Alexandria University
Faculty of Engineering
Computer and Systems Engineering Department
CS 402 - Graduation Project

MIPS Home Computer System

System Design Reference

Student
Mostafa Abd El-Aziz

Supervisor
Prof. Dr. Layla Abou Hadid

Table of Contents

1 Central Processing Unit **1**
1.1 Introduction 2
1.1.1 Load and Store Instructions 3
1.1.2 ALU Instructions 4
1.1.3 Jump and Branch Instructions 7
1.1.4 Miscellaneous Instructions 9
1.1.5 Control Instructions 10
1.1.6 Instruction Encoding Summary 10
1.1.7 Addressing Modes 12
1.1.8 Programmer-Visible Pipeline Effects 13
1.2 Registers 14
1.3 Exception Handling 14
1.4 Paging 14
1.5 Cache Management 14
1.6 VHDL Implementation: Introduction 14
1.7 VHDL Implementation: The Pipeline 14
1.8 VHDL Implementation: Exceptions 14
1.9 VHDL Implementation: Cache and TLB 14

1 Central Processing Unit

The CPU module implements MIPS-I instruction set architecture along with additional instructions that control the behavior of the processor. In this section we present detailed description for the programming interface of the CPU, followed by a description for the implementation of that interface.

Before going into details, we shall define two keywords that will be used throughout the section:

- **Architecture**: The behaviour of the processor as seen by the programmer. This includes the instruction set, their opcodes and formats, architectural registers, interrupt/exception

protocols, and other behavioral concerns that will be seen later in the section. The architecture describes all that can be abstracted from the implementation of the processor. Other authors may call this an instruction set architecture (ISA) or macro-architecture.

- **Micro-architecture**: How the architecture of the processor is implemented in switching circuits. We use this keyword to refer to our VHDL implementation for the architecture described in this section. There may exist several different implementations for the same architecture. The only restriction on the micro-architecture of a processor is to behave exactly as described by the architecture, but the implementation itself doesn't matter. For example, the architecture might guarantee that the 'effect' of the execution of a set of instructions will be in order, however, the processor might execute the instructions in an out of order fashion (OoOE) yet effectively behaves the same as described by the architecture.

The following subsections are divided into two sets corresponding to the two titles described above; Until subsection 5 (inclusive), we are just talking about the architecture. It's not before subsection 6 where we start to show how we implement the behaviour described in earlier subsections. Programmers can skip subsections 6, 7, 8, and 9.

Here we summarize what the subsections are about (SS# = subsection #):

SS1. an introduction to the basics of the architecture of our CPU.

SS2. detailed description for the architectural registers.

SS3. interrupt and exception handling.

SS4. paging and translation look-aside buffer (TLB).

SS5. caching as seen by the programmer.

SS6. introduction to the VHDL implementation of our CPU.

SS7. VHDL implementation for the pipeline.

SS8. VHDL implementation for interrupts and exceptions.

SS9. VHDL implementation for the cache and TLB.

1.1 Introduction

CPU instruction (also called order, operation) is the basic unit of execution. The main task of a processor is to execute a sequence of instructions (i.e, program) which shall have effect on the state of the processor and devices connected with the processor.

Following Von Neumann's model, the program is stored in memory along with variables which are accessible and modifiable by the program. Memory is an array of words, where a word in MIPS is 32 bits. Every instruction in MIPS is encoded in a single word that contains the opcode and the operands of the instruction.

Every word in memory has an address, that also means that every instruction in memory has an address called 'instruction address'. CPU maintains a register that stores the instruction address of next instruction to fetch, called program counter (PC). When an instruction is fetched, program counter is increased by 4 (number of bytes in word) without touching lower 2 bits (they are always zero), a behaviour which implies that:

- Instructions are executed in sequence unless there is a branch, jump, or exception.
- Instructions should always be stored in 4-byte aligned words.

It is important to note that MIPS-I instruction set is a proper subset of the instruction set of our processor; meaning that the processor supports additional instructions that control the behaviour of the processor as will be seen below.

