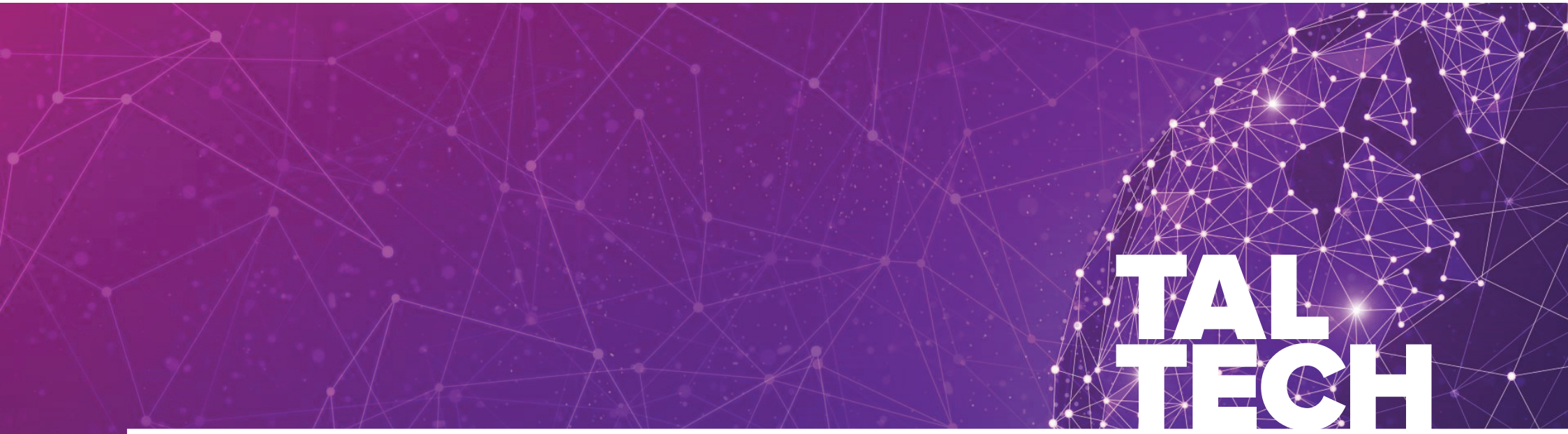


MICROPROCESSOR SYSTEMS (IAS0430)

Department of Computer Systems
Tallinn University of Technology

24.09.2021

MIPS

- **What is MIPS**
  - Stands for **M**icroprocessor without **I**nterlocked **P**ipelined **S**tages
  - MIPS is a perfectly **RISC** design
    - Data and buses (data bus and address bus) size of 32 bits.
    - Supports four integer data types:
      - 8bit,
      - 16bit **half-words**,
      - 32bit **words**, and
      - 64 bit **double-words**.
    - Supports two floating point data types:
      - 32bit **single precision**, and
      - 64 bit **double precision**
    - Has 32 bit **general purpose registers**
      - How many bits needed to address those registers?

MIPS

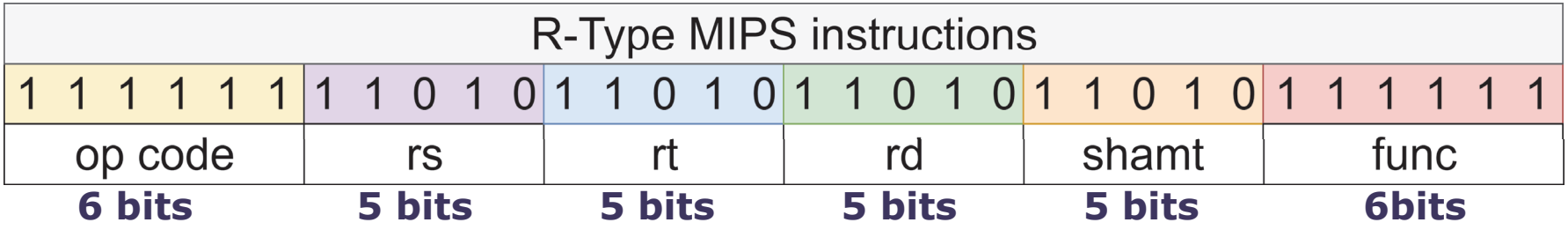
- **What is MIPS**
  - Stands for **M**icroprocessor without **I**nterlocked **P**ipelined **S**tages
  - MIPS is a perfectly **RISC** design
    - Data and buses (data bus and address bus) size of 32 bits.
    - Supports four integer data types:
      - 8bit,
      - 16bit **half-words**,
      - 32bit **words**, and
      - 64 bit **double-words**.
    - Supports two floating point data types:
      - 32bit **single precision**, and
      - 64 bit **double precision**
    - Has 32 bit **general purpose registers**
      - How many bits needed to address those registers?
        - We need 5 bits to address 32 registers

