**North South University**

Department of Electrical & Computer Engineering

**PPROJECT MILESTONE**

Course Name: **CSE332- Computer Organization and Architecture**

# ISA DESIGN

| | |
|---|---|
| ASIF BHUIYAN | Score |
| | |

## Question 1: How many operands?

### Ans to the Question No-1

Ans: up to 2(two).

## Question 2: Types of operand? (Register based?? Memory based? Mixed?)

### Ans to the Question No-2

Ans: Our design requires three types of operands: register based, memory based, and immediate.

**Register-based operand:**

These operands are located in registers within the hardware. They can be referred to using $t0, $t1,… $t9 for temporary values, and $s0, $s1,… $s7 for saved values.
An instruction that requires register-based operand: **ADD $s2,$s0,$s1**. Here, the sum of the values saved in registers $s0 and $s1 is stored in register $s2. These are all register-based operands.

**Memory based operand**:

These operands are stored in the memory. Since memory is array-like and byte-addressed, these operands are referred to using an offset and the register containing its base address. For example, for locating A[2], where the base address of A is in register $s0, we use 8($s0).
An instruction that requires a memory-based operand**: lw $t0, 8($s0)**. This loads the value from the memory-based operand 8($s0) and stores it into register $t0.

**Immediate operand:**
These are the constant data specified in an instruction. They are referred to by directly using the constant data.
An instruction that requires immediate operand: **ADDi $s1, $s1, 4**. Here, 4 is the immediate operand and it is added to register-based operand $s1 and stored in $s1.

The examples of instructions stated above, are all included in our design, along with many more, which is why we require these three types of operands.

**Question 3: How many operations? Why?**

## Ans to the Question No-3

Ans: 5 operations.

1. Arithmetic
2. Logical
3. Data transfer
4. Conditional branch
5. Unconditional jump

Because we want to execute 3 kinds of operations

- Simple arithmetic & logic operations
- Programs that require checking operations
- Operations with loops.

The arithmetic and logic instructions perform operations on data and demonstrate the processor's computational capabilities. Such instructions are also known as data manipulation instructions.

Data transfer instructions transfer data between the various storage locations in the computer system, such as registers, memory, and I/O. Because both instructions and data are stored in memory, the processor must read both instructions and data from memory. The results of processing must be saved in memory.

The branch instruction adds a new value to the program counter, after which the processor fetches and executes the instruction at the new address. The conditional or unconditional branch instruction can be used. An unconditional branch instruction jumps to the specified address regardless of the condition. A conditional branch instruction creates a branch only when a certain condition is met. If the condition is not met, the PC is normally incremented and the next instruction in sequential address order is fetched and executed.

**Question 4: Types of operations? (Arithmetic,logical,branch type?? How many from each category? Draw a table with list of instructions, instruction type, their opcode, functionality (if any)**

## Ans to the Question No-4

1. **Arithmetic operation**

   ADD: Rd = Rs + Rt
   Operands A and B stored in register locations Rs and Rt are added and written to the destination register specified by Rd.

   SUB: Rd = Rs - Rt
   Operand B (Rt) is subtracted from Operand A (Rs) and written to Rd

2. **Logical operation:**

   AND: Rd = Rs & Rt
   Operand A (Rs) is bitwise ANDed with Operand B (Rt) and written into Rd

   OR: Rd = Rs | Rt
   Operand A (Rs) is bitwise ORed with Operand B (Rt) and written into Rd.

