Sta per **Microprocessor** without **Interlock Pipeline Stages** e fa parte della famiglia dei processori **RISC.**
Ha un semplice *instruction set* basato su **load/store.**
E' stato pensato per avere una **pipeline** efficiente, grazie alle istruzioni a lunghezza **fissa** e al fatto che consuma poca energia.
Lavora con **word** da **32 bit**.

# Registri

Tutti i registri sono a 64 bit.
Ha dei **registri speciali** come **PC**, **HI** e **LO** e dei registri dedicati alla **FPU.**
Poi ha 32 registri **general purpose** per interi dove il primo registro, chiamato **R0**, è sempre settato a 0.
Inoltre ha altri 32 registri **general purpose** per *floating point*.
I floating point adottano lo standard **IEEE754**.

# Addressing Modes

## Immediate

Usa **16 bit** per **immediate field**, ovvero 16 bit per una *costante*.
*DADDUI R1,R0,#32* equivale a *R1 = 32*

## Displacement

*LD R1,30(R2)* equivale a *R1* $\leftarrow$ *MEM[R2+30]*

*LD R1,64(R0)* equivale a *R1* $\leftarrow$ *MEM[64]* OVVERO **absolute addressing**

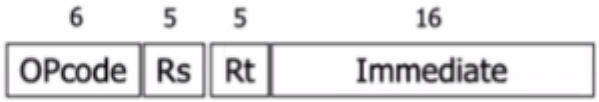
# Instruction Format

Ogni istruzione è una **word** di 32 bit **aligned.**
I primi **6 bit** indicano l'**opcode.**
Ci sono 3 tipi di istruzione:

- **Immediate**
- **Register**
- **Jump**

## Immediate

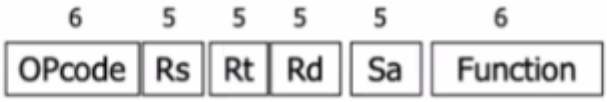
- I – type instruction




| Field | Description |
|---|---|
| *opcode* | 6-bit primary operation code |
| *Rs* | 5-bit specifier for the source register |
| *Rt* | 5-bit specifier for the target (source/destination) register |
| *Immediate* | 16-bit signed *immediate* used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement |

## Register

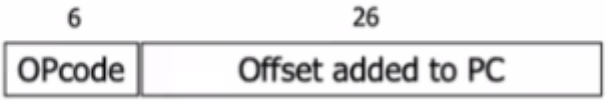
- R – type instruction




| Field | Description |
|---|---|
| *opcode* | 6-bit primary operation code |
| *Rd* | 5-bit specifier for the destination register |
| *Rs* | 5-bit specifier for the source register |
| *Rt* | 5-bit specifier for the target (source/destination) register |
| *Sa* | 5-bit shift amount |
| *Function* | 6-bit function field used to specify functions within the primary opcode SPECIAL |

## Jump

- J – type instruction




| Field | Description |
|---|---|
| *opcode* | 6-bit primary operation code |
| *Offset* | 26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address |

*Indica di quante word devo spostarmi. Dal momento che ogni istruzione è lunga 32 bit, ha più*

*senso specificare l'offset in termine di numero di word.*
*I due bit più bassi quindi saranno sempre a 0(basti vedere 4,8,12,16 etc in binario).*

# Load And Store functions

- **LD - Load double world:** *LD R1,28(R8)*
- **LB - Load Byte:** Come LD, tuttavia il segno viene esteso nei restanti bit( i registri sono da 64 bit, il byte è da 8 bit). Quindi si converte il valore in complemento a 2.
- **LBU - Load Byte unsigned:** Nei restanti 56 bit si mette 0
- **L.S - Load FP Single:** Carica 32 bit floating point nei registri floating point. 32 zero vengono messi a destra.
- **L.D - Load FP Double:** Carica 64 bit floating point nei registri floating point
- **SD - Store Double:** Salvo 64 bit in una locazione di memoria
- **SH - Store Half Word:**** Salvo 32 bit in una locazione di memoria
- **SB - Store byte:** Salvo un byte in una locazione di memoria
- **S.S - Store FP Single:** Salvo 32 bit floating precision
- **S.D - Store FP Double:** Salvo 64 bit floating precision

# ALU operations

Tutte le operazioni vengono eseguite sui registri.

### R0

Per caricare una costante, posso usare R0 come secondo operando, visto che R0 vale sempre 0.

## ADD

- **DADDU - Double Add Unsigned:** Somma di due numeri da 64 bit unsigned
- **DADDUI - Double Add Immediate:** Somma di due numeri da 64 bit unsigned di cui uno immediate
- **LUI - Load Upper Immediate:** Carico i 16 bit più significativi.
- **DSLL - Double Shift left logical:** Applica uno shift a sinistra *logico*
- **SLT - Set Less than:** se *operando2<operando3*, imposta *operando1* a **1**, altrimento a **0**.

# Jump and Branch

- **J - Jump:** Salto ad una etichetta
- **JAL - Jump and link:** Prima di saltare ad una etichetta, mi salvo l'indirizzo dell'istruzione successiva in R31(mi salvo quindi PC+4). Utile nelle procedures.

- **JALR - Jump and link register:** Come JAL, ma in PC mi salvo il valore di un registro invece del valore di una etichetta.
- **JR - Jump Register:** come **J** ma uso il valore di un registro invece di una etichetta.
- **BEQZ - Branch Equal Zero:** Se il primo operando è a 0, salto all'etichetta specificata. (utilizza 2 operandi)
- **BNE - Branch Not Equal:** Se il primo operando è diverso dal secondo, allora salto all'etichetta specificata.(utilizza 3 operandi)

## Miscellanous

- **MOVZ - Move if Zero:** Se *operando3=0*, allora *operando1=operando2*
- **NOP - No Operation:** Non fa nulla ed equivale a *SLL R0,R0,0*.

## Assembler Programs

Composto da una **Data Section** contenente i dati e una **Code Section** contenente il codice effettivo.

### Direttive

Comandi speciali dell'assembler che permettono di definire variabili costanti etc