CPU instructions are divided into the following groups:

1. Load and Store
2. ALU
3. Jump and Branch
4. Miscellaneous
5. Control

1.1.1 Load and Store Instructions

This group holds all instructions related to memory access, either read/load or write/store accesses. Every instruction word contains the following fields:

- **Opcode**: specifies that this instruction is a memory instruction, along with whether this instruction is a byte, half-word, or a word instruction, and whether this instruction is an aligned/unaligned, load/store, and signed/unsigned instruction (6 bits).
- **Source/destination register**: If the instruction is a load instruction, this is the address of the register where the value loaded from memory will be stored. If the instruction is a store instruction, this is the address of the register where the value stored to memory will be loaded. Since there are 32 data registers specified by MIPS ISA, this field takes a value from 0 to 31 (5 bits).
- **Base register**: address of data register that contains the base memory address. Just like source/destination register field, this field takes a value from 0 to 31 (5 bits).
- **Index immediate**: A 16-bit signed integer that is extended to 32-bit signed integer then added to the value of the base register. Result is the memory address that will be accessed.

This instruction group has a total of 12 instructions, each instruction has a unique value for the opcode field:

1. **LB**: Load Byte
2. **LBU**: Load Byte Unsigned
3. **SB**: Store Byte
4. **LH**: Load Halfword
5. **LHU**: Load Halfword Unsigned

6. **SH**: Store Halfword
7. **LW**: Load Word
8. **SW**: Store Word
9. **LWL**: Load Word Left
10. **LWR**: Load Word Right
11. **SWL**: Store Word Left
12. **SWR**: Store Word Right

The detailed behaviour of those instructions is described in the appendix.

As described above, the target memory address is calculated by adding the extended 32-bit signed immediate to the value of the base register, forming a 32-bit memory address. This implies that memory address space of MIPS-I instruction set is 4GB. There are two characteristics that apply to the address field:

- **Alignment**: All instructions (except LWL, LWR, SWL, and SWR) are aligned. This means that LW and SW target address must be a multiple of 4, while LH, LHU, and SH target address must be a multiple of 2.
- **Endianness**: Our processor is little endian; the least significant byte of a word in memory is the byte having the lowest address, and the most significant byte is the byte having the highest address.

1.1.2 ALU Instructions

This group contains all instructions related to arithmetic and logical operations. We divide them into 4 categories, according to how they are encoded:

1. ALU instructions with an immediate operand
2. ALU instructions with three operands
3. Shift instructions
4. Multiply and divide instructions.

The reader may expect that the first group (ALU instructions with immediate operand) is encoded exactly the same way as memory instructions. Both groups are included in the same encoding class (called I-TYPE) as will be seen later in the encoding subsubsection. Just like memory instructions, an ALU instruction with an immediate operand shall consist of the following fields:

- **Opcode**: specifies that this instruction is an ALU with immediate operand instruction, along with the type of operation. (6 bits)
- **Source register**: except for LUI, all operations in this group are binary operations, thus this field contains the address of the register that contains the value of the first parameter of the binary operation. For LUI instruction, this field must be zero. (5 bits)

- **Destination register**: address of data register where the result of the operation will be stored. (5 bits)
- **Index immediate**: except for ANDI, ORI, XORI, and LUI instructions, this field is a 16-bit signed integer that is extended to 32-bit signed integer then used as the second parameter for the binary operation. For ANDI, ORI, and XORI, this field is a 16-bit unsigned integer that is extended to 32-bit unsigned integer then used as the second parameter for the binary operation. For LUI, this field is just copied into the highest 16 bits of the destination register.

Following is a listing for all ALU with immediate operand instructions specified by MIPS-I ISA, with every instruction having a unique value for the opcode field:

1. **ADDI**: Add Immediate Word
2. **ADDIU**: Add Immediate Unsigned Word
3. **SLTI**: Set on Less Than Immediate
4. **SLTIU**: Set on Less Than Immediate Unsigned
5. **ANDI**: And Immediate
6. **ORI**: Or Immediate
7. **XORI**: Exclusive Or Immediate
8. **LUI**: Load Upper Immediate

The detailed behaviour of those instructions is described in the appendix.

