Design and Implementation of AHD–2494, a 24–bit RISC Processor on a VLSI Chip

Hossam Fahmy
Dept. of Communications and Electronics,
Faculty of Engineering,
Cairo University, Cairo, EGYPT.

Abstract

This paper presents the work done by the three students: Ahmed El-Wakeel, Hossam Fahmy and Dalia El-dib during their graduation project.

A 24 bit RISC processor following the Von Neumann architecture and having two modes of operation (operating system mode/user mode) was designed. It has 16 general purpose registers and executes one instruction per clock cycle. The instruction set contains 16 instructions all of fixed 24 bit length. It follows a load/store architecture. Simulation results show that a clock speed of 10 MHz driving a 50pf load off-chip capacitance can be used.

The implementation was done on a semi-custom process, the sea-of-gates fishbone image which is a 1.6 micron CMOS technology with two metal layers. The layout and layer interconnects were done manually in the basic cells. The full design consists of about 120 000 transistors.

Introduction

The RISC versus CISC debate during the past decade resulted in a lot of improvements to the microprocessor industry and finally RISC dominated the majority of the designs [1, 2]. Another factor that helped also in this improvement was the introduction of more powerful and easy to use CAD tools.

It was thus thought that undergraduate students should begin designing and implementing such architectures using available CAD's. The objective was to produce a RISC processor with comparable characteristics to those already built by researchers in universities in the past [2].

The design is done using the OCEAN CAD tools available from Delft university in the Netherlands where the final chip is to be fabricated. This affected the choice of architecture since the packaging allows 32 pins only. Therefore a reduced architecture was implemented, while a more powerful one was originally thought of. It may be implemented in the future by a group having less limitations.

Proposed Architecture

This architecture was based on the academic course we took on computer architectures and on our personal readings [1, 2, 3, 4].

- All instructions are 24 bits long.
- Four types of instructions are present and are differentiated according to the first 2 bits of the instruction word:

  00: integer ALU instructions

  01: floating point (only in full architecture)

  10: control flow instructions

  11: privileged system instructions
- The data buses are 24 bits wide.
- The addresses are 24 bits long giving up to 16 Megawords of memory (16 Megawords = 48 MBytes).
- The register file has 2 read ports and 1 write port and is composed of:
  – 8 registers for operating system usage.

– 8 integer registers for user programs.

Of these registers, register number zero is a special one. It is hardwired to the value 0 and any output of the ALU having R_0 as destination will be discarded [4]. All the registers are one word wide.

- Memory reference instructions addressing mode uses “base register + signed offset”. The base register may be the program counter (PC) or R_0 or any other register which enables this scheme to be very versatile and allows it to perform direct, relative and indexed modes. The signed offset is 11 bits long.
- The condition code flags may be conditionally affected according to a bit in the instruction word. The user accessible flags are: carry, zero, negative, overflow. Other flags are special to the operating system and used only in the system mode. This will be explained in details later.
- Three-stage pipeline is used with no interlocks nor data forwarding done. Such dependencies must be taken care of by the compiler.
- Instruction formats used:

Bits	**ALU operation**	**Control flow**
23–22	00	1x
21–19	opcode	opcode
18–15	R_{dest}	$R_{s/d}$ or cond.jmp
14–11	R_{op1}	R_{base}
10–7	R_{op2}	|
6	c	Signed offset
5–0	shift	|

where

R_{dest}:
destination register where result is stored,

R_{op1};
the first operand register,

R_{op2};
the second operand register,

c:
a bit indicating to set the flags or no,

shift:
amount of shift required on the result,

$R_{s/d}$:
source register for Store and Out instructions and the destination register for Load and In instructions,

cond.:
same field, used to indicate the condition for conditional instructions (Jump,Call,..)

R_{base}:
register holding the base address

signed offset:
offset added to the base address.

- The output of the ALU is connected to a barrel shifter. So, ALU operations are:
  R_{dest}= rotation [(R_{op1}) operation (R_{op2})] .
- If the c bit is set, the flags will be affected according to the result of the instruction, otherwise the flags will remain unaffected.
- For the shifting a bit in the shift field specifies whether it is to left or right, another bit is used to indicate if it is logical shift or a rotate and the remaining 4 bits specify the amount of the shift.

The choice of the instructions was influenced by the following factors:

- They must all be directly implemented by the hardware without iterations in order to facilitate the control unit and the pipeline implementation.
- Simple formats must be used to ease the decoding.
- They should not have any side effects on registers other than the destination register to minimize the data dependency [4].
- Simple and versatile instructions should be used in order to get the maximum possibilities out of the defined instruction set.
- Support to the ideas present in the operating system world must be provided [1]. For example, privileged system instructions, security for system registers, user mode/system mode, etc.

Proposed instruction set

The instruction set is presented in table (1). The integer operations are from number 1 to 7 where A is considered to be R_{op1} and B is R_{op2} (any 2 of the 16 registers.) The control flow operation are from 8 to 13. And the system instructions are from 14 to 16.

