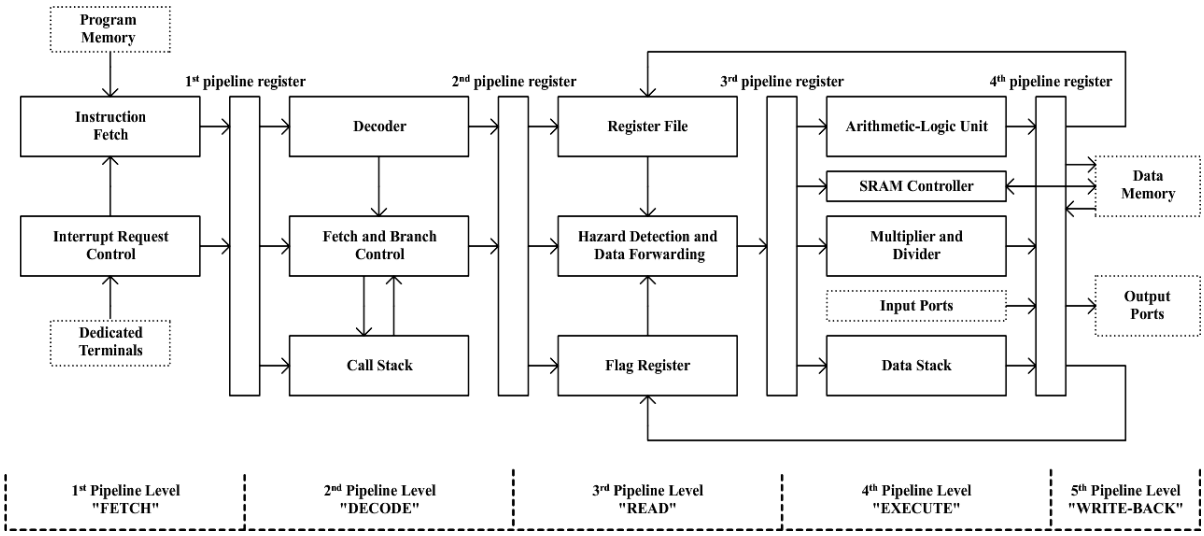

Fig. 3. Savage16 Microprocessor Block Diagram

5. ASSEMBLER TOOL

The Savage16 microprocessor has a dedicated assembler tool, which converts a user assembly text file into specific Savage16 machine code.

The assembler tool identifies branch labels and naming aliases and converts them into their numeric correspondents, allows for comments within the user assembly code, checks for illegal characters or wrong register use and chooses between the various types of the same instruction depending on the addressing mode.

5.1. Programming

After the machine code has been generated, an UART serial transmitter sends the data to an auxiliary block of the microprocessor. This block multiplexes the address and data busses that connect the program memory, and store the serially received data in the program memory.

When the programming is complete, it reconnects the microprocessor with the program memory and generates a hardware reset pulse.

6. CONCLUSIONS

The architecture described in this paper has been described in Verilog code and synthesized in a medium-performance FPGA. The FPGA had an internal block RAM which was used as memory cores for program and data storage.

The Savage16 microprocessor has been used in some university projects and is fully functional. The projects were written in assembly language and programmed into the microprocessor using the previously described assembler and programming tool.

REFERENCES

[1] C. Burileanu, *Microprocessor Architecture,* Denix, Bucharest, 1994.

[2] Z. Hascsi, M. Stoian, *Processor Architecture,* Fair Partners, Bucharest, 2003.

[3] D. A. Patterson, J. L. Hennessy – *Computer Organization and Design – The Hardware / Software Interface,* Morgan Kaufmann, San Francisco, 2004.