The remaining three groups of ALU instructions (ALU instructions with three operands, shift instructions, and multiply/divide instructions) are all encoded the same and have the following fields for each instruction:

- **Opcode**: always zero. (6 bits)
- **Source register 1**: except for SLL, SRL, SRA, SLLV, SRLV, SRAV, MFHI, MTHI, MFLO, and MTLO instructions, this field contains the address of the register that holds the value of the first parameter for the binary operation. For SLLV, SRLV, and SRAV, this field contains the address of the register that holds the amount of shift. For SLL, SRL, SRA, MFHI, and MFLO instructions, this field must be zero. For MTHI and MTLO instructions, this field contains the address of the register that holds the value that will be moved into the HI/LO register. (5 bits)
- **Source register 2**: except for SLL, SRL, SRA, SLLV, SRLV, SRAV, MFHI, MTHI, MFLO, and MTLO instructions, this field contains the address of the register that holds the value of the second parameter for the binary operation. For SLL, SRL, SRA, SLLV, SRLV, and SRAV instructions, this field contains the address of the register that holds the value to be shifted. For MFHI, MTHI, MFLO, and MTLO instructions, this field must be zero. (5 bits).

- **Destination register**: except for MULT, MULTU, DIV, DIVU, MFHI, MTHI, MFLO, and MTLO instructions, this field contains the address of the data register where the result of the binary operation or shift operation will be stored. For MFHI and MFLO instructions, value stored in LO/HI register will be moved into the register addressed by this field. For MULT, MULTU, DIV, DIVU, MTHI, and MTLO instructions, this field must be zero, since the result of these instructions is stored directly in LO register, HI register, or both. (5 bits)
- **Shift amount**: for SLL, SRL, and SRA instructions, this field simply contains the amount of shift. For all other instructions, this field must be zero. (5 bits)
- **Function code**: An extension to the opcode field, contains a unique code that defines the instruction to be performed. (6 bits).

All instructions encoded that way are called R-TYPE instructions. We will summarize all encoding classes later in this section.

With a unique function code for every instruction (and an opcode of 0), ALU instructions with 3 operands group includes the following instructions:

1. **ADD**: Add Word
2. **ADDU**: Add Unsigned Word
3. **SUB**: Subtract Word
4. **SUBU**: Subtract Unsigned Word
5. **SLT**: Set on Less Than
6. **SLTU**: Set on Less Than Unsigned
7. **AND**: And operation
8. **OR**: Or operation
9. **XOR**: Exclusive Or operation
10. **NOR**: Nor operation

The third group (shift instructions) contains the following instructions:

1. **SLL**: Shift Word Left Logical
2. **SRL**: Shift Word Right Logical
3. **SRA**: Shift Word Right Arithmetic
4. **SLLV**: Shift Word Left Logical Variable
5. **SRLV**: Shift Word Right Logical Variable
6. **SRAV**: Shift Word Right Arithmetic Variable

Finally, following is a listing for all instructions in the last group (multiply/divide instructions):

1. **MULT**: Multiply Word
2. **MULTU**: Multiply Unsigned Word
3. **DIV**: Divide Word
4. **DIVU**: Divide Unsigned Word
5. **MFHI**: Move From HI
6. **MTHI**: Move To HI
7. **MFLO**: Move From LO
8. **MTLO**: Move To LO

For detailed description for the behaviour of all of the instructions listed above, please refer to the appendix.

1.1.3 Jump and Branch Instructions

MIPS-I architecture defines a number of operations that control the path of execution. Jump instruction shall unconditionally modify PC register, while branch instructions modify PC register only when the condition specified within the instruction word is tautology.