MIPS

- Data is only processed when they are in the registers.
  - MIPS is a **load/store** driven design.
  - All data must be **loaded to the registers before it is processed.**
- **MIPS provide the following types of operations:**
  - **LOAD** registers
  - **STORE** from registers
  - Integer arithmetic: **add**, **subtract**, multiply, divide with remainder.
  - Floating point arithmetic: add, subtract, multiply, divide
  - logical operations: AND, OR, NOR, exclusive OR (XOR).
  - comparison operations: **==**, !=, <, >, <=, >=
  - Branches and **jump operations**
  - shift operations: shift left, shift right

MIPS FORMATS

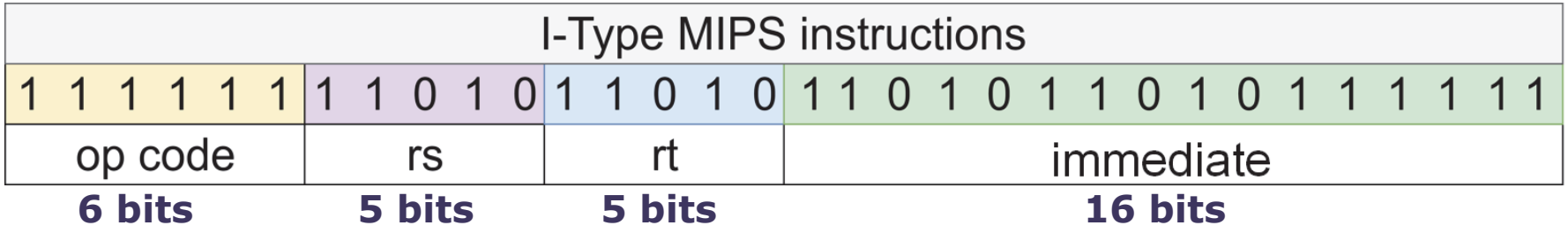
- MIPS have three notable formats:
  - R-type instructions:
    - R stands for register
    - These instructions specialize in operating **arithmetic** and **logical operation** on values stored in the registers.



  - **Examples include:**
    - **Addition :** add $1, $2, $3
    - **Subtraction:** sub $6, $3, $5
    - **Multiplication:** mult $12, $22
    - **Division:** div $2, $33
    - **Indirect Addressing:** Jr $14

MIPS FORMATS

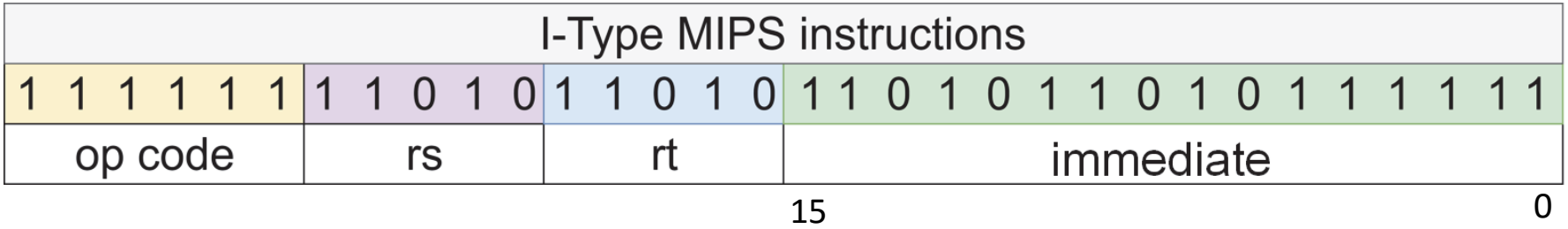
- MIPS have three notable formats:
  - I-type instructions:
    - I stands for Immediate
    - These instructions specialize in operation such as **LOAD** and **STORE** and other operations performed on immediate values