3. **Data transfer**

   LW: Rd = Mem[Rs, offset]
   Operand A (Rs) is loaded into Rd

   SW: Mem[Rs, offset] = Rd
   Operand A stored in Rd is transferred to Rs

4. **Conditional branch**

   Beq Rs, Rt, L
   If Operand A (Rs) == Operand B (Rt) then branch to instruction labeled L1

   Bne Rs, Rt, L1
   If Operand A (Rs) != Operand B (Rt) then branch to instruction labeled L1

5. **Unconditional jump**

   J L1 Instruction jumps to instruction labeled L1

| Instruction | Type | Format | OP code | Example | Meaning |
|---|---|---|---|---|---|
| ADD | Arithmetic | R | 0000 | ADD $r1 $r2 | $r1 = $r1+$r2 |
| SUB | Arithmetic | R | 0001 | SUB $r1 $r2 | $r1 = $r1-$r2 |
| ADDi | Arithmetic | I | 0010 | ADDi $r1 3 | $r1 = $r1+3 |
| Not | Logical | R | 1111 | Not $r1 $r2 | $r1 = $r2’ |
| AND | Logical | R | 0011 | AND $r1 $r2 | $r1 = $r1 ∩ $r2 |
| OR | Logical | R | 0100 | OR $r1 $r2 | $r1 = $r1 U $r2 |
| Sll | Logical | I | 0101 | Sll $r1 2 | $r1 = $r1<<2 |
| Srl | Logical | I | 0110 | Srl $r1 2 | $r1 = $r1>>2 |
| | | | | | |
| LW | Data transfer | I | 0111 | LW 5($r1) | $s0 = Memory [$r1+5] |
| SW | Data transfer | I | 1000 | SW 5($r1) | Memory [$r1+offset] = $s1 |
| IN | Data transfer | I | 1001 | IN $r1 0 | Takes input from $IN and pass the value to $r1 |
| OUT | Data transfer | I | 1010 | OUT $r1 0 | Load $OUT value to $r1 |
| | | | | | |
| Beq | Conditional | I | 1011 | Beq $r1,L1 | If ($r1==$s2) then jump to L1 |
| Slt | Conditional | R | 1100 | Slt $r1,$r2 | If ($r1<$r2) then $t0=1 else $t0=0 |
| Slti | Conditional | I | 1101 | Slti $r1,10 | If ($r1<10) then $t1=1 else $t1=0 |
| J | Unconditional | J | 1110 | J 10 | Jump to 10 |

**Question 5: How many formats would you choose? Draw the formats along with field name and number of bits in each field.**

## Ans to the Question No-5

I will choose 3 formats. There are 15 MIPS instructions and all the MIPS instructions are based on 3 formats. Basically, all instructions in MIPS Architecture are 32 Bit long. But here we’ll draw the 12 Bit format. I am enclosing the all 3 formats bellow:

**R Type Format:**

The first format is R Type format. It is also called as Register Type format. In this format there are 6 fields. Two operand and destination are specified by location of register file.

| 4 Bit | 4 Bit | 4 Bit |
|---|---|---|
| OP | rs | rt |

**I Type format:**

I Type format is the second format. Here i means immediate. It can be load ward/store ward operation or immediate ALU operation.

| 4 Bit | 4 Bit | 4 Bit |
|---|---|---|
| OP | rs | immediate |

**J Type Format:**

The last format is J Type format. It means Jump type format. J Type instruction contains only two fields. One is the OP and another is target.

| 4 Bit | 8 Bit |
|---|---|
| OP | target |

**Question 6: List of registers? Draw a register table.**

**Ans to the Question No-6**

- Memory address register
- Accumulator
- Memory data register
- Program counter register
- Index register

**Register Description**

| Register no | Register name | Binary Code | Operation | Usage |
|---|---|---|---|---|
| $0 | $zero | 0000 | Zero | Def |
| $8 | $t0 | 0001 | Temp | Def |
| $9 | $t1 | 0010 | Temp | Def |
| $10 | $t2 | 0011 | Temp | Cust |
| $11 | $t3 | 0100 | Temp | Cust |
| $12 | $t4 | 0101 | Temp | Cust |
| $16 | $s0 | 0110 | save | Def |
| $17 | $s1 | 0111 | Save | Def |
| $18 | $s2 | 1000 | Save | Def |
| $19 | $s3 | 1001 | save | Cust |
| $20 | $s4 | 1010 | Save | Cust |
| $21 | $s5 | 1011 | Save | Cust |
| $22 | $s6 | 1100 | save | Cust |
| | | | | |