Jump and branch instructions are divided into four groups:

1. Jump Instructions Jumping Within a 256MB Region
2. Jump Instructions to Absolute Address
3. PC-Relative Conditional Branch Instructions Comparing 2 Registers
4. PC-Relative Conditional Branch Instructions Comparing Against Zero

The way the instructions of the first group (jump instructions jumping within 256MB region) are encoded is different from I-TYPE and R-TYPE. It's called J-TYPE and consists of the following fields:

- **Opcode**: specifies that the instruction is a jump instruction of J-TYPE, along with whether the instruction should alter the link register or not. (6 bits)
- **Instruction Index**: a 26-bit integer that is multiplied by 4 to form the lowest 28-bit of the new value of PC register. The highest 4 bits are taken from the old value of PC, that's why the instructions of this group are said to jump within 256MB region (4GB/16).

This group only contains two instructions:

1. **J**: Jump
2. **JAL**: Jump and Link

Second group (jump instructions to absolute address) consists of instructions similar to instructions of the first group, yet have the ability to jump to any memory location, not just locations inside the current 256MB region. These instructions are encoded as R-TYPE like most ALU instructions, thus the instruction word shall consist of the following fields:

- **Opcode**: always zero. (6 bits)
- **Source register 1**: contains address of the register that holds the target 32-bit address. (5 bits)
- **Source register 2**: must be zero. (5 bits)
- **Destination register**: for JALR instruction, this field contains the address of the register where return address shall be stored. For JAL instruction, this field should be zero.
- **Shift amount**: must be zero. (5 bits)
- **Function code**: An extension to the opcode field, contains a unique code that defines the instruction to be performed (only jump or jump with link). (6 bits).

Analogous to the first group, this group consists of these two instructions:

1. **JR**: Jump Register
2. **JALR**: Jump and Link Register

The remaining two groups are concerned with conditional branching. The third group consists of instructions that compare two registers, then jump if the comparison yields a specific result, while the last group compare one register against zero, and jump if the comparison yields a specific result.

The third group are just encoded as I-TYPE (like ALU immediate instructions and load/store instructions). The fields of the instruction word are interpreted as following:

- **Opcode**: specifies that this instruction is a conditional branch instruction comparing 2 registers, along with the type of comparison to be performed. (6 bits)
- **Source register**: this field contains the address of the register that contains the value that will be used as the first operand for comparison. (5 bits)
- **Destination register**: except for BLEZ and BGTZ instructions, this field contains the address of the register that contains the value that will be used as the second operand for comparison. For BLEZ and BGTZ instructions, this field must be zero. (5 bits)
- **Index immediate**: a 16-bit signed integer that is extended to a 32-bit signed integer, then added to PC to form the effective address. Thus this group of instructions are 'PC-relative'.

The group consists of the following instructions:

1. **BEQ**: Branch on Equal
2. **BNE**: Branch on Not Equal
3. **BLEZ**: Branch on Less Than or Equal to Zero
4. **BGTZ**: Branch on Greater Than Zero

The fourth group is also encoded as I-TYPE instructions, however, the fields shall have special values and meanings:

- **Opcode**: always one. (6 bits)
- **Source register**: this field contains the address of the register that contains the value that will be compared against zero. (5 bits)
- **Destination register**: used as an extension to the opcode field. This field shall contain a unique identifier for the operation to be performed. (5 bits)
- **Index immediate**: a 16-bit signed integer that is extended to a 32-bit signed integer, then added to PC to form the effective address. Thus this group of instructions are 'PC-relative'.

Finally, this last group consists of the following instructions:

1. **BLTZ**: Branch on Less Than Zero
2. **BGEZ**: Branch on Greater Than or Equal Zero
3. **BLTZAL**: Branch on Less Than Zero and Link
4. **BGEZAL**: Branch on Greater Than or Equal Zero and Link

For detailed description for the behaviour of all of the instructions listed above, please refer to the appendix.

1.1.4 Miscellaneous Instructions

This group of instruction consists of instructions used to trigger exceptions by software. In MIPS-I ISA, this type of instructions is called 'exception instructions'.

