Chapter 7: Microarchitecture

Extending the Single-Cycle Processor

Extended Functionality: I-Type ALU

Enhance the single-cycle processor to handle **I-Type ALU instructions**: `addi`, `andi`, `ori`, and `slti`

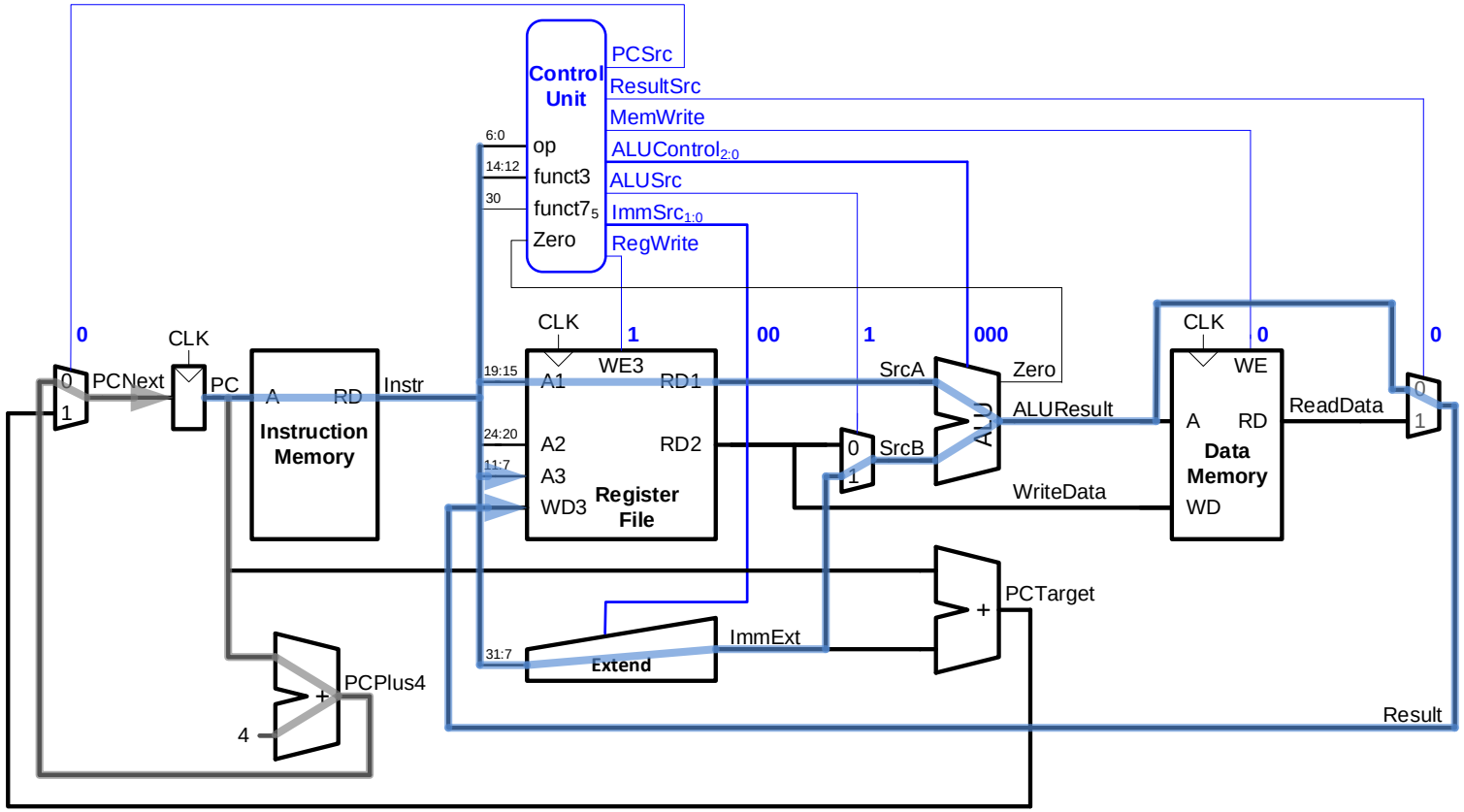
- **Similar to R-type** instructions
- But **second source** comes from **immediate**
- Change ***ALUSrc*** to select the immediate
- And ***ImmSrc*** to pick the correct immediate