## Question 7: Addressing Modes.

**Ans to the Question No-7**

**Addressing Modes:**

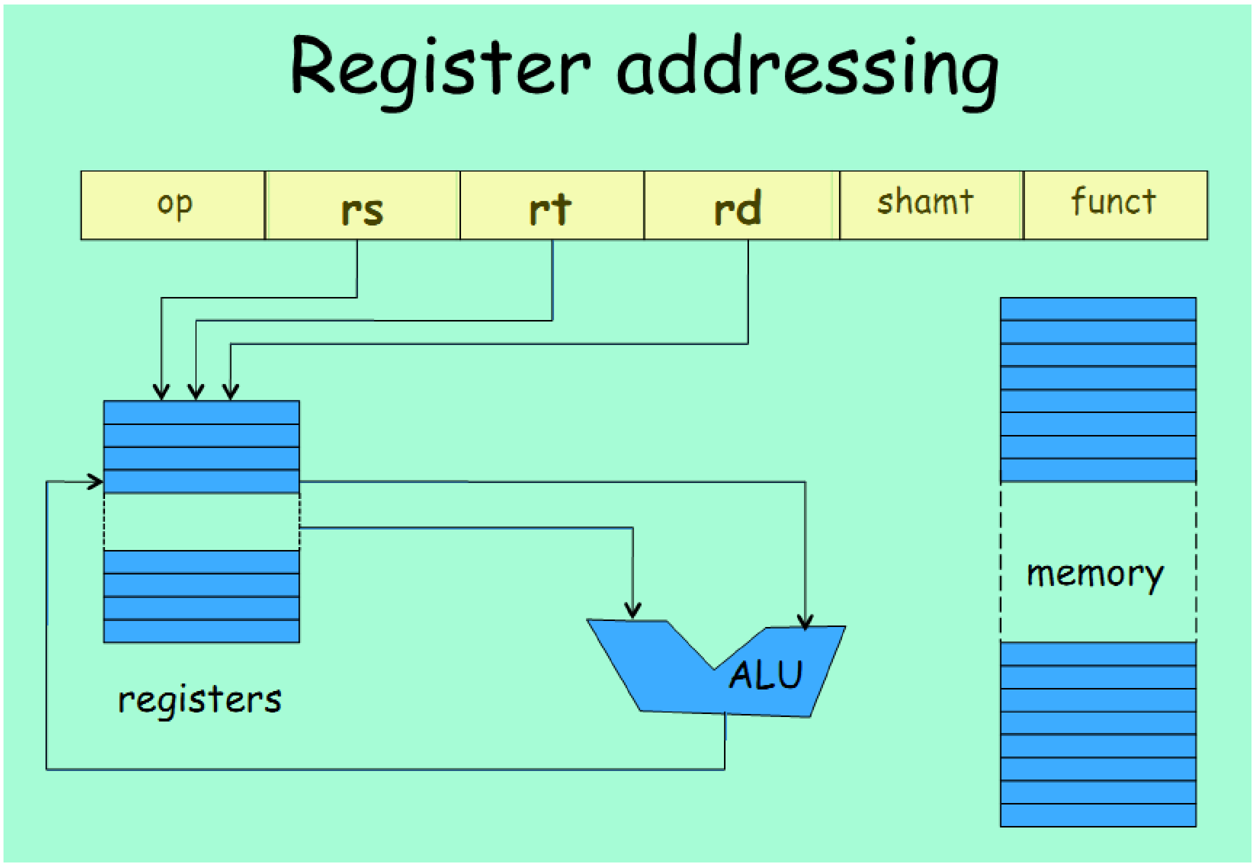
Addressing Modes are the ways of specifying an operand or a memory address. There are a number of addressing modes but we'll use 3 of them.

1. Register Addressing
2. Immediate Addressing
3. Base addressing

**Register Addressing:**

In this addressing mode operands are in a register. It takes n Bits to address $2^n$ register.