  - **Examples include:**
    - **Addition on immediate:** addi $5, $22, 83
    - **AND immediate:** andi $23, $2, 46
    - **OR immediate:** ori $5, $16, 152

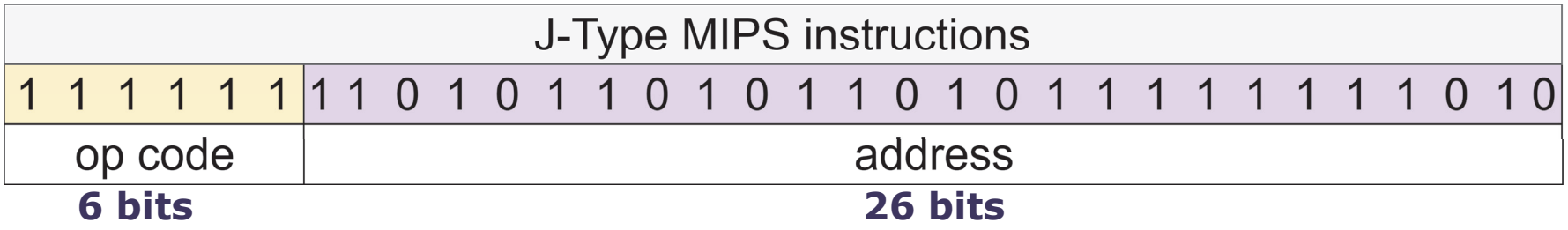
MIPS FORMATS

- MIPS have three notable formats:
  - I-type instructions can also perform **I-type branching**
    - These instruction allows us to change the value in the PC register
      - This helps reducing the need for condition-based instructions
  - **Example: bne $13, $26, 3**
  - The above instruction performs the following:
