

COMP110: Principles of Computing

10: Machine Architecture

Learning outcomes

- ▶ **Explain** the difference between interpretation, just-in-time compilation and ahead-of-time compilation
- ▶ **Describe** how common high-level code structures translate to machine code
- ▶ **Explain** how floating-point numbers are represented in the computer

How programs are executed



Executing programs

- ▶ CPUs execute **machine code**

Executing programs

- ▶ CPUs execute **machine code**
- ▶ Programs must be **translated** into machine code for execution

Executing programs

- ▶ CPUs execute **machine code**
- ▶ Programs must be **translated** into machine code for execution
- ▶ There are three main ways of doing this:

Executing programs

- ▶ CPUs execute **machine code**
- ▶ Programs must be **translated** into machine code for execution
- ▶ There are three main ways of doing this:
 - ▶ An **interpreter** is an application which reads the program source code and executes it directly

Executing programs

- ▶ CPUs execute **machine code**
- ▶ Programs must be **translated** into machine code for execution
- ▶ There are three main ways of doing this:
 - ▶ An **interpreter** is an application which reads the program source code and executes it directly
 - ▶ An **ahead-of-time (AOT) compiler**, often just called a **compiler**, is an application which converts the program source code into executable machine code

Executing programs

- ▶ CPUs execute **machine code**
- ▶ Programs must be **translated** into machine code for execution
- ▶ There are three main ways of doing this:
 - ▶ An **interpreter** is an application which reads the program source code and executes it directly
 - ▶ An **ahead-of-time (AOT) compiler**, often just called a **compiler**, is an application which converts the program source code into executable machine code
 - ▶ A **just-in-time (JIT) compiler** is halfway between the two — it compiles the program on-the-fly at runtime

Examples

Interpreted:

- ▶ Python
- ▶ Lua
- ▶ JavaScript
(in old web
browsers)
- ▶ Bespoke
scripting
languages

Examples

Interpreted:

- ▶ Python
- ▶ Lua
- ▶ JavaScript
(in old web
browsers)
- ▶ Bespoke
scripting
languages

Compiled:

- ▶ C
- ▶ C++
- ▶ Swift
- ▶ Rust

Examples

Interpreted:

- ▶ Python
- ▶ Lua
- ▶ JavaScript
(in old web browsers)
- ▶ Bespoke scripting languages

Compiled:

- ▶ C
- ▶ C++
- ▶ Swift
- ▶ Rust

JIT compiled:

- ▶ Java
- ▶ C#
- ▶ JavaScript
(in modern web browsers)
- ▶ Jython

Examples

Interpreted:

- ▶ Python
- ▶ Lua
- ▶ JavaScript
(in old web browsers)
- ▶ Bespoke scripting languages

Compiled:

- ▶ C
- ▶ C++
- ▶ Swift
- ▶ Rust

JIT compiled:

- ▶ Java
- ▶ C#
- ▶ JavaScript
(in modern web browsers)
- ▶ Jython

NB: technically any language could appear in any column here, but this is where they typically are

Interpreter vs compiler

- ▶ Run-time efficiency: compiler > interpreter

Interpreter vs compiler

- ▶ Run-time efficiency: compiler > interpreter
 - ▶ The compiler translates the program **in advance**, on the developer's machine

Interpreter vs compiler

- ▶ Run-time efficiency: compiler > interpreter
 - ▶ The compiler translates the program **in advance**, on the developer's machine
 - ▶ The interpreter translates the program **at runtime**, on the user's machine — this takes extra time

Interpreter vs compiler

- ▶ Portability: compiler < interpreter

Interpreter vs compiler

- ▶ Portability: compiler < interpreter
 - ▶ A compiled program can only run on the operating system and CPU architecture it was compiled for

Interpreter vs compiler

- ▶ Portability: compiler < interpreter
 - ▶ A compiled program can only run on the operating system and CPU architecture it was compiled for
 - ▶ An interpreted program can run on any machine, as long as a suitable interpreter is available

Interpreter vs compiler

- Ease of development: compiler < interpreter

Interpreter vs compiler

- ▶ Ease of development: compiler < interpreter
 - ▶ Writing an AOT or JIT compiler (especially a good one) is hard, and required in-depth knowledge of the target machine

Interpreter vs compiler

- ▶ Ease of development: compiler < interpreter
 - ▶ Writing an AOT or JIT compiler (especially a good one) is hard, and required in-depth knowledge of the target machine
 - ▶ Writing an interpreter is easy in comparison

Interpreter vs compiler

- Dynamic language features: compiler < interpreter

Interpreter vs compiler

- ▶ Dynamic language features: compiler < interpreter
 - ▶ The interpreter is already on the end user's machine, so programs can use it e.g. to dynamically generate and execute new code