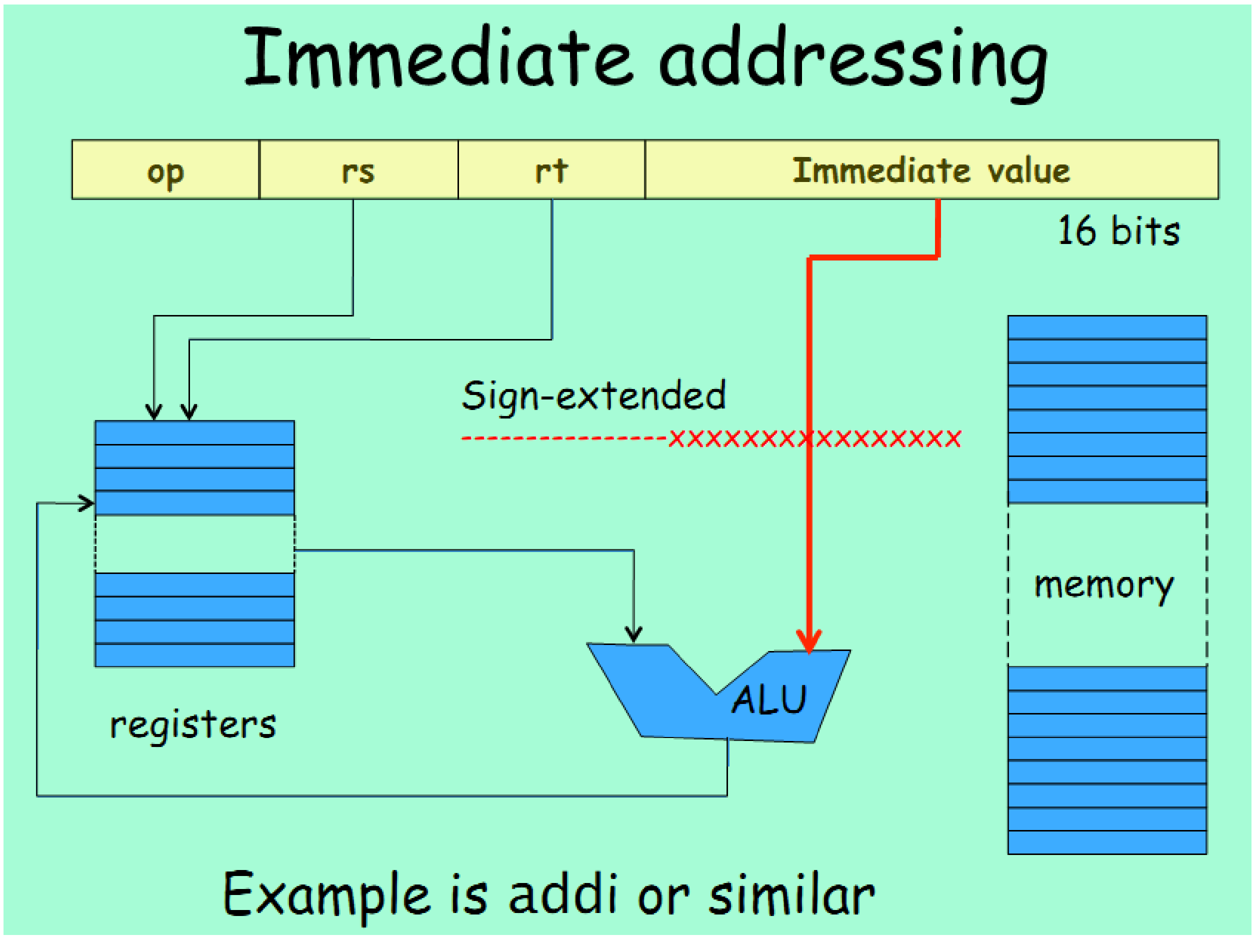
Example: **Sub $S1, $S2**



**Immediate Addressing:**

In this addressing mode operand is embedded inside the encoded instruction.