Extended Functionality: I-Type ALU

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	**lw**	1	00	1	0	1	0	00
35	**sw**	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	**beq**	0	10	0	0	X	1	01
19	**I-type**	1	00	1	0	0	0	10

Extended Functionality: addi

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
19	**I-type**	1	00	1	0	0	0	10



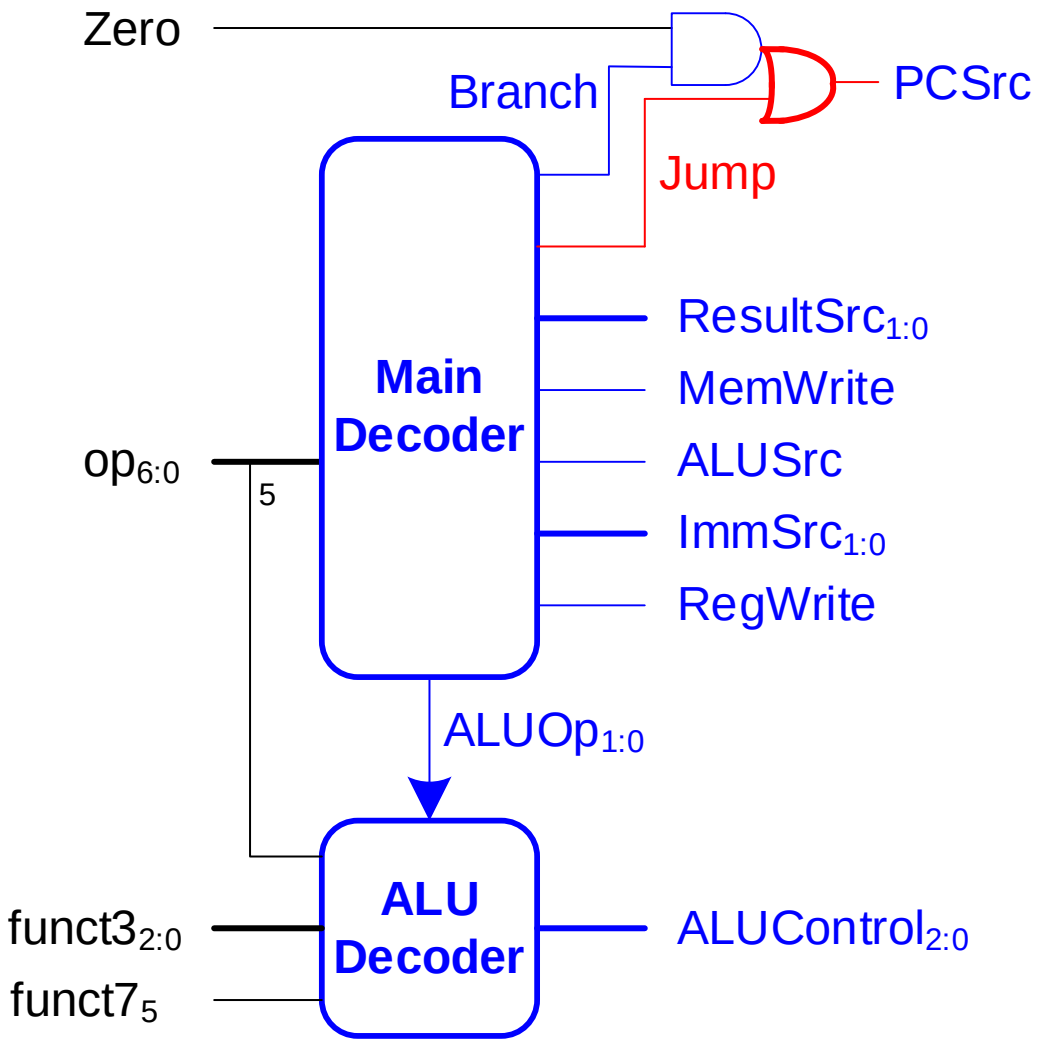
addi x5, x6, -33

Extended Functionality: `jal`

Enhance the single-cycle processor to handle `jal`

- **Similar to `beq`**
- But jump is **always taken**
  - *PCSrc* should be 1
- **Immediate format** is different
  - Need a new *ImmSrc* of 11
- And `jal` must **compute PC+4** and **store in `rd`**
  - Take PC+4 from adder through ResultMux

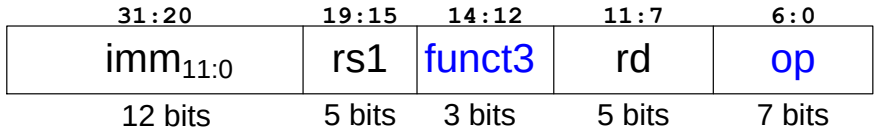
Extended Functionality: jal



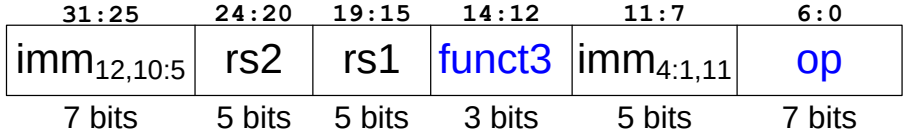
Extended Functionality: *ImmExt*

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type
11	{{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0}	J-Type

