- Performance $= 1/(\text{Execution time})$
- $\text{CPU}_{\text{time}} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time (T)} = \text{Instruction Count (IC)} \times \text{Cycles per Instruction}$

  $= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instructuin}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$ CPI：Cycles per Instruction
  - Average CPI 可能會< 1
  - CPI 會受 Computer architecture 影響，因為 pipeline 可以影響 CPI
- Amdahl's Law：$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{Improvement factor}} + T_{\text{unaffected}}$，Speedup $= \frac{1}{\frac{f}{n}+(1-f)}$

  f：% that can be improved ($T_{\text{aff}}/T_{\text{orig}}$), n：improvement factor

	Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
add, sub, and, or, xor, sll, srl	R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
addi, andi, ori, …, ld	I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
sd ex. sd x9, 120 (x22)	S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
beq(=), bne(≠), blt(<), bge(≥)	SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
jal, jalr	UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
lui (load upper immediate)	U-type	immediate[31:12]				rd	opcode	Upper immediate format

- 【R-format】For srl, shift right and fill with 0 bits. For sra, shift right and fill with sign bits.
- 【I-format】immediate → constant operand, -2^{11}(-2048) to 2^{11}-1(2047)

  For "slli", only use imm[4:0] cuz 32-bit data can only be shifted 32 bit positions

  For "ld", ex. ld x9, 120(x22) → immediate = 120, rs1 = 22, rd = 9
- 【S-format】For "sd", ex. sd x9, 120(x22) → immediate = 120, rs1 = 22, rs2 = 9
- 【SB-format】imm 只放 13 bits 中的 12 bits，最右邊的"0"被丟掉。

  bne rs1, rs2, +8 = bne rs1, rs2, 4 → "+8"表示"PC+8", "4"表示"immediate 的值"

  ex.【beq rs1, rs2, imm】"imm" is "Branch offset" = n×32-bit instructions = 4n bytes

  target address = PC + immediate
- 【U-format】lui x19, 26138 (107062541 / 4096 取整數) 【lui rd, constant】

  addi x19, x19, 1293 (107062541 % 4096)
- 【UJ-format】jal x1, ProcedureLabel (jumps to address "ProcedureLabel")

  Jalr x0, 0(x1) … return to caller
- ld：memory → reg。sd：reg → memory。
- ex. ld x9, 120(x22) → x22 裡面本應要放 memory, 此是 visit x22(register) get value → use this value (address of memory) to find the value in the memory.

RISC-V Procedure Call Convention

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	---
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	---
x4	tp	Thread pointer	---
x5	t0	Temporary/alternate link register	Caller
x6~x7	t1~t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10~x11	a0~a1	Function arguments/return values	Caller
x12~x17	a2~a7	Function arguments	Caller
x18~x27	s2~s11	Saved registers	Callee
x28~x31	t3~t6	Temporaries	Caller

Fig. 2.1

Non-Leaf Procedure Example (RISC-V Code)

```
fact: addi sp,sp,-16     // make space on stack
      sd   x1,8(sp)      // save return address in x1 onto stack
      sd   x10,0(sp)     // save argument in x10 onto stack
      addi x5,x10,-1     // x5 = n – 1
      bge  x5,x0,L1      // if n >= 1, go to L1
      addi x10,x0,1      // else, set return value to 1
      addi sp,sp,16      // pop stack, don't bother restoring values
      jalr x0,0(x1)      // return
L1:   addi x10,x10,-1    // n = n – 1
      jal  x1,fact       // call fact(n – 1)
      addi x6,x10,0      // move return value of fact(n – 1) to x6
      ld   x10,0(sp)     // restore caller's n
      ld   x1,8(sp)      // restore return address
      addi sp,sp,16      // return space on stack
      mul  x10,x10,x6    // return n * fact(n – 1)
      jalr x0,0(x1)      // return
```

- Design Principle：Simplicity favors regularity, Smaller is faster, Make the common case fast
-