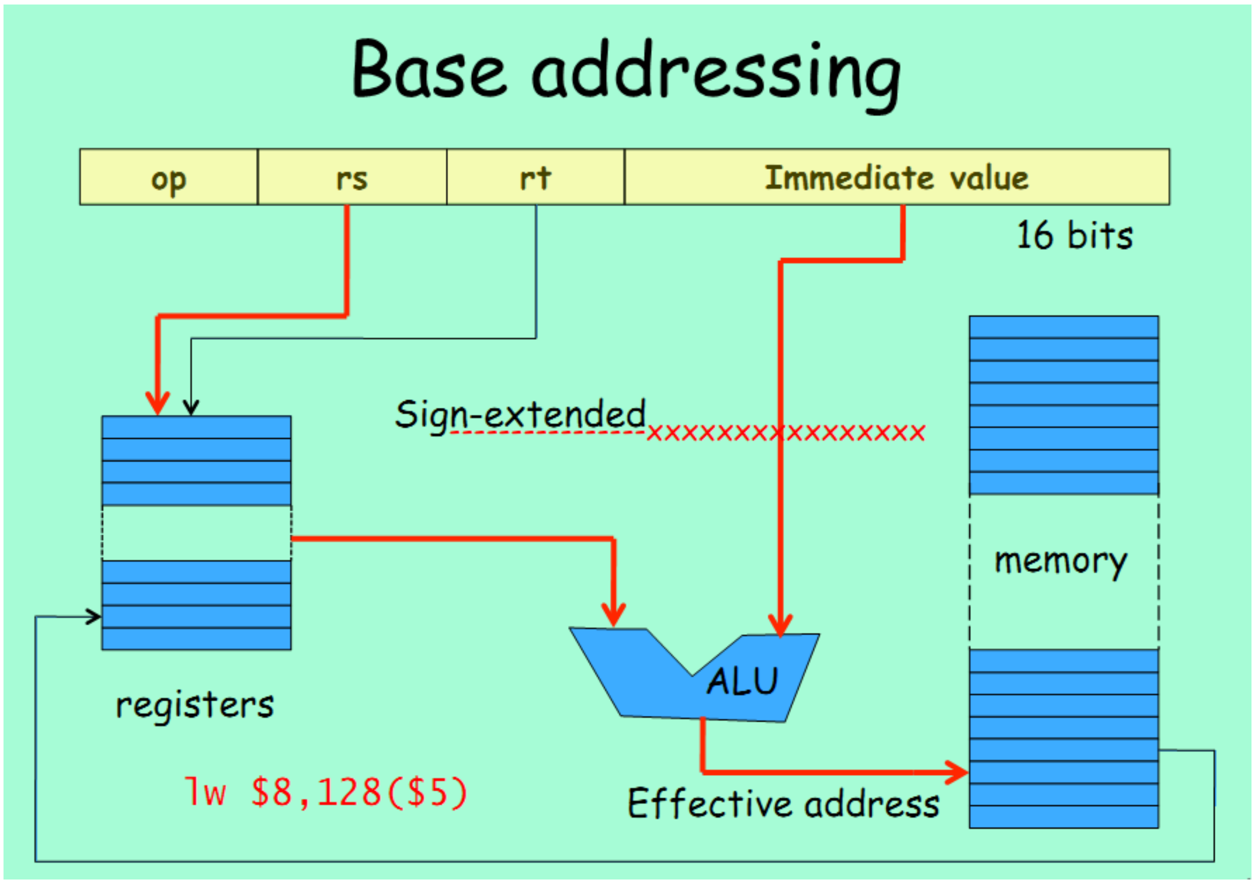
Example: **Add$_i$ $S1, 8**



**Base Addressing:**

In this addressing mode, the address of the operand is the sum of the immediate and the value in a register(rs).

Example: **Lw 10($S2)**


Base addressing
op
rs
rt
Immediate value
16 bits
Sign-extended
xxxxxxxxxxxxxxxx
registers
ALU
memory
lw $8,128($5)
Effective address