

Harnessing FPGAs for Computer Architecture Education



Mark Holland {mholland@ee.washington.edu}, James Harris {harrisj@u.washington.edu}, Scott Hauck {hauck@ee.washington.edu}
 Department of Electrical Engineering, University of Washington, Seattle, WA, 98195, USA



Motivation

- In computer architecture and design, a common university level course, students use software to design and simulate individual pieces of a computer processor
- The work of students is limited to software simulations, while we feel that the students would benefit greatly from actually designing, implementing, and running their own processors

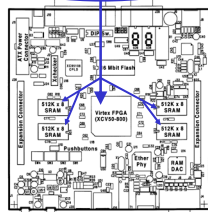
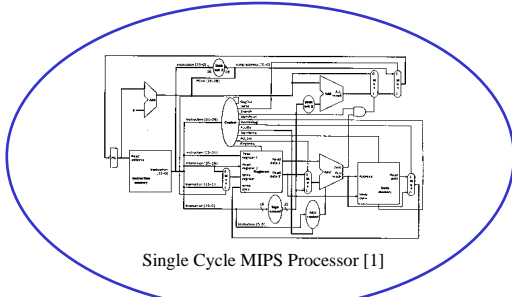
Solution

- Program selected CPU parts onto an FPGA, give the incomplete processors to the students, and allow them to design and integrate the missing pieces
- Requires FPGA-optimized MIPS processor, processor debugging tool, and cross-compiler for mapping any MIPS instruction to our limited set

Implementation of 32-bit MIPS processor on an FPGA

Used XESS XSV Board

- XILINX Virtex XCV300 FPGA for control and most datapath elements
- Four 512K x 8 bit SRAM banks for instruction and data memories
- Push-button switch for resetting the processor
- Parallel port interface which allows communication with the debugging tool

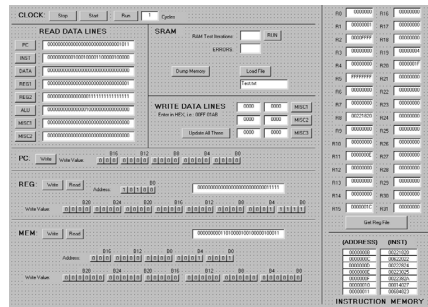


PROCESSOR TYPE	FPGA CLOCK	PROC. CLOCK	% LUTS USED	% BRAM USED
SINGLE-CYCLE MIPS	25 MHz	1.5MHz	22%	25%
PIPELINED MIPS	23 MHz	1.4MHz	29%	25%

Processor Implementation Results on XILINX Virtex XCV300 FPGA

Debugging Tool to allow students to debug their incomplete processors

- Written using Visual Basic 6.0
- Communicates with the XSV board over a parallel port
- Allows the student to observe and control all parts of the processor