These instructions are quite versatile when used with R_0. Examples are:

Negate: R_0–B,
Transfer: R_0+B,
Increment: R_0+B+1,
Decrement: A–R_0–1,
Clear: R_{dest}=R_0+R_0,
Compare: R_0=A–B and set flags,
Complement: R_0–B–1,
NOP: R_0=R_0+R_0

Table (1):

function	mnemonic	operation
1. addition	ADD	A+B
2. ADD+1	ADDP	A+B+1
3. SUB–1	SUBM	A–B–1
4. subtraction	SUB	A–B
5. logic and	AND	A and B Bit by bit and
6. logic or	OR	A or B Bit by bit or
7. logic xor	XOR	A xor B Bit by bit XOR
8. loading	LOAD	R_{dest}= Memory[R_{base}+signed offset]
9. storing	STORE	Memory[R_{base}+signed offset] = R_{source}
10. Cond.jump	JUMP	upon condition: PC=R_{base}+ signed offset
11. Cond.call	CALL	upon condition: save PC; PC=R_{base}+ signed offset
12. Cond.re-turn	RET	upon condition: retrieve PC
13. system call SYS	SYS	upon condition: switches to system mode in the program status word and jumps to system code
14. I/O input	IN	R_{dest}= I/O[R_{base}+ signed offset]
15. I/O output	OUT	I/O[R_{base}+ signed offset]= R_{source}
16. system RET	SRET	upon condition: switches back to user mode and retrieves PC

I/O instructions are used only by the operating system to communicate with other chips and/or processors over the board.

A bit in the condition code register indicates the processor mode. If one of the privileged instructions is detected while in user mode it is treated as if it was a **N**o **OP**eration (NOP) instruction. This bit as well as another flag for the program counter are only accessible when in system mode.

Implementation

The design was implemented using the sea-of-gates technology as already mentioned. This is a prefabricated “image” consisting of MOS transistors. Using the OCEAN tools [5] to implement a circuit reduces to interconnecting these transistors with metal wires.

The full layout with all the routing channels for the internal buses covered the area of about 120 000 transistors. All the internal buses are 24–bits wide. Due to pins limitations multiplexing the data and address buses was necessary at the output stage.The layout and interconnections of the basic cells was done manually, and for these, efficient area utilization was an important factor as well as regularity of the whole design.

The simulations were done using the switch level simulator SLS available with OCEAN. This powerful simulator, used on an HP workstation, allows the simulation of the whole processor on the transistor level while reading and executing a multiplication program taking a real time of 30 micro seconds within two minutes only.

However, the basic cell of the register file got special attention because of its effect on the overall speed of the processor and the complications of its design. That’s why several iterations were needed to design it, simulate it using SPICE, evaluate the performance and redesign it. Simulation results show that an external two phase clock of 10 MHz can be used to supply the processor when it has a 50pf capacitive load on its pins.

The rising and falling edges of both phases are used to synchronize different events within the pipeline, the control unit and the output pins interface. The pipeline is composed of three stages which are performed each clock cycle. This corresponds to state T0 in the control unit which is repeated till an instruction of a type other than the ALU instructions is detected in stage two of the pipe. Only the LOAD, STORE, IN and OUT instructions use more than 3 clocks and thus stall the pipeline causing the control unit to shift to other states than the simple T0.

The different conditionally executed instructions which alter the value of the program counter(PC) needed to be finished within three clock cycles, too. To accomplish this, a bit in the condition codes register is used to enable or disable the copying of the PC to register R_{15}. Normally PC is copied to R_{15} every clock cycle, this allows relative addressing by using R_{15} as the base. When a CALL, RET or SYS instruction is to be executed, this copying is disabled, the address of the called routine is stored in R_{13}, and control is transferred to the system code. The operating system thus has full control over calling and returning from subroutines and can make any desired checks, saving of the task data, etc. Then it uses R_{15} to know the return address and R_{13} to know the destination address and switches to the user code again.

This idea together with the use of R_0 for a large number of functions enabled a very efficient and easy implementation of the pipeline and gave much power to the addressing mode used (R_{base} + signed offset).

Conclusion

The processor was sent for fabrication last March and it passed the preliminary tests, but in the final tests before fabrication in May a misalignment in the metalization layers of one of the cells was detected. This fault is corrected now and it will go to the next run. This is real life!! Nothing is easy as it appears to be when simulating!!!

However, to stress again on the basics, the processor has the following main features:

24-bit RISC microprocessor, clock speed 10MHz with 50pf capacitive loading, 3-stage pipelined architecture, throughput of 1 instruction per clock cycle, 5V supply, 2-layer metal, static CMOS, semi-custom VLSI chip, 1.6 micron technology, 24–bit multiplexed address/data bus, hardware protected instructions and registers for the operating system, 16 general purpose registers 24 bits each, supporting up to 16 MegaWords (48 MegaBytes) of physical memory.

Acknowledgments:
The members of the project wish to thank their supervisors: Prof.Dr. M.S. Metwally and Dr. A. Badawi. Thanks are due also to Dr. A.E. Salama for his help and encouragement.

To contact with the author via email send to: fahmy@cairo.eun.eg

References

[1] Stephen B. Furber, VLSI RISC architecture and organization, Marcel Dekker Inc., 1989.

[2] D.A. Patterson, “Reduced Instruction Set Computers”,Communications of the ACM, vol. 28, No. 1, Jan. 1985, pp 8–21.

[3] M. Morris Mano, Computer engineering Hardware design, Prentice Hall, 1988, (chap. 7–10).

[4] V. Carl Hamacher; Zvonko G. Vranesic; Safwat G. Zaky, Computer organization, McGraw Hill, 1990, (chap. 11).

[5] P. Groeneveld and P. Stravers, Ocean: the Sea-of-Gates Design System.
This is the manual of OCEAN. The OCEAN package is free software and can be retrieved via anonymous ftp: donau.et.tudelft.nl, directory pub/ocean.