    - If value in register number 13 is not equal to value in register 26
      - PC = PC+4+BranchAddr;
      - BranchAddr = { 14{immediate[15]}, immediate, 2'b0 };
        - Source: MIPS Green Sheet; Syntax: SystemVerilog
      - If PC was at instruction 256, it will branch to 272 (256+4+3*4).



MIPS FORMATS

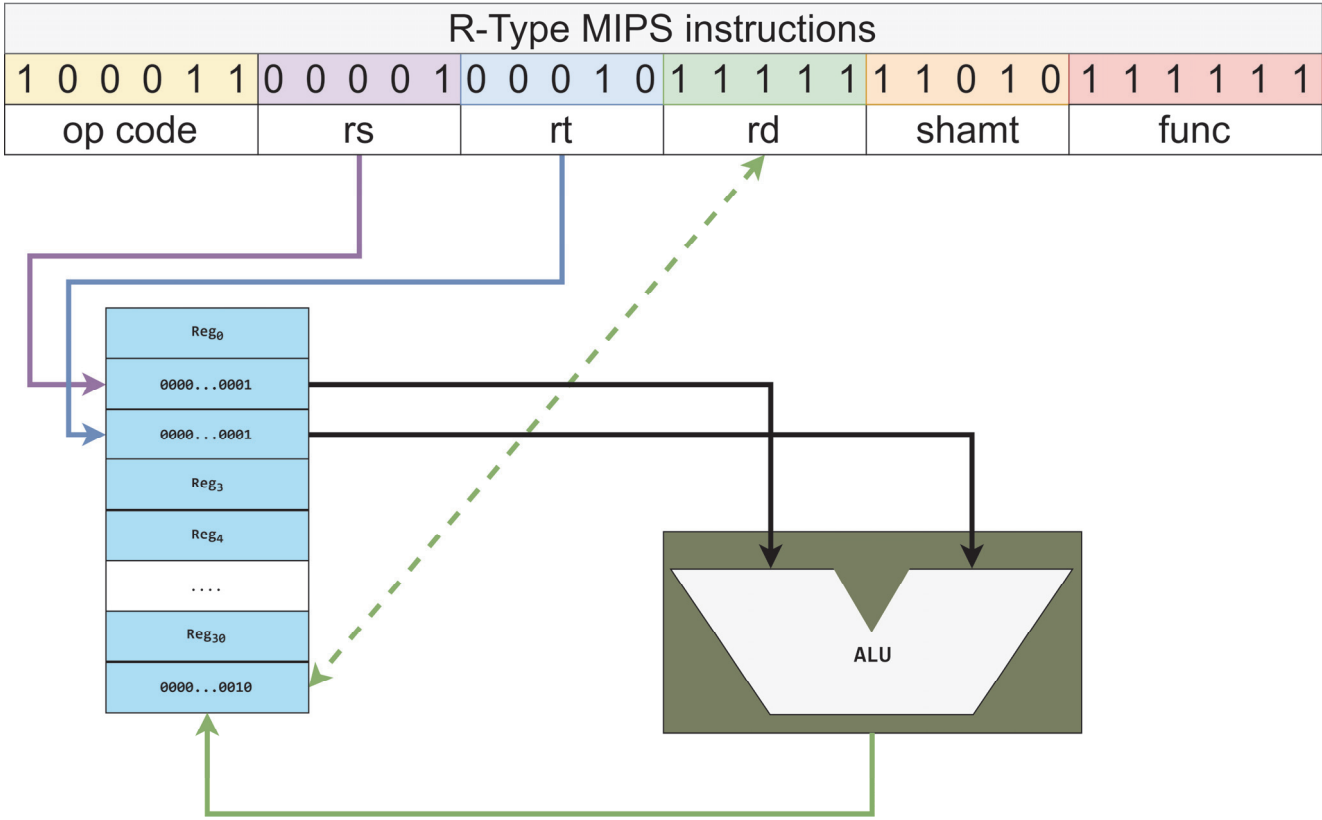
- MIPS have three notable formats:
  - J-type instructions:
    - J stands for Jump
    - These instructions specialize in operation such as jump instructions and function calls when a new program is called to execute.