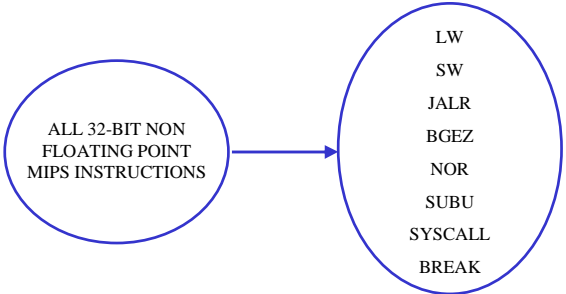


DEBUGGING OPERATION	RESULT
CLOCK	
Run	Runs the processor clock for a specified number of cycles.
Start	Starts the processor clock.
Stop	Stops the processor clock.
READ DATA LINES	
PC	Returns the current value of the program counter.
INST	Returns the current output of the instruction memory.
DATA	Returns the current output of the data memory.
REG1	Returns the current output of register 1.
REG2	Returns the current output of register 2.
ALU	Returns the current output of the datapath ALU.
MISC1	Returns the current output of the first miscellaneous line (hooked up by the user to any output).
MISC2	Returns the current output of the second miscellaneous line (hooked up by the user to any output).
SRAM	
RUN	Runs a routine that tests the SRAM banks.
Dump Memory	Dumps the contents of the SRAM to a file.
Load File	Loads the specified file into the SRAM.
WRITE DATA LINES	
MISC1	Set the first miscellaneous input line to a specified value (hooked up by the user to any input).
MISC2	Set the second miscellaneous input line to a specified value (hooked up by the user to any input).
MISC3	Set the third miscellaneous input line to a specified value (hooked up by the user to any input).
Update All These	Update all three miscellaneous lines.
PC: Write	Write a value to the program counter.
REG: Write	Write a value to any register.
REG: Read	Read a value from any register.
MEM: Write	Write a value to any instruction or data address.
MEM: Read	Read a value from any instruction or data address.
Get Reg File	Obtain the complete state of the register file.
INSTRUCTION MEMORY	Observe the instructions near the current instruction.

Debugging Tool Operations

Cross Compiler for reducing all non-floating point MIPS instructions to our set.

- Written using LEX and YACC



INSTRUCTION	FORMAT	OPERATION
Nor	NOR rd, rs, rt	perform bit-wise logical NOR on the contents of register rs and rt, place the result in rd
Subtract Unsigned	SUBU rd, rs, rt	subtract the contents of rt from rs (using unsigned subtraction), place the result in rd
Load Word	LW rd, offset(base)	load into rd the value from memory location that is the sum of offset and the value in register: base
Store Word	SW rd, offset(base)	store the value from register rd into the memory location that is the sum of offset and the value in register base
Branch on Greater Than or Equal to 0	BGEZ rs, offset	if the contents of rs == 0, branch to (PC + offset)
Jump and Link Register	JALR rd, rs	jump to the address in register rs, placing the next instruction in rd
System Call	SYSCALL	a system call exception transferring control to the exception handler.
Break	BREAK	a breakpoint trap occurs, transferring control to the exception handler.

Supported Instructions

INST	TOTAL	AVG	INST	TOTAL	AVG	INST	TOTAL	AVG	INST	TOTAL	AVG
LB	13	12.5	J	2	2	ORI	3	3	ORL	30	32.5
LBU	8	9	JAL	2	2	ORI	2	2	SLLI	9	5.83
LH	15	9	JR	1	1	XORI	6	6	SRLIU	9	6
LHU	9	6.5	JALR	1	1	XORI	5	5	SRT	8	4.83
LW	1	1	SECO	5	4	NOR	1	1	SLTU	9	5
LW	22	47.25	BNE	6	3.63	ADDI	2	2	SLL	9	51.53
LWR	54	81.75	BLEZ	4	2.5	ADDUI	2	2	SRL	16	64.13
LUI	1	1	BGTZ	5	2.75	ADD	17	5.9	SRA	17	65.09
LJ	1	1	BLTZ	2	1.5	ADDU	2	2	SLLV	8	52.53
SB	29	17.75	BGEZ	1	1	SUB	2	2	SRLV	15	63.13
SH	15	9.5	BLTZAL	3	2.5	SUBU	1	1	SRAV	16	64.09
SW	1	1	BIGZAL	2	2	MULT	39	212	SYSCALL	1	1
SWR	25	14	AND	3	3	MULTU	26	336.8	BREAK	1	1
SWL	26	16.25	ANDI	4	4	DV	38	356.8			

TOTAL = Number of instructions used to replace the given instruction
 AVG = Average number of instructions executed in place of the given instruction.
 Note that the AVG can be greater than the TOTAL code expansion because of loops within the code.

Instruction Expansion

[1] Taken from *Computer Organization and Design: The Hardware/Software Interface* by Patterson and Hennessy
 [2] Taken from the *XSV Board V1.0 Manual*, XESS Corporation