A 16 bit RISC processor

by
Nebu John Mathai
mathai@ieee.org

I. Introduction

This design is a 16-bit processor, ie, the data and instruction word sizes are 16 bits. It is pipelined to a depth of 5 stages. The instruction set implements ALU, immediate, load- store, and branch instructions.

The processor is divided into 6 units: the instruction fetch unit (ifetch.vhd), the decoder (decode.vhd), the branch condition detector (branch.vhd), the ALU (alu.vhd), the register file (regfile.vhd), and the data load-store unit (dls.vhd). The units are connected together with appropriate glue logic in system.vhd.

For easier conversion of the design to a DSP (digital signal processor), the processor implements the Harvard architecture of separate instruction and data memory spaces. For this prototype, main memory is simulated by a server program running on a PC that serves requests for transactions to the data and instruction memory spaces. For practical reasons, namely the availability of only one external bus to the PC, the bus transfers from the processor are serialized by an arbiter to transfer over one bus. However, logically from the point of view of both the processor and the simulated memory device, there are two separate memory spaces.

II. Pipeline

The processor has a 5 stage pipeline, with stages corresponding to: Instruction Fetch, Instruction Decode and Register Fetch, ALU Operation, Memory Operation, and Register File Write. The pipeline was implemented by placing registers following the instruction fetch, decode, ALU, and data load-store units, and enabling these units when the global signal stall is deasserted. Only two units can stall the pipeline, the instruction fetch and data load-store units. On a stall, the stalling unit will raise a stall signal which will disable all pipeline registers to maintain the processor's state until stall's deassertion.

Read After Write (RAW) hazards are solved by register forwarding. The source register addresses of instruction i are compared with the destination register address of instruction i-1, and an appropriate signal is sent to the multiplexor feeding the ALU source 0 or source 1 input. The multiplexor can select between either the register file or the contents of the ALU result register (which corresponds to what the destination register should be for instruction i-1). Hence, successive reads from registers that were the destination of preceding ALU operations occur correctly. Due to the difference in position of the "ALU Operation" and "Memory Operation" pipeline stages, there is a one slot penalty when the preceding operation is a "load" memory operation (see Appendix).

To reduce the branch delay penalty to one, the branch condition is computed during the fetch stage of branch, and the new program counter is loaded by the decode stage of branch. Hence, fetching from the branch target will occur one cycle after branch is received; only one additional instruction will be fetched in the meantime. The drawback of this is that for correctness, the register file must contain the correct condition by the time branch is fetched; the last instruction modifying branch conditions or target addresses, must occur 4 instructions before a branch (see Appendix).

III. Memory Subsystem

The processor implements the Harvard memory architecture, and so the instruction and data memory spaces are both physically and logically separate. This was done to facilitate easy conversion of the design to a digital signal processor.

Space was not available to incorporate instruction and data caches, so all memory transfers occur directly to memory; data and instruction transfer collisions are resolved by an arbiter circuit.

The arbiter circuit is a simple 5 state FSM that serializes requests from the data and instruction units. This is necessary since this prototype of the design and it's memory has only one external bus. The arbiter communicates requests to a bus master circuit.

The bus master circuit is composed of two independent FSMs (one 3 state FSM indicates the current operation of the bus master: idle, read or write; another 9 state FSM implements the bus protocol in which four 8-bit packets, representing 16-bit address and 16-bit data words, are transferred) that perform the operation requested by arbiter.

IV. Implementation Issues on XC4005XL FPGA

i. The register file of sixteen 16 bit registers would not fit on the device, so the design only includes four registers (register 0 having an unalterable value of 0).

ii. An ALU including an adder, subtractor, and/or multiplier would not fit on the device, hence only logical ALU operations can be performed (with this implementation).

iii. Caches for the data and instruction units would not fit on the device.