When the CPU encounters one of the instructions in this group, it pauses execution of the current program and transfers control to the exception handler. This behaviour is similar to software interrupts in x86 architecture.

The format of this class of instructions is similar to R-TYPE, with the 'code' field replacing source register, target register, destination register, and shift amount fields:

- **Opcode**: always zero. (6 bits)
- **Code**: available for use as a software parameter to be examined by the exception handler by loading the instruction word into a register.
- **Function code**: An extension to the opcode field, contains a unique code that defines the instruction to be performed (only break or system call). (6 bits).

This group consists of the following instructions:

1. **SYSCALL**: System Call
2. **BREAK**: Breakpoint

For detailed description for the behaviour of all of the instructions listed above, please refer to the appendix.

1.1.5 Control Instructions

MIPS-I architecture provides an ability to define coprocessor instructions in the instruction address space. The architecture can contain up to 4 groups of coprocessor instructions. MIPS-I assigns system control purposes to coprocessor 0, thus we allocate coprocessor 0 instructions for system control. Coprocessor instructions are specified by system designer, not MIPS-I ISA itself.

In this document, we shall use the keyword 'control instructions' to refer to this group of instructions. Control instructions are R-TYPE instructions with the following fields:

- **Opcode**: always 16. (6 bits)
- **Source register 1**: used to specify type of instruction. (5 bits)
- **Source register 2**: for MFC0 and MTC0 instructions, this field specifies the address of the general-purpose register whose value will be transferred to a control register, or altered by the value of the control register specified by 'destination register' field. For other instructions, this field must be zero. (5 bits)
- **Destination register**: for MFC0 and MTC0 instructions, this field specifies the address of the control register whose value will be transferred to or altered by the general-purpose register specified in the previous field. For other instructions, this field must be zero. (5 bits)
- **Shift amount**: must be zero. (5 bits)
- **Function code**: used, along with source register 1 field, to specify the type of the instruction. (6 bits)

The following list shows the instructions contained in this group:

1. **MFC0**: Move From Control Register
2. **MTC0**: Move To Control Register
3. **TLBR**: Read TLB Entry at Index
4. **TLBWI**: Write TLB Entry at Index
5. **RFE**: Return From Exception

For detailed description for the behaviour of all of the instructions listed above, please refer to the appendix.

1.1.6 Instruction Encoding Summary

As seen above, all instructions are encoded using either R-TYPE, I-TYPE, or J-TYPE format. Figure 1 summarizes the three formats and their corresponding fields. Note that field use/meaning depends on the encoded instruction itself, not the format.

