Instruction Set Architecture of a MIPS based 16-bit RISC Processor

Nirmal Haldikar, Sooraj Sekhar

Abstract— Microcontrollers and microprocessors are finding their way into almost every field in today's world, incorporating an element of smartness into conventional devices. Energy efficient, space efficient and optimized microcontrollers are the need of the day. Our paper proposes a new Instruction Set that is a subset of the MIPS architecture. It derives the advantages of MIPS like simplicity and speed. Besides, since it is a smartly optimized subset of MIPS, it is a smaller version consisting of the most commonly required instructions.

Index Terms— ISA, MIPS, Processor design, RISC, Operand, Opcode, Pipeline.

1 Introduction

MIPS is a reduced instructions set computer (RISC) architecture. It is one of the first RISC Instruction set architectures. MIPS is an acronym for "Microprocessor without interlocked pipeline stages". A team led by John Hennessey at Stanford University developed it. MIPS implementations are primarily used in embedded systems such as Windows CE devices, routers, residential gateways, and video game consoles such as the Sony PlayStation 2 and PlayStation Portable. Until late 2006, they were also used in many of SGI's computer products. Digital Equipment Corporation, NEC, Pyramid Technology, Siemens Nixdorf, Tandem Computers and others also used MIPS implementations during the late 1980s and 1990s. Since MIPS is a RISC computer it employs less number of transistors and hence decreases the transistor count. Pipelining is thus heavily employed to make use of extra available space on the chip to improve code execution performance. MIPS was defined to be a 32-bit architecture called MIPS32. Later Revisions of this architecture is 64 bit in size and hence called MIPS64. [1]

2 MIPS 16 Instruction Set Description

2.1 Motivation

Small-scale applications do not require that much of computing power. This paper proposes a reduced version of MIPS instruction set for such small-scale applications. This ISA will be called MIPS 16. The main aim of this ISA is to reduce the transistor count of a MIPS processing unit by scaling down the bus and register width and providing less but enough number of instructions for small-scale applications. The implementation of such an instruction set would take up less real estate on the chip (or FPGA) and will allow more peripherals to be fabricated on a single chip making it ideal for a System-On-Chip (SOC) implementation of an application. It will also be beneficial in embedded system design where a custom processor core implementation is required with tight instruction requirements so that it takes less space on a FPGA.

- *Nirmal Haldikar is currently pursuing a post graduate diploma (VLSI Design) from Centre for Development Of Advanced Computing, Pune and has a Bachelor in Engineering degree(Electronics) from Datta Meghe College Of Engineering Mumbai.Email:nirmalh91@gmail.com*
- *Sooraj Sekhar is currently pursuing a post graduate diploma (VLSI Design) from Centre for Development Of Advanced Computing, Pune and has a Bachelor in Technology degree (Electronics and Communication) from Lovely Professional University, Jalandhar.Email:soorajsekhar31@gmail.com*

2.2 Instruction Set Specification [3]

MIPS instructions have fixed width. The original MIPS 32 ISA has 32 bits wide instructions. Each instruction in MIPS16 is 16 bits wide. Further, MIPS16 has 8 internal registers as opposed to the 32 registers of MIPS32. As the name suggests, data bus is 16 bits wide and address bus is preferably 16 bits wide too. I/O support is memory mapped. Memory is accessed by LOAD and STORE instructions. The instructions follow an <operand register, register, register> format.

The Instructions can be divided into 4 groups:

1. Arithmetic: Basic computational instructions add and subtract.
2. Logical: Operations like AND, OR, EXOR
3. Data Transfer: Load and Store operations
4. Branch and control: Jump, Call, and Return, etc.The ISA supports direct and immediate addressing modes.