Interpreter vs compiler

- ▶ Dynamic language features: compiler < interpreter
 - ▶ The interpreter is already on the end user's machine, so programs can use it e.g. to dynamically generate and execute new code
 - ▶ The AOT compiler is not generally on the end user's machine, so this is more difficult

Interpreter vs compiler

Interpreter vs compiler

- ▶ JIT compilers have similar pros/cons to interpreters

Interpreter vs compiler

- ▶ JIT compilers have similar pros/cons to interpreters
 - ▶ Runtime efficiency: JIT > interpreter (e.g. code inside a loop only needs to be translated once, then can be executed many times)

Interpreter vs compiler

- ▶ JIT compilers have similar pros/cons to interpreters
 - ▶ Runtime efficiency: JIT > interpreter (e.g. code inside a loop only needs to be translated once, then can be executed many times)
 - ▶ Ease of development: JIT < interpreter

Virtual machines

Virtual machines

- ▶ Many modern interpreters and JIT compilers translate programs into **bytecode**

Virtual machines

- ▶ Many modern interpreters and JIT compilers translate programs into **bytecode**
- ▶ Bytecode is essentially machine code for a **virtual machine (VM)**

Virtual machines

- ▶ Many modern interpreters and JIT compilers translate programs into **bytecode**
- ▶ Bytecode is essentially machine code for a **virtual machine (VM)**
- ▶ Translation from source code to bytecode can be done ahead of time

Virtual machines

- ▶ Many modern interpreters and JIT compilers translate programs into **bytecode**
- ▶ Bytecode is essentially machine code for a **virtual machine (VM)**
- ▶ Translation from source code to bytecode can be done ahead of time
- ▶ At runtime, translate the bytecode (by interpretation or JIT compilation) into machine code for the physical machine

Virtual machines

- ▶ Many modern interpreters and JIT compilers translate programs into **bytecode**
- ▶ Bytecode is essentially machine code for a **virtual machine (VM)**
- ▶ Translation from source code to bytecode can be done ahead of time
- ▶ At runtime, translate the bytecode (by interpretation or JIT compilation) into machine code for the physical machine
- ▶ E.g. a Java JAR file, a .NET executable, a Python .pyc or .pyo file all contain bytecode for their respective VMs

Assemblers

Assemblers

- ▶ **Assembly language** is designed to translate directly into machine code

Assemblers

- ▶ **Assembly language** is designed to translate directly into machine code
- ▶ An ahead-of-time compile for assembly language is called an **assembler**

Assemblers

- ▶ **Assembly language** is designed to translate directly into machine code
- ▶ An ahead-of-time compile for assembly language is called an **assembler**
- ▶ Generally much simpler than an AOT compiler for a higher-level language

The MIPS architecture



MIPS

MIPS

- ▶ An example of a **Reduced Instruction Set Computer (RISC)** architecture

MIPS

- ▶ An example of a **Reduced Instruction Set Computer (RISC)** architecture
 - ▶ Small number of simple instructions — computational power comes from executing many instructions per second

MIPS

- ▶ An example of a **Reduced Instruction Set Computer (RISC)** architecture
 - ▶ Small number of simple instructions — computational power comes from executing many instructions per second
 - ▶ Compare with **Complex Instruction Set Computer (CISC)** architecture (e.g. Intel x86) — large number of complex instructions — fewer instructions per second, but shorter programs

MIPS

- ▶ An example of a **Reduced Instruction Set Computer (RISC)** architecture
 - ▶ Small number of simple instructions — computational power comes from executing many instructions per second
 - ▶ Compare with **Complex Instruction Set Computer (CISC)** architecture (e.g. Intel x86) — large number of complex instructions — fewer instructions per second, but shorter programs
- ▶ MIPS was popular in 1980s – 2000s

MIPS

- ▶ An example of a **Reduced Instruction Set Computer (RISC)** architecture
 - ▶ Small number of simple instructions — computational power comes from executing many instructions per second
 - ▶ Compare with **Complex Instruction Set Computer (CISC)** architecture (e.g. Intel x86) — large number of complex instructions — fewer instructions per second, but shorter programs
- ▶ MIPS was popular in 1980s – 2000s
 - ▶ Embedded systems

MIPS

- ▶ An example of a **Reduced Instruction Set Computer (RISC)** architecture
 - ▶ Small number of simple instructions — computational power comes from executing many instructions per second
 - ▶ Compare with **Complex Instruction Set Computer (CISC)** architecture (e.g. Intel x86) — large number of complex instructions — fewer instructions per second, but shorter programs
- ▶ MIPS was popular in 1980s – 2000s
 - ▶ Embedded systems
 - ▶ Consoles (Nintendo 64, PlayStation 1 and 2)