  - **Examples include:**
    - **Jump to address:** J 11010110101101011111111010
      - PC = JumpAddr; JumpAddr = { (PC+4)[31:28], address, 2'b0 }
      - PC = xxxx_11010110101101011111111010_00
        - Source: MIPS Green Sheet; Syntax: SystemVerilog
    - **Jump to address in register:** Jr $22 [NB! R-type instruction]

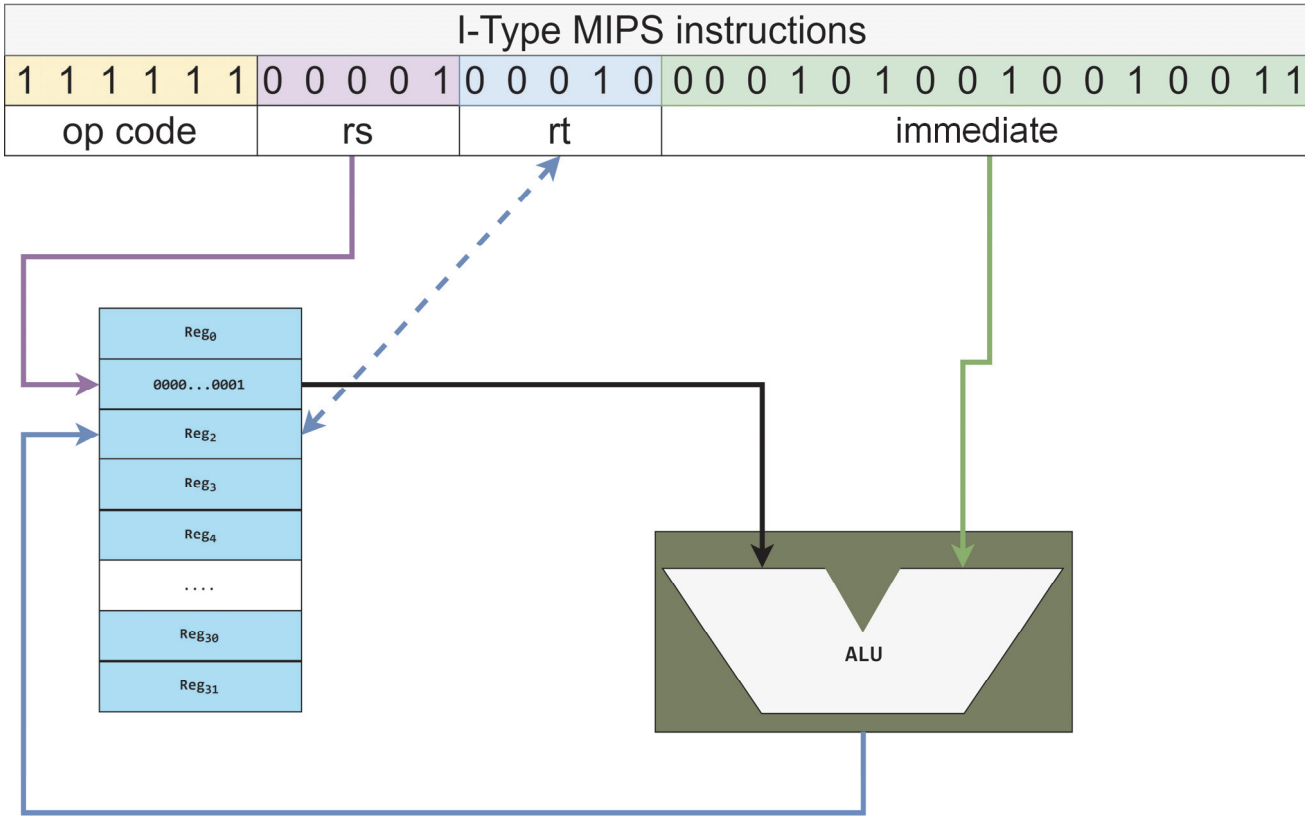
MIPS – ADDRESSING MODES: REGISTER ADDRESSING

- How does each format work?
  - **R-type instructions take advantage of the several registers that come with MIPS**
  - For example: **add $1, $2, $31**
  - Values from source register (rs) and target register (rt) are loaded to the ALU.
  - In this case, value in reg1 and reg2 are loaded to the ALU
  - The ALU performs the addition
  - The ALU saves the output of the addition to the destination register (rd)