2.3 Instruction Word Format

A MIPS16 instruction is 16 bits wide. Since MIPS uses a Register-Register type of instruction a general instruction specifies two source registers and a destination registers. The format of such an instruction will be

ADD R_{s1}, R_{s2}, R_d
R_{s1} = First source operand register
R_{s2} = Second Source operand register
R_d = Destination register

The instruction word has a 5-bit op-code specifying the operation to be performed. Number of operands may be variable e.g. ADD requires three operands while NOT requires only two. Format of a three-operand instruction word is shown in Table I

International Journal of Scientific & Engineering Research Volume 4, Issue 12, December-2013
ISSN 2229-5518

TABLE 1
THREE OPERAND INSTRUCTION

Op-code	Rs1	Rs2	Rd	Reserved
5 bits	3 bits	3 bits	3 bits	2 bits

In case of ALU instructions, the 2 reserved bits act as function bits where they are used to distinguish between versions of a common instruction. For instance, the instructions ADD and ADC have the same opcode but different function bits. This results in simpler control logic as the reserved bits are decoded directly by the ALU control logic.

In case of lesser operands appropriate operand is given a constant value. E.g. NOT instruction requires only one source and one destination operand. Therefore, Rs2 field will be made "000" as shown in Table 2. Likewise a POP instruction will require only destination and hence both the source operands will be constant and only destination

TABLE 2
TWO OPERAND INSTRUCTION

Op-code	R_s	Constant	R_d	Reserved
5 bits	3 bits	000	3 bits	2 bits

NOT R_s, R_d

needs to be provided as shown in Table 3.

TABLE 3
ONE OPERAND INSTRUCTION

Op-code	Constant	Constant	R_d	Reserved
5 bits	000	000	3 bits	2 bits

POP R_d

The unused fields in an instruction are also used to provide immediate input. The size of the immediate field depends on the number of operands instruction uses.

ADDI R_s, R_d, #10
R_s = Source Register
R_d = Destination register
#10 = Immediate value (0-31 in decimal)

TABLE 4
IMMEDIATE INSTRUCTION

Op-code	R_s	R_d	Immediate
5 bits	3 bits	3 bits	5 bits

Instructions like the move immediate (MVIH / MVIL) require an 8-bit value to be specified within the instruction. In such a case, the 8-bit value is split into 2 parts. The higher 3 bits are specified in place of the R_{s1} operand and the next lower 5 bits are specified in the lower 5 bits of the instruction.

TABLE 5
MOVE IMMEDIATE INSTRUCTION

Op-code	R_s	R_d	Immediate
5 bits	7:5 bits of imm value	3 bits	4:0 bits of imm value

Jump instructions have two modes viz. PC relative and absolute modes. In PC relative mode. The lower 5 bits of the instruction are used to specify a 5 bit signed value as shown in Table IV. This value is added/subtracted to the PC to get the jump address. The PC relative mode is used for conditional jump instructions. The absolute mode is used for unconditional jumps and jump-&-link instructions. In these instructions the all bits other than the opcode are used to specify an 11 bit signed PC offset value.

The instruction set is so designed so as to simplify the instruction decoding logic and the control logic.

TABLE 6
JUMP (ABSOLUTE MODE) INSTRUCTION

Op-code	Immediate
5 bits	11 it signed PC Relative offset

2.4 Comparison between MIPS -16 and MIPS-32 [7]

MIPS-16 can be considered to be a derivative of MIPS-32 instruction set. But the philosophies behind their design are different. MIPS-16 provides more flexibility in terms of optimizing the design by keeping only the required instructions. MIPS-16 is designed for small-scale applications while MIPS-32 is a high performance 32-bit architecture, which can handle large data and perform fast calculations by employing multiple pipelines and multiple registers at the cost of larger chip area and complicated logic design.

TABLE 7
COMPARISON OF MIPS-16 AND MIPS-32 ISA

Serial No.	MIPS-16 Instruction Set	MIPS-32 Instruction Set
1	Instruction word length is 16-bit	Instruction word length is 32-bit
2	Supports only 8 general purpose registers	Supports up to 32 general purpose registers
3	Register is 16 bits wide	Register is 32 bits wide
4	Program counter should be incremented by 2 after every instruction (for non-branching instructions)	Program counter should be incremented by 4 after every instruction (for non-branching instructions)
5	ALU is simpler. It does not support operations with bulky logic like Multiplication and Division	ALU is Complicated. It supports complicated operations like Multiplication and Division.
6	Floating point instructions are not included in MIPS-16	Floating Point instructions are included and are called SIMD instructions

Some key differences have been highlighted in TABLE 7

International Journal of Scientific & Engineering Research Volume 4, Issue 12, December-2013
ISSN 2229-5518

TABLE 7 (CONTINUED)
COMPARISON OF MIPS-16 AND MIPS-32 ISA

7	Pipelining is not essential and depends on the application.	Pipelining is a key feature of a MIPS-32 based processor.
8	Transistor count and chip area is less	Transistor count and chip area is more
9	Suitable for small scale applications and applications with low computing requirement	Suitable for high performance high throughput applications.