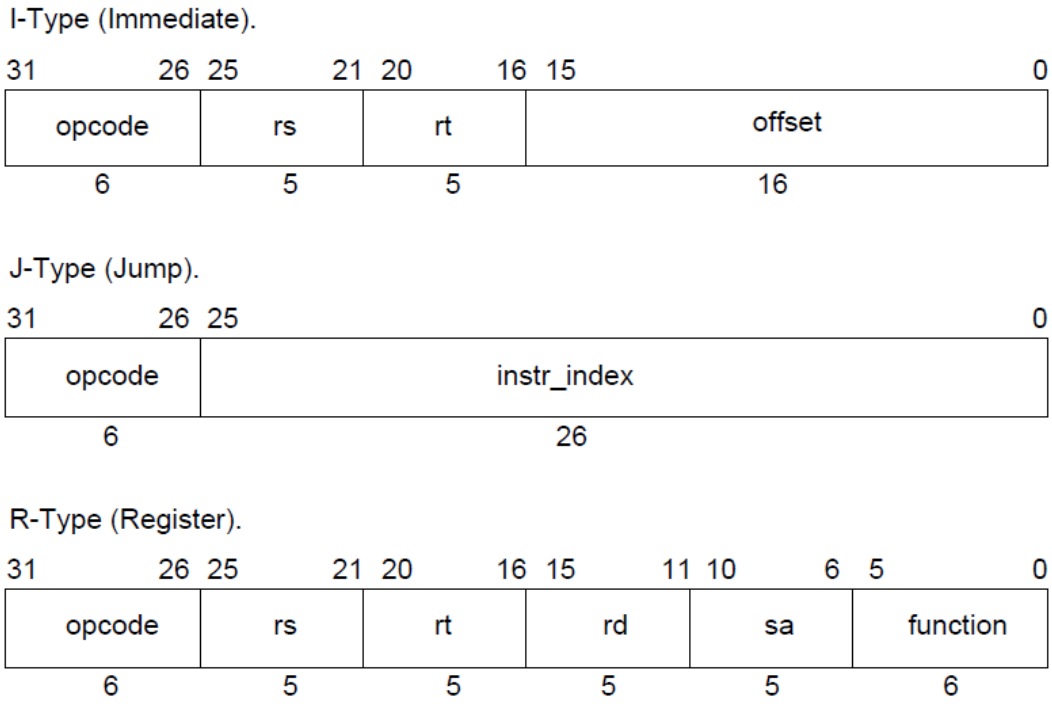
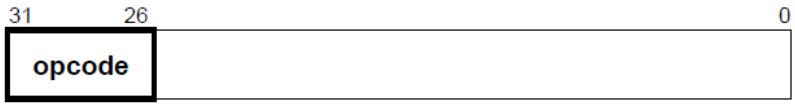


Figure 1: MIPS-I instruction formats

In all J-TYPE and I-TYPE instructions (except branch instructions comparing one register against zero), the opcode field is used to specify instruction type. This class of instructions is called 'instructions encoded by opcode field'. All possible values for opcode field under this class are shown in figure 2, along with valid mnemonics.

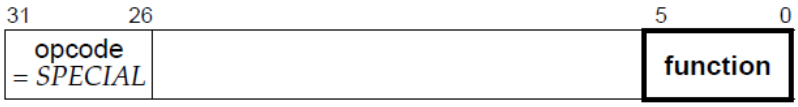


opcode bits 28..26 Instructions encoded by **opcode** field.

bits 31..29		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	000	*SPECIAL* δ	*REGIMM* δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	010	*COP0* δ,π	*COP1* δ,π	*COP2* δ,π	*COP3* δ,π,κ	*	*	*	*
3	011	*	*	*	*	*	*	*	*
4	100	LB	LH	LWL	LW	LBU	LHU	LWR	*
5	101	SB	SH	SWL	SW	*	*	SWR	*
6	110	*	LWC1 π	LWC2 π	LWC3 π,κ	*	*	*	*
7	111	*	SWC1 π	SWC2 π	SWC3 π,κ	*	*	*	*

Figure 2: Instructions encoded by opcode field

All R-TYPE instructions (including control instructions) are encoded using funct field, where opcode field is constant. For control instructions, the opcode field is equal to *COP*0 constant, while opcode field of remaining instruction is equal to *SPECIAL* constant. R-TYPE instructions with opcode equal to *SPECIAL* constant form another third class, called 'instructions encoded by function field when opcode field = *SPECIAL*'. Figure 3 shows members of this class.

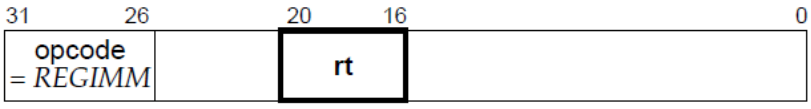


function bits 2..0 — Instructions encoded by **function** field when opcode field = SPECIAL.

bits 5..3		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	000	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1	001	JR	JALR	*	*	SYSCALL	BREAK	*	*
2	010	MFHI	MTHI	MFLO	MTLO	*	*	*	*
3	011	MULT	MULTU	DIV	DIVU	*	*	*	*
4	100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	101	*	*	SLT	SLTU	*	*	*	*
6	110	*	*	*	*	*	*	*	*
7	111	*	*	*	*	*	*	*	*