MIPS

- ▶ An example of a **Reduced Instruction Set Computer (RISC)** architecture
 - ▶ Small number of simple instructions — computational power comes from executing many instructions per second
 - ▶ Compare with **Complex Instruction Set Computer (CISC)** architecture (e.g. Intel x86) — large number of complex instructions — fewer instructions per second, but shorter programs
- ▶ MIPS was popular in 1980s – 2000s
 - ▶ Embedded systems
 - ▶ Consoles (Nintendo 64, PlayStation 1 and 2)
- ▶ Easier to understand than most CPU instruction sets in common use today

Online MIPS simulator

<http://rivoire.cs.sonoma.edu/cs351/wemips/>

Registers

Registers

- ▶ Memory locations inside the CPU

Registers

- ▶ Memory locations inside the CPU
- ▶ Faster to access than main memory

Registers

- ▶ Memory locations inside the CPU
- ▶ Faster to access than main memory
- ▶ Registers in MIPS architecture include:

Registers

- ▶ Memory locations inside the CPU
- ▶ Faster to access than main memory
- ▶ Registers in MIPS architecture include:
 - ▶ `$zero`: constant 0

Registers

- ▶ Memory locations inside the CPU
- ▶ Faster to access than main memory
- ▶ Registers in MIPS architecture include:
 - ▶ `$zero`: constant 0
 - ▶ `$t0–$t9`: temporary storage

Registers

- ▶ Memory locations inside the CPU
- ▶ Faster to access than main memory
- ▶ Registers in MIPS architecture include:
 - ▶ `$zero`: constant 0
 - ▶ `$t0–$t9`: temporary storage
 - ▶ `$s0–$s7`: saved temporary storage

Registers

- ▶ Memory locations inside the CPU
- ▶ Faster to access than main memory
- ▶ Registers in MIPS architecture include:
 - ▶ `$zero`: constant 0
 - ▶ `$t0–$t9`: temporary storage
 - ▶ `$s0–$s7`: saved temporary storage
- ▶ Each register holds a single 32-bit value

Adding register values

Adding register values

```
add $d, $s, $t
```

Adding register values

```
add $d, $s, $t
```

- ▶ `$d`, `$s` and `$t` are register names

Adding register values

```
add $d, $s, $t
```

- ▶ `$d`, `$s` and `$t` are register names
- ▶ This adds the value of `$s` to the value of `$t`, and stores the result in `$d`

Adding register values

```
add $d, $s, $t
```

- ▶ `$d`, `$s` and `$t` are register names
- ▶ This adds the value of `$s` to the value of `$t`, and stores the result in `$d`

```
sub $d, $s, $t
```

Adding register values

```
add $d, $s, $t
```

- ▶ `$d`, `$s` and `$t` are register names
- ▶ This adds the value of `$s` to the value of `$t`, and stores the result in `$d`

```
sub $d, $s, $t
```

- ▶ Subtracts the value of `$t` from the value of `$s`, and stores the result in `$d`

Adding a constant

Adding a constant

```
addi $d, $s, C
```

Adding a constant

```
addi $d, $s, c
```

- ▶ `$d` and `$s` are register names, `c` is an integer constant

Adding a constant

```
addi $d, $s, c
```

- ▶ `$d` and `$s` are register names, `c` is an integer constant
- ▶ This adds the value of `$s` to `c`, and stores the result in `$d`

Adding a constant

```
addi $d, $s, c
```

- ▶ `$d` and `$s` are register names, `c` is an integer constant
- ▶ This adds the value of `$s` to `c`, and stores the result in `$d`
- ▶ `addi` = “add immediate” — as in `c` is specified immediately in the code, not looked up from a register

Adding a constant

```
addi $d, $s, c
```

- ▶ `$d` and `$s` are register names, `c` is an integer constant
- ▶ This adds the value of `$s` to `c`, and stores the result in `$d`
- ▶ `addi` = “add immediate” — as in `c` is specified immediately in the code, not looked up from a register
- ▶ There is no `subi` instruction — to subtract `c`, add `-c`

More fun with addi

More fun with addi

- Socrative FALCOMPED

More fun with addi

- ▶ Socratic FALCOMPED
- ▶ What does this code do?

```
addi $s0, $s1, 0
```


More fun with addi

- ▶ Socratic FALCOMPED
- ▶ What does this code do?

```
addi $s0, $s1, 0
```

- ▶ What does this code do?

```
addi $s0, $zero, 12
```

More fun with addi

- ▶ Socratic FALCOMPED
- ▶ What does this code do?

```
addi $s0, $s1, 0
```

- ▶ What does this code do?

```
addi $s0, $zero, 12
```

- ▶ MIPS does not have dedicated instructions for setting a register value to a constant or to the value of another register — it has to be done with **addi**

Control flow in MIPS



Labels and jumping

Labels