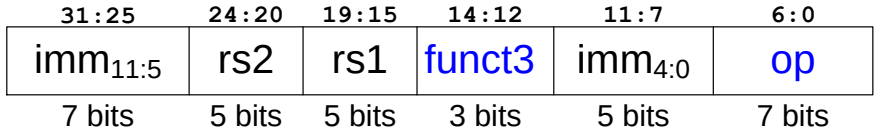
I-Type



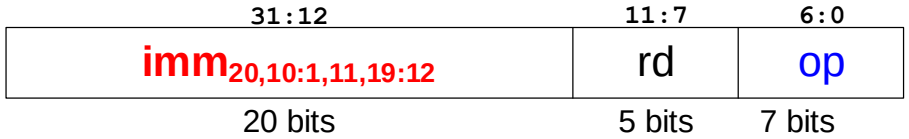
B-Type



S-Type



J-Type

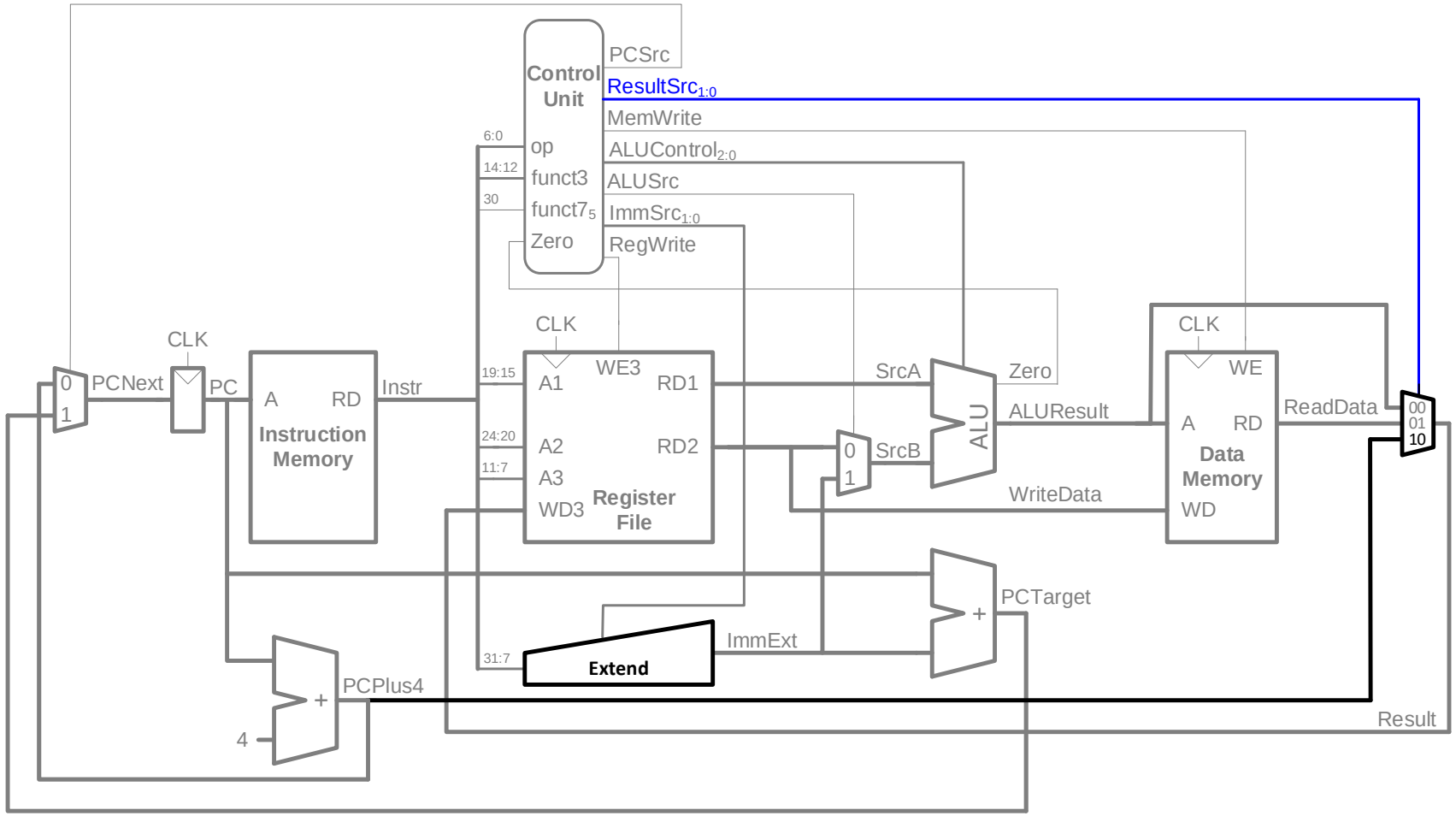


Extended Functionality: jal

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
3	**lw**	1	00	1	0	01	0	00	0
35	**sw**	0	01	1	1	XX	0	00	0
51	R-type	1	XX	0	0	00	0	10	0
99	**beq**	0	10	0	0	XX	1	01	0
19	**I-type**	1	00	1	0	00	0	10	0
111	**jal**	1	11	X	0	10	0	XX	1