2.5 List of Instructions

As the op-code has a 5-bit length there are 32 possible distinct instructions. If the reserved bits at the end of the instruction are utilized for grouping 2 or more similar instructions more op-codes can be incorporated in the instruction set e.g. ADD and ADC can be grouped together as they perform similar function with the difference being inclusion of carry into the sum. A complete list of 37 instructions has been provided in Table 8 with a short description of each instruction.

TABLE 8
MIPS-16 INSTRUCTION SET

Sr. No	Mnemonic	Instruction Format	Description
1.	ADD	ADD R_{s1}, R_{s2} ,R_d	Adds R_{s1} and R_{s2} and stores the sum in R_d ignoring carry.
2.	ADC	ADC R_{s1}, R_{s2} ,R_d	Adds R_{s1} and R_{s2} and stores the sum in R_d with previous carry.
3.	SUB	SUB R_{s1},R_{s2},R_d	Subtracts R_{s2} from R_{s1} and stores the difference in Rd ignoring the previous borrow.
4.	SBB	SUB R_{s1}, R_{s2} ,R_d	Subtracts R_{s2} from $R_{s1}$and stores the difference in Rd with the previous borrow.
5.	AND	AND R_{s1}, R_{s2} ,R_d	Performs Bitwise AND of R_{s1} and R_{s2} and stores the result in R_d
6.	OR	OR R_{s1}, R_{s2} ,R_d	Performs Bitwise OR of R_{s1} and R_{s2} and stores the result in R_d
7.	XOR	XOR R_{s1}, R_{s2} ,R_d	Performs Bitwise XOR of R_{s1} and R_{s2} and stores the result in R_d
8.	NOT	NOT R_{s1} ,R_d	Performs Complement of R_{s1} and stores the result in R_d
9.	SHIFTL	SHIFTL R_{s1} ,R_d	Shifts R_{s1} by one place to the left and store it in R_d
10.	SHIFTR	SHIFTR R_{s1} ,R_d	Shifts R_{s1} by one place to the right and store it in R_d
11.	ADDI	ADDI R_{s1} ,R_d #5-bit	Adds a 5-bit unsigned value to R_{s1} and stores the sum in R_d
12.	SUBI	SUBI R_{s1} ,R_d #5-bit	Subtracts a 5-bit unsigned value from R_{s1} and stores the difference in R_d
13.	MOV	MOV R_{s1} ,R_d	Copies R_{s1} to R_d
14.	MVIH	MVIH R_d,#8-bit	Copies immediate value into higher byte of R_d
15.	MVIL	MVIL R_d,#8-bit	Copies immediate value into lower byte of R_d
16.	LDIDR	LDIDR R_{s1} ,R_d #5-bit	Loads R_d with a nibble at address given by [R_{s1} +5 bit immediate value]
17.	STIDR	STIDR R_{s1} ,R_d #5-bit	Stores R_d with a nibble at address given by [R_{s1} +5 bit immediate value]
18.	LDIDX	LDIDX R_{s1}, R_{s2} ,R_d	Loads R_d with a nibble at address given by [R_{s1} + R_{s2}]
19.	STIDX	STIDX R_{s1}, R_{s2} ,R_d	Stores R_d with a nibble at address given by [R_{s1} + R_{s2}]
20.	JMP	JMP #11-bit	Unconditional jump to address offset by 11 bit signed value from current PC value
21.	JMPI	JMPI R_d,#15	Unconditional jump to address offset by 5 bit signed value added to R_d
22.	JGEO	JGEO R_{s1} ,R_{s2} #5-bit	Conditional Jump to PC + 5 bit signed offset if R_{s1} is greater than or equal to R_{s2}
23.	JLEO	JLEO R_{s1} ,R_{s2} #5-bit	Conditional Jump to PC + 5 bit signed offset if R_{s1} is less than or equal to R_{s2}
24.	JCO	JCO #5-bit	Conditional Jump to PC + 5 bit signed offset if carry is set
25.	JEO	JEO R_{s1} ,R_{s2} #5-bit	Conditional Jump to PC + 5 bit signed offset if R_{s1} equals to R_{s2}
26.	PUSH	PUSH R_{s1}	Push R_{s1} to the stack top and update stack top
27.	POP	POP R_d	Pop from the stack top and store the value to R_d and update stack top
28.	CALL	CALL R_{s1}	Calls a subroutine located at [R_{s1}]. Return address is pushed onto stack
29.	JAL	JAL #11-bit	Calls a subroutine located at [PC + 11 bit signed offset].Return address pushed onto stack
30.	MOVSP	MOVSP R_{s1}	Copies value at R_{s1} to stack pointer SP
31.	RET	RET	Return from a function. Return address is popped from the stack
32.	STC	STC	Set the carry flag
33.	NOP	NOP	No operation. Idle machine cycle should be executed
34.	HLT	HLT	Halts the processor.
35.	RST	RST	Resets the processor.
36.	IE	IE	Enables the interrupt.
37.	ID	ID	Disables the interrupt.

