The different fields are:
- **op**: operation ("opcode") of the instruction
- **rs, rt, rd**: the source and destination register specifiers
- **shamt**: shift amount
- **funct**: selects the variant of the operation in the "op" field
- **address/immediate**: address offset or immediate value
- **target address**: target address of jump instruction
Register Transfer Language

- **RTL gives the meaning of the instructions**
  \[
  \text{inst} \equiv \{\text{op}, \text{rs}, \text{rt}, \text{rd}, \text{shamt}, \text{funct}\} = \text{MEM}[\text{PC}] \\
  \{\text{op}, \text{rs}, \text{rt}, \text{Imm16}\} = \text{MEM}[\text{PC}] \\
  \]

- **All start by fetching the instruction**
  - **Register Transfers**
    - **ADDU**
      \[
      \text{R}[\text{rd}] = \text{R}[\text{rs}] + \text{R}[\text{rt}]; \quad \text{PC} = \text{PC} + 4
      \]
    - **SUBU**
      \[
      \text{R}[\text{rd}] = \text{R}[\text{rs}] - \text{R}[\text{rt}]; \quad \text{PC} = \text{PC} + 4
      \]
    - **ORI**
      \[
      \text{R}[\text{rt}] = \text{R}[\text{rs}] \mid \text{zero_ext(Imm16)}; \quad \text{PC} = \text{PC} + 4
      \]
    - **LOAD**
      \[
      \text{R}[\text{rt}] = \text{MEM}[\text{R}[\text{rs}] + \text{sign_ext(Imm16)}]; \quad \text{PC} = \text{PC} + 4
      \]
    - **STORE**
      \[
      \text{MEM}[\text{R}[\text{rs}] + \text{sign_ext(Imm16)}] = \text{R}[\text{rt}]; \quad \text{PC} = \text{PC} + 4
      \]
    - **BEQ**
      \[
      \text{if ( R[rs] == R[rt] ) then} \quad \text{PC} = \text{PC} + 4 + (\text{sign_ext(Imm16)} \parallel 00) \\
      \text{else PC} = \text{PC} + 4
      \]

Step 1: Requirements of the Instruction Set

- **Memory (MEM)**
  - instructions & data
- **Registers (R: 32 x 32)**
  - read RS
  - read RT
  - Write RT or RD
- **PC**
- **Extender (sign extend)**
- **Add and Sub register or extended immediate**
- **Add 4 or extended immediate to PC**

Step 2: Components of the Datapath

- **Combinational Elements**
- **Storage Elements**
  - Clocking methodology

Combinational Logic Elements (Building Blocks)

- **Adder**
- **MUX**
- **ALU**

ALU Needs for MIPS-lite + Rest of MIPS

- **Addition, subtraction, logical OR, ==:**
  - **ADDU**
    \[
    \text{R}[\text{rd}] = \text{R}[\text{rs}] + \text{R}[\text{rt}]; \quad \ldots
    \]
  - **SUBU**
    \[
    \text{R}[\text{rd}] = \text{R}[\text{rs}] - \text{R}[\text{rt}]; \quad \ldots
    \]
  - **ORI**
    \[
    \text{R}[\text{rt}] = \text{R}[\text{rs}] \mid \text{zero_ext(Imm16)}; \quad \ldots
    \]
  - **BEQ**
    \[
    \text{if ( R[rs] == R[rt] )} \quad \ldots
    \]
  - **Test to see if output == 0 for any ALU operation gives == test. How?**
  - **P&H also adds AND, Set Less Than (1 if A < B, 0 otherwise)**

**ALU follows chap 5**

Administrivia

- **Final Exam location TBA (exam grp 5)**
  - **Sat, 2005-05-14, 12:30–3:30pm**
  - **ALL students are required to complete ALL of the exam (even if you aced the midterm)**
  - **Same format as the midterm**
    - 3 Hours
    - Closed book, except for 2 study sheets + green
    - Leave your backpacks, books at home
- **Homework 6 out today, due in a week (mon)**
**Storage Element: Idealized Memory**

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => Data Out valid after “access time.”

**Storage Element: Register (Building Block)**

- Similar to D Flip Flop except
  - N-bit input and output
  - Write Enable input
- Write Enable:
  - negated (or deasserted) (0): Data Out will not change
  - asserted (1): Data Out will become Data In

**Storage Element: Register File**

- Register File consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after “access time.”

**Step 3: Assemble DataPath meeting requirements**

- Register Transfer Requirements => Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation

**3a: Overview of the Instruction Fetch Unit**

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC = PC + 4
    - Branch and Jump: PC = “something else”

**3b: Add & Subtract**

- \( R[rd] = R[rs] \text{ op } R[rt] \)
- Ex.: addU rd, rs, rt
- Ra, Rb, and Rw come from instruction’s Rs, Rt, and Rd fields
- ALUctr and RegWr: control logic after decoding the instruction
- Already defined register file, ALU
Peer Instruction

A. We should use the main ALU to compute $PC = PC + 4$

B. We’re going to be able to read 2 registers and write a 3rd in 1 cycle

C. Datapath is hard, Control is easy

<table>
<thead>
<tr>
<th>ABC</th>
<th>FFF</th>
<th>FFT</th>
<th>FTF</th>
<th>FTT</th>
<th>TFF</th>
<th>TTF</th>
<th>TTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) => datapath requirements
   - meaning of each instruction is given by the register transfers
   - datapath must include storage element for ISA registers
   - datapath must support each register transfer

2. Select set of datapath components and establish clocking methodology

3. Assemble datapath meeting requirements

4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

5. Assemble the control logic (hard part!)