Extended Functionality: jal

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
111	jal	1	11	X	0	10	0	XX	1



Chapter 7: Microarchitecture

Single-Cycle Performance

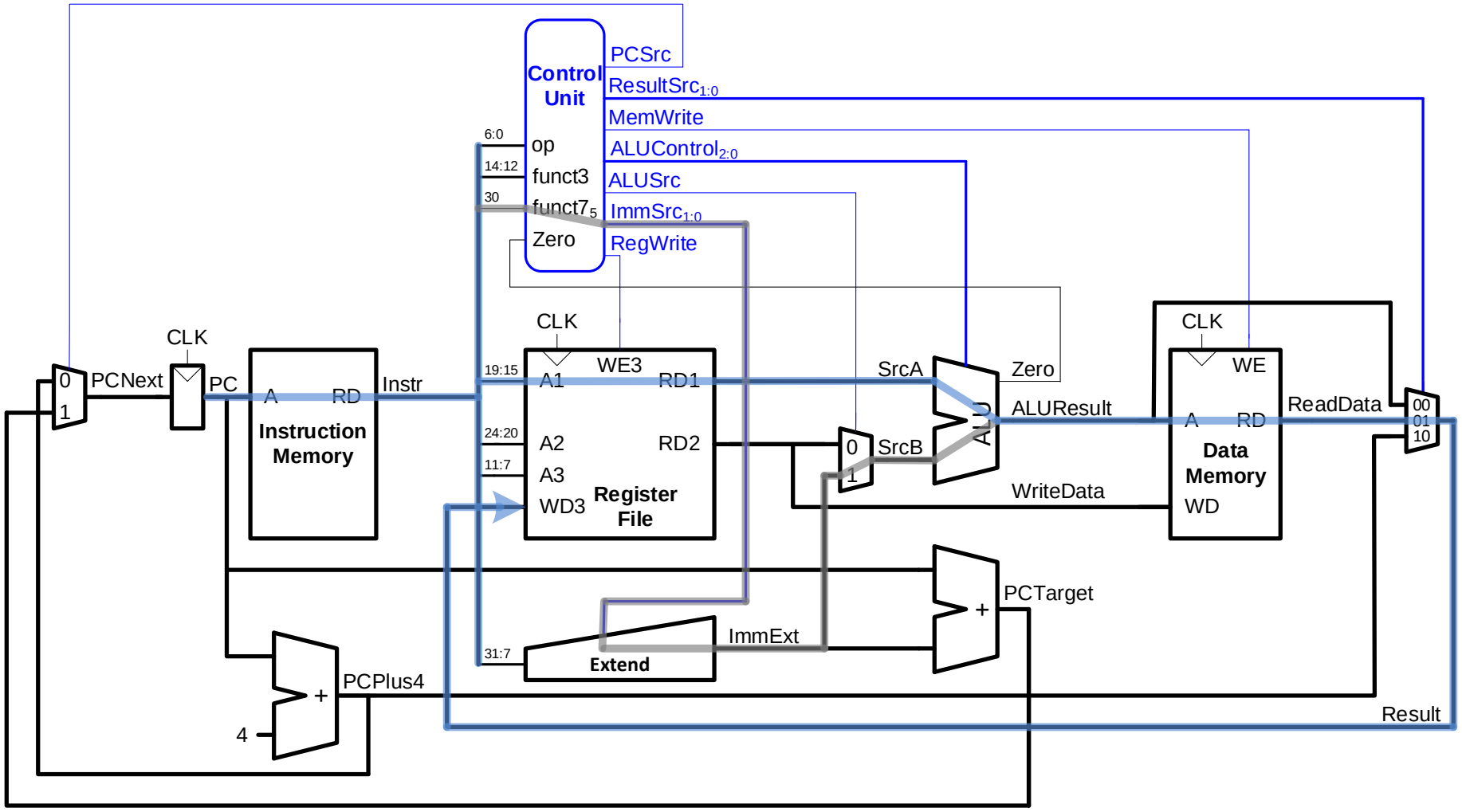
Processor Performance

Program Execution Time

= (#instructions)(cycles/instruction)(seconds/cycle)

= # instructions x CPI x T_C

Single-Cycle Processor Performance



T_C limited by critical path (`lw`)

Single-Cycle Processor Performance

- **Single-cycle critical path:**

$$T_{c\_single} = t_{pcq\_PC} + t_{\text{mem}} + \max[t_{RF\text{read}}, t_{dec} + t_{ext} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- **Typically, limiting paths are:**
  - memory, ALU, register file
  - *So,*

$$T_{c\_single} = t_{pcq\_PC} + t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$
$$= t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mux}} + t_{RF\text{setup}}$$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{\text{AND-OR}}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend unit	t_{ext}	35
Memory read	t_{mem}	200
Register file read	$t_{RF\text{read}}$	100
Register file setup	$t_{RF\text{setup}}$	60

$$T_{c\_single} = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mux}} + t_{RF\text{setup}}$$
$$=$$

Single-Cycle Performance Example

Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C