



The format for the 3 types of instructions we care about is shown in Figure 2. The notation for the parts of the instruction in the figure are below (Shamt is not used by us):

- OP-CODE – operation
- RS – First register source operand
- RT – Second register source operand
- RD – Register destination operand
- Shamt – Shift amount
- Funct – Specifies variant on operation given in opcode

As mentioned, the first 6 bits (the OP-CODE) drive the controls of the rest for the computer. To do this the instruction is used to determine the state of several smaller 'micro-instructions' that can be put together to form a 'control word'. We'll use Figure 1's naming conventions for each signal for now. The OP-CODES we are interested in are below:

OP-CODE	Mode
0	Register
2	Jump
7	Immediate (Branch Greater Than [BGT])
8	Immediate (Add [ADD])
10	Immediate (Set Less Than [SLT])
12	Immediate (And [AND])
13	Immediate (Or [OR])
14	Immediate (Xor [XOR])
35	Immediate (Load Word [LW])
43	Immediate (Store Word [SW])

**Table 1: OP-CODES**

For register mode (OP-Code = 0) there are additionally several ALU operations that can be specified by the Funct field:

Funct	ALU Operation
4	Shift Left Logical [SLLV]
8	Jump Register [JR]
32	Add [ADD]
34	Subtract [SUB]
36	And [AND]
37	Or [OR]
38	Xor [XOR]
42	Set Less Than [SLT]

**Table 2: ALU Functions**

Below are the designations for each of the registers in the reg file. For the register operations, we will primarily use the temporary registers (T or TR) (the '\$' sign is a convention for the file register).

Register Number	Conventional Name	Usage
\$0	\$zero	Hard-wired to 0
\$1	\$at	Reserved for pseudo-instructions
\$2 - \$3	\$v0, \$v1	Return values from functions
\$4 - \$7	\$a0 - \$a3	Arguments to functions - not preserved by subprograms
\$8 - \$15	\$t0 - \$t7	Temporary data, not preserved by subprograms
\$16 - \$23	\$s0 - \$s7	Saved registers, preserved by subprograms
\$24 - \$25	\$t8 - \$t9	More temporary registers, not preserved by subprograms
\$26 - \$27	\$k0 - \$k1	Reserved for kernel. Do not use.
\$28	\$gp	Global Area Pointer (base of global data segment)
\$29	\$sp	Stack Pointer
\$30	\$fp	Frame Pointer
\$31	\$ra	Return Address
\$f0 - \$f3	-	Floating point return values
\$f4 - \$f10	-	Temporary registers, not preserved by subprograms
\$f12 - \$f14	-	First two arguments to subprograms, not preserved by subprograms
\$f16 - \$f18	-	More temporary registers, not preserved by subprograms
\$f20 - \$f31	-	Saved registers, preserved by subprograms

**Table 3: File Register Designations**

Below is the logic for most of the control signals shown in Figure 1:

OP-CODE	0*	2	7**	8	10	12	13	14	35***	43****
<b>RegDst</b>	1	X	X	0	0	0	0	0	0	X
<b>Jump</b>	1	0	0	1	1	1	1	1	1	1
<b>Jump Reg</b>	X	0	X	X	X	X	X	X	X	X
<b>Branch</b>	X	0	1	X	X	X	X	X	X	X
<b>MemRead</b>	0	0	0	0	0	0	0	0	1	0
<b>MemtoReg</b>	0	X	X	0	0	0	0	0	1	0
<b>ALUOp</b>	6x1	X	SUB	ADD	SLT	AND	OR	XOR	ADD	ADD
<b>MemWrite</b>	0	0	0	0	0	0	0	0	0	1
<b>ALUSrc</b>	0	X	0	1	1	1	1	1	1	1
<b>RegWrite</b>	1	0	0	1	1	1	1	1	1	0

**Table 4: Control Signal Logic**

For Immediate Arithmetic Operations, RT is the destination register for the result: so  $RT = RS + \text{Immed. Value}$

\* ALUOp for Register-type is a mask for the ALU funct:  $\text{ALU control} = \text{ALUOp} \& \text{funct}$  (for ALUOp = 6'b111111)

\*\* Uses the Negative Flag to determine branch: Branches if  $RT > RS$

\*\*\* Writes to RT from memory address at  $RS + \text{Immed. Value}$ :  $RT = \text{mem}@(\text{RS} + \text{Immed. Value})$

\*\*\*\* Writes value in RT to memory address at  $RS + \text{Immed. Value}$ :  $\text{mem}@(\text{RS} + \text{Immed. Value}) = RT$