# Nios II Instruction Set Summary

**Learning Goal:** Assembly language basics.

## 1 Introduction

This document summaries the subset of instruction implemented in your multi-cycle Nios II and describes the function of its 32 registers.

## 2 Registers

The following table lists the conventional function and names[1] of the 32 registers of the **Register File**.

| Register | Name | Function |
|---|---|---|
| `r0` | `zero` | 0x00000000 |
| `r1` | `at` | Assembler Temporary |
| `r2` | `v0` | Return Value |
| `r3` | `v1` | Return Value |
| `r4` | `a0` | Register Arguments |
| `r5` | `a1` | Register Arguments |
| `r6` | `a2` | Register Arguments |
| `r7` | `a3` | Register Arguments |
| `r8` | `t0` | Temporary Register |
| `r9` | `t1` | Temporary Register |
| `r10` | `t2` | Temporary Register |
| `r11` | `t3` | Temporary Register |
| `r12` | `t4` | Temporary Register |
| `r13` | `t5` | Temporary Register |
| `r14` | `t6` | Temporary Register |
| `r15` | `t7` | Temporary Register |

| Register | Name | Function |
|---|---|---|
| `r16` | `s0` | Saved Register |
| `r17` | `s1` | Saved Register |
| `r18` | `s2` | Saved Register |
| `r19` | `s3` | Saved Register |
| `r20` | `s4` | Saved Register |
| `r21` | `s5` | Saved Register |
| `r22` | `s6` | Saved Register |
| `r23` | `s7` | Saved Register |
| `r24` | `et` | Exception Temporary |
| `r25` | `bt` | Breakpoint Temporary |
| `r26` | `gp` | Global Pointer |
| `r27` | `sp` | Stack Pointer |
| `r28` | `fp` | Frame Pointer |
| `r29` | `ea` | Exception Return Address |
| `r30` | `ba` | Breakpoint Return Address |
| `r31` | `ra` | Return Address |

## 3 Instructions

The following table lists the instructions implemented by your Nios II processor. We write "$\texttt{rA}_s$" to consider the `rA` as signed, and "$\texttt{rA}_u$" when it is unsigned. We write "$\texttt{imm}_s$", when the immediate value is signed extend, and "$\texttt{imm}_u$", when the immediate value is unsigned (i.e., extended with zeros).

[1] To improve the readability of the code, we extend the official Nios II registers naming: we added names to the registers `r2` to `r23`, which were unnamed. These names are only supported by the **Nios2Sim** simulator.

| Category | Instruction | | Meaning |
|---|---|---|---|
| Arithmetic | **addi** | rB, rA, imm | $rB \leftarrow rA + imm_s$ |
| | **add** | rC, rA, rB | $rC \leftarrow rA + rB$ |
| | **sub** | rC, rA, rB | $rC \leftarrow rA - rB$ |
| Logical | **and** | rC, rA, rB | $rC \leftarrow rA$ *and* $rB$ |
| | **andi** | rB, rA, imm | $rB \leftarrow rA$ *and* $imm_u$ |
| | **or** | rC, rA, rB | $rC \leftarrow rA$ *or* $rB$ |
| | **ori** | rB, rA, imm | $rB \leftarrow rA$ *or* $imm_u$ |
| | **xor** | rC, rA, rB | $rC \leftarrow rA$ *xor* $rB$ |
| | **xori** | rB, rA, imm | $rB \leftarrow rA$ *xor* $imm_u$ |
| | **nor** | rC, rA, rB | $rC \leftarrow rA$ *nor* $rB$ |
| Comparator | **cmpgei** | rB, rA, imm | $rB \leftarrow (rA \geq imm_s)?\ 1 : 0$ |
| | **cmplti** | rB, rA, imm | $rB \leftarrow (rA < imm_s)?\ 1 : 0$ |
| | **cmpnei** | rB, rA, imm | $rB \leftarrow (rA \neq imm_s)?\ 1 : 0$ |
| | **cmpeqi** | rB, rA, imm | $rB \leftarrow (rA = imm_s)?\ 1 : 0$ |
| | **cmpgeui** | rB, rA, imm | $rB \leftarrow (rA_u \geq imm_u)?\ 1 : 0$ |
| | **cmpltui** | rB, rA, imm | $rB \leftarrow (rA_u < imm_u)?\ 1 : 0$ |
| | **cmpge** | rC, rA, rB | $rC \leftarrow (rA \geq rB)?\ 1 : 0$ |
| | **cmplt** | rC, rA, rB | $rC \leftarrow (rA < rB)?\ 1 : 0$ |
| | **cmpne** | rC, rA, rB | $rC \leftarrow (rA \neq rB)?\ 1 : 0$ |
| | **cmpeq** | rC, rA, rB | $rC \leftarrow (rA = rB)?\ 1 : 0$ |
| | **cmpgeu** | rC, rA, rB | $rC \leftarrow (rA_u \geq rB_u)?\ 1 : 0$ |
| | **cmpltu** | rC, rA, rB | $rC \leftarrow (rA_u < rB_u)?\ 1 : 0$ |
| Shift | **sll** | rC, rA, rB | $rC \leftarrow rA \ll rB_{4..0}$ |
| | **slli** | rC, rA, imm | $rC \leftarrow rA \ll imm_{4..0}$ |
| | **srl** | rC, rA, rB | $rC \leftarrow rA_u \gg rB_{4..0}$ |
| | **srli** | rC, rA, imm | $rC \leftarrow rA_u \gg imm_{4..0}$ |
| | **sra** | rC, rA, rB | $rC \leftarrow rA_s \gg rB_{4..0}$ |
| | **srai** | rC, rA, imm | $rC \leftarrow rA_s \gg imm_{4..0}$ |
| | **rol** | rC, rA, rB | $rC \leftarrow rA$ *rol* $rB_{4..0}$ |
| | **ror** | rC, rA, rB | $rC \leftarrow rA$ *ror* $rB_{4..0}$ |
| | **roli** | rC, rA, imm | $rC \leftarrow rA$ *rol* $imm_{4..0}$ |
| Memory | **ldw** | rB, imm (rA) | $rB \leftarrow$ MEM[$imm_s$+rA] |
| | **stw** | rB, imm (rA) | MEM[$imm_s$+rA] $\leftarrow rB$ |
| Branch | **br** | imm | goto PC+4+$imm_s$ |
| | **bge** | rA, rB, imm | if $(rA \geq rB)$ goto PC+4+$imm_s$ |
| | **blt** | rA, rB, imm | if $(rA < rB)$ goto PC+4+$imm_s$ |
| | **bne** | rA, rB, imm | if $(rA \neq rB)$ goto PC+4+$imm_s$ |
| | **beq** | rA, rB, imm | if $(rA = rB)$ goto PC+4+$imm_s$ |
| | **bgeu** | rA, rB, imm | if $(rA_u \geq rB_u)$ goto PC+4+$imm_s$ |
| | **bltu** | rA, rB, imm | if $(rA_u < rB_u)$ goto PC+4+$imm_s$ |
| Jump | **call** | imm | goto $imm \ll 2$; $ra \leftarrow$ PC+4 |
| | **callr** | rA | goto rA; $ra \leftarrow$ PC+4 |
| | **ret** | | goto ra |
| | **jmp** | rA | goto rA |
| | **jmpi** | imm | goto $imm \ll 2$ |
| Misc | **break** | | stops the processor[2] |

[2]This is not the official function of the instruction.