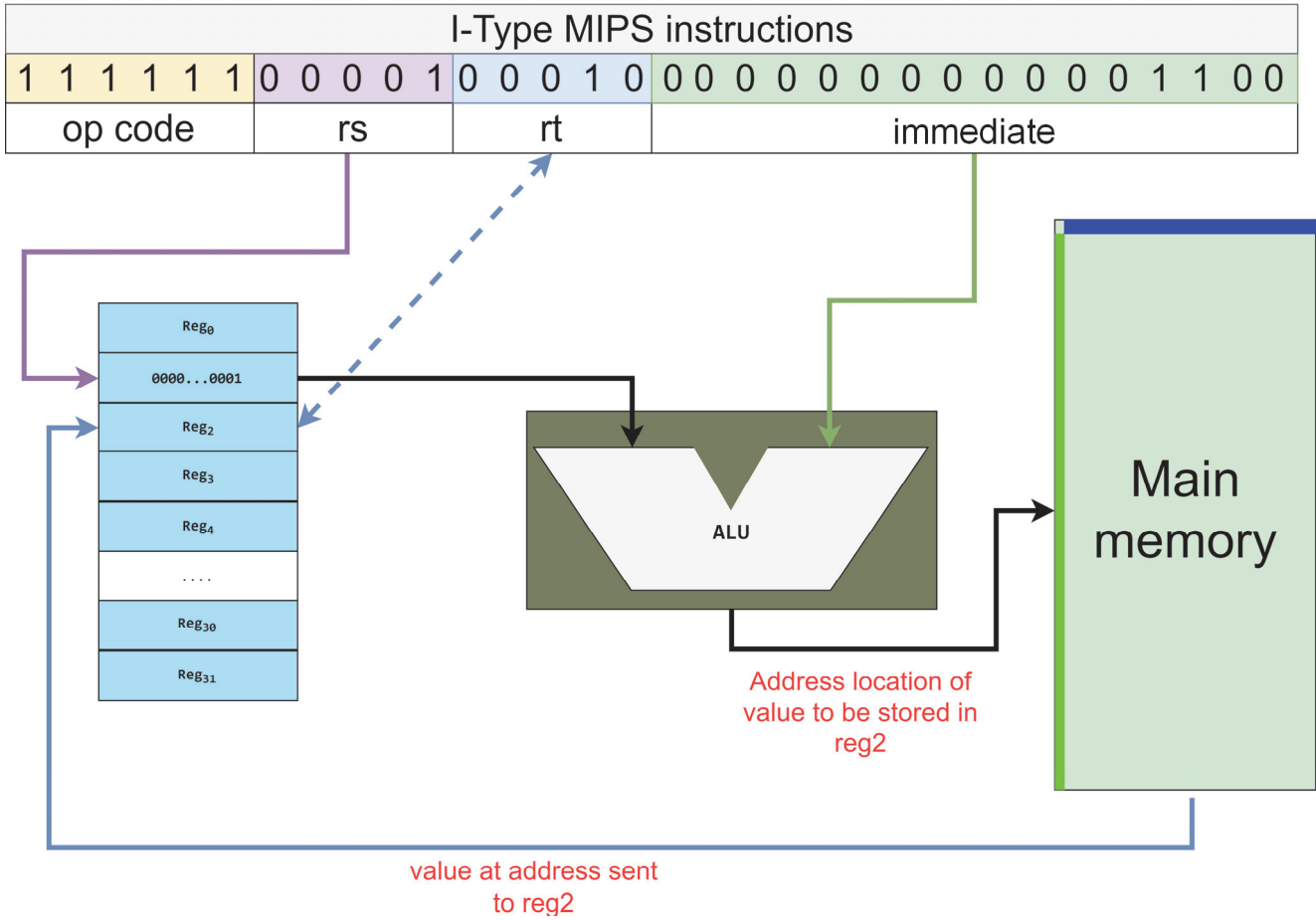
MIPS – ADDRESSING MODES: IMMEDIATE ADDRESSING

- How does each format work?
  - **I-type instructions can be used to update values in the register by performing immediate value addition.** Similar to previous addition, but …
  - For example: addi $1, $2, 5267
  - Values from source register (rs) and the immediate bits are loaded to the ALU
  - In this case, value in reg1 and immediate value 5267 are loaded to the ALU
  - The ALU performs the addition
  - The ALU saves the output of the addition to the target register (rt)



MIPS – ADDRESSING MODES: BASE ADDRESSING

- How does each format work?
  - **I-type instructions can help us figure out where values in memory are and load them to registers**.
  - We can use I-type instructions to calculate memory addresses.
  - For example: lw $2,12($1)
  - This means that the value we want to store in register 2 is located at memory location equal to value stored in register 1 + 12
  - Value of reg1 and immediate are loaded to ALU.
  - ALU signals memory to send value stored in location equal to ALU output.
  - Value is stored in reg2.



MIPS – ADDRESSING MODES

- **Base addressing** is very useful in making programs more flexible.
- **Instead of having a fixed memory location in an instruction**, base addressing allow us to **store the program in any place in memory.**
- Other addressing modes include:
  - **Indirect Addressing** (also know as register direct addressing)
    - Where the memory location is stored inside a register: Jr $22
  - **Direct Addressing**
    - The immediate acts as a memory location
    - But, a 32-bit address can not be included in a 32-bit instruction
  - **Pseudo direct Addressing**
    - Also called JUMPS
    - Where a 26-bit is used as an offset to address memory region within the current 256MB of address space.
    - Example is regular jump or **J Label**