International Journal of Scientific & Engineering Research Volume 4, Issue 12, December-2013
ISSN 2229-5518

2.6 Implementation Strategies

The implementation strategies that can be employed will depend on the application. Some of the design considerations are listed below:

1. Single-Cycle or Multi-cycle implementation: [3], [4],[5]

   Implementation can use a single cycle or a multi cycle control system for its data path.

   A single cycle control system performs all the elementary data path operations in a single cycle. This generally requires dedicated hardware for every phase of instruction fetching, decoding and execution. It is faster at the cost of larger chip area.

   A multi cycle implementation divides the execution of an instruction into well-defined time states. The execution happens in a timely order and might require different number of time states for different instruction. The main advantage is the hardware can be shared for similar elementary functions in different time states. Multi cycle should be preferred for applications with smaller chip area requirements.

2. Pipelining Requirement:[6]

   Pipelining provides performance enhancement by concurrent execution of more than one pair able instructions. The design involves use of multiple data paths and logic to check for pair ability and hazard removal that occurs due to concurrent execution. This significantly complicates the design and takes a larger chip area. But the performance improvement would be tremendous.

3 CONCLUSION

MIPS-16 is thus a low-cost, compact and hence in effect a low power RISC instruction set architecture as compared to the MIPS-32 architecture. Its compact size and flexibility makes it ideal for an optimized implementation of an embedded system. It provides all the basic instruction and functionality for a small-scale embedded system not involving heavy arithmetic calculations. MIPS-16 can be implemented on FPGA by an appropriate strategy as per the application's requirement. Single cycle design should be used for better performance while multi cycle design should be preferred for compactness. Pipelining can further improve the system performance.

ACKNOWLEDGMENTS

It gives us immense pleasure to thank Prof Parikshit Godbole, who always motivated us. He was kind enough to help us with our doubts and was a source of inspiration for this project.

REFERENCES

[1] http://en.wikipedia.org/wiki/MIPS_architecture.
[2] http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm
[3] David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 3rd Ed.
[4] Lecture notes by Howard Huang, University of Illinois at Urbana-Champaign. [Online]. Available: http://www.howardhuang.us/teaching/cs232/11-Single-cycle-MIPS- processor.pdf
[5] Lecture notes by Howard Huang, University of Illinois at Urbana-Champaign. [Online]. Available: http://www.howardhuang.us/teaching/cs232/12-Multicycle-datapath.pdf
[6] Lecture notes by Howard Huang, University of Illinois at Urbana-Champaign. [Online]. Available: http://www.howardhuang.us/teaching/cs232/15-Pipelining.pdf
[7] MIPS Official Website. Available: http://www.mips.com/products/architectures/mips32/