Figure 3: Instructions encoded by function field when opcode field = *SPECIAL*

Finally, branch instructions comparing one register against zero are encoded using rt field, while opcode field hold a constant value equal to *REGIMM* constant (register-immediate). This class, shown in figure 4, is called 'instructions encoded by the rt field when opcode field = *REGIMM*'.



rt bits 18..16 — Instructions encoded by the **rt** field when opcode field = REGIMM.

bits 20..19		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	00	BLTZ	BGEZ	†	†	†	†	†	†
1	01	†	†	†	†	†	†	†	†
2	10	BLTZAL	BGEZAL	†	†	†	†	†	†
3	11	†	†	†	†	†	†	†	†

Figure 4: Instructions encoded by rt field when opcode = *REGIMM*

1.1.7 Addressing Modes

Figure 5 summarizes all possible addressing modes along with instructions that support them:

Addressing Mode	Corresponding Instructions
Register, [Base Register]+Immediate Offset	Load/Store Instructions
Register, Register, Immediate	ALU Instructions with Immediate Operand
Register, Register, Register	ALU Instructions with Three Operands
	SLLV, SRLV, and SRA
Register, Register, Shift Amount	SLL, SRL, and SRA
Register, Register	MULT, MULTU, DIV, and DIVU
Register	MFHI, MTHI, MFLO, and MTLO
Absolute Address Within 256MB Segment	Jump Instructions Jumping Within 256MB Region
Register Absolute	Jump Instructions to Absolute Address
Register, Register, PC-Relative Immediate	Branch Instructions Comparing 2 Registers
Register, PC-Relative Immediate	Branch Instructions Comparing Against Zero
Immediate Code	Exception Instructions
Register, Control Register	MFC0 and MTC0
Implicit	TLBR, TLBWI, and RFE

Figure 5: All possible addressing modes

We note that every instruction has no more than one single addressing mode, thus MIPS doesn't need an addressing mode field in instruction word.

1.1.8 Programmer-Visible Pipeline Effects

Several effects related to the behaviour of the pipeline should be stated clearly so the programmer can be aware of the exact performance parameters of the running program:

- **Delayed branching**: Since the effective address is not available immediately after fetching a jump/branch instruction, a delay slot should be inserted after the jump/branch instruction. The delay instruction is always fetched and executed after the jump/branch instruction. The effective address becomes ready after fetching the delay instruction and next instruction would be the instruction at the new PC.
- **Load data not available**: If a memory load instruction is followed by an ALU instruction that requires the data acquired by the load instruction (addresses the same register), a data hazard will occur since ALU must wait until MEM unit is done fetching data from memory. This shall delay the execution of the ALU instruction by one delay slot (stalls the pipeline for one slot).
- **Consequent control instructions**: A control instruction may stall if there is another control instruction in the pipeline, to prevent data hazards and simultaneous access to TLB and stuff.
- **Branch instruction operands not ready**: If a branch instruction depends on result of the previous instruction (or second previous instruction), the pipeline will stall until data is ready.
- **Multiply/divide instructions**: These four special instructions take more than one CPU clock cycle to complete, therefore they might stall the pipeline.
- **Cache misses**: The pipeline shall stall if the instruction to be fetched or data specified by load instruction is not in cache. Furthermore, store instructions would always stall the pipeline since the cache system is write-through. This means that an instruction will take more than one CPU clock cycle if a cache miss happens.

- **Unaligned memory access**: This class of instructions shall take more than one CPU clock cycles because atomic access to unaligned word is not possible.

Other than the cases specified above, the pipeline will have no hazards/stalls and every instruction will take no more than one clock cycle delay.

1.2 Registers

1.3 Exception Handling

1.4 Paging

1.5 Cache Management

1.6 VHDL Implementation: Introduction

1.7 VHDL Implementation: The Pipeline

1.8 VHDL Implementation: Exceptions

1.9 VHDL Implementation: Cache and TLB