MIPS – MORE IS MORE

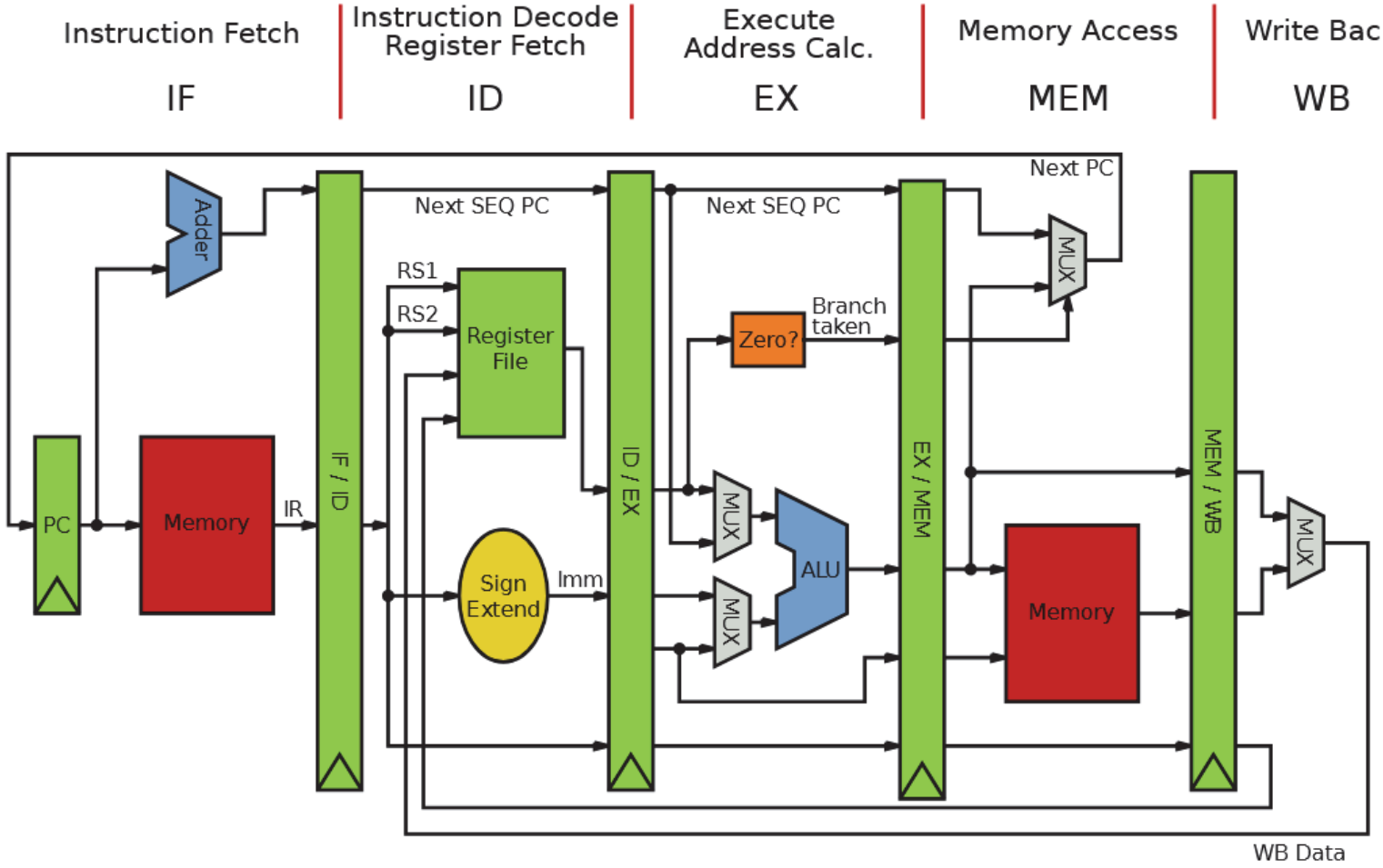
- Important to keep in mind:
  - **Addressing modes are not instruction types**
  - Instruction types indicate the format and how an instruction looks like.
  - Address modes include how an instruction can access memory (reg file or main memory)
    - addi is an **I-type instruction** that uses both **immediate** and **register** address modes
    - lw is another **I-type** that uses both **base addressing** and **register** address
    - Jr is a **J-type** that uses **indirect addressing**
    - Etc..

MIPS – MORE IS MORE

- Summary of address modes
  - **REGISTER**: Operands are a content of a **source, target, and destination register** labels as $0-$31
  - **IMMEDIATE**: the operand is an **immediate value** stored in the instruction itself.
  - **PC-RELATIVE**: a memory location is specified as **an offset in accordance to the incremented PC.**
  - **BASE**: the memory location or data is **calculated by an adding an immediate value to another value** stored in a register.
  - **REGISTER-DIRECT**: The location is stored in a specified **register**.
  - **PSEUDODIRECT**: The **memory location can be found in the instruction itself**, but may need some further operation to be calculated.

MIPS - PIPELINE

- MIPS Green Sheet https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_Green_Sheet.pdf



Source
Wikipedia

RISC-V – OPEN STANDARD INSTRUCTION SET ARCHITECTURE

- UC Berkeley – started in 2010 – 32, 64 & 128-bit versions

32-bit RISC-V instruction formats

Format	Bit																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

- *opcode* (7 bits): Partially specifies which of the 6 types of *instruction formats*.
- *funct7*, and *funct3* (10 bits): These two fields, further than the *opcode* field, specify the operation to be performed.
- *rs1*, *rs2*, or *rd* (5 bits): Specifies, by index, the register, resp., containing the first operand (i.e., source register), second operand, and destination register to which the computation result will be directed.

Source
Wikipedia