V. Additional Requirements

i. Circuitry

The following connections to the PC parallel port bus (for the memory transfers), as well as switches and LEDs are either required or facilitate debugging:

NAME	XS40-005XL PIN	PARALLEL PORT PIN	ADDITIONAL INFO
resetbar	68	none	to active low switch
clkspeed	62	none	to switch
ack	38	16	none
rdy	39	13	10kohm pulldown
mode	40	12	10 kohm pulldown
typeo	41	10	10 kohm pulldown
drivemode	3	none	to LED and 1kohm to ground (inc
ad0	27	2	none
ad1	28	3	none
ad2	79	4	none
ad3	80	5	none
ad4	81	6	none
ad5	82	7	none
ad6	83	8	none
ad7	84	9	none
pcdrive	none	14	to LED and 1kohm to ground (inc

ii. Software

A server program is required to run on a PC. It will listen to the parallel port for requests from the processor, and serve the requests. The files: Makefile, server16.c and hiz.c, implement the server (*server*) and a utility (*hiz*) to put the parallel port in high impedance mode. To create a program for the processor create two files with the following format:

```
HEXADDRESS   HEXOPCODE or HEXWORD
```

Then run the server: server INSTRUCTIONFILE DATAFILE OUTPUTFILE

Requests for the instruction memory will be satisfied with data from INSTRUCTIONFILE, and requests for the data ram will be satisfied with data from DATAFILE. A file called trace will be created, logging all transactions. Issuing '-1' while the server is running causes the server to quit, and write the current status of data memory to OUTPUTFILE. The files 'ins', and 'dat' are sample files to test out the processor with.

Note: the binaries provided are for a GNU Linux x86 system. The software was developed under GNU Linux. To build the software, issue "make" in a directory where Makefile, server16.c and hiz.c can be found. I do not have a Windows system at my disposal to port the software.

VI. Instruction Set Summary

Convention:

```
rX refers to the address of a register
[rX] refers to the value stored by the register addressed by rX
*[rX] refers to the memory location referenced by the value [rX]
```

i. ALU INSTRUCTIONS

```
Format:
instruction rsrc0,rsrc1,rdest
Operands:
rdest addresses the destination register
rsrc0 and rsrc1 address the source registers
Operation:
[rdest] gets [rsrc0] instruction [rsrc1]
```

INSTRUCTION	OPCODE
add	0
subtract	1
multiply	2
and	3
or	4
not	5
xor	6

ii. IMMEDIATE INSTRUCTIONS

```
Format:
instruction immed,rdest
Operands:
immed is an 8 bit immediate value
rdest addresses the destination register
Operation:
movil (movih) moves the immed into the lower (upper) 8 bits of [rdest]
```

INSTRUCTION	OPCODE
movil	7
movih	8

iii. LOAD AND STORE

```
Operands:
radx addresses the register containing an address to memory
rdata addresses the register to (from) which data is to be
loaded (stored) from (to) memory
Hazards:
(load only) [rdata] does not contain the loaded data until
after the instruction following a load (RAW hazard)
```

INSTRUCTION	OPCODE	DESCRIPTION
load empty,radx,rdata	9	[rdata] gets *[radx]
store rdata,radx,empty	a	*[radx] gets [rdata]

iv. BRANCH

```
Operands:
[r1] contains condition 0
[r2] contains condition 1
[r3] contains the label to branch to (address of the
instruction to branch to in main memory)
Operation:
I. if branch is taken: [program counter] gets [r3]
II. branch is taken when [r1] condition [r2]
Hazards:
I. there is one branch delay slot
II. the branch condition and labels depend on the state of
r1, r2, and r3 four instructions prior to the branch instruction.
That is, for a branch operation to logically occur as written,
```

`the preceding three instructions to a branch must not alter r1, r2, and r3.`

INSTRUCTION	OPCODE	DESCRIPTION
bgt	b	condition is greater than
blt	c	condition is less than
beq	d	condition is equal to
bne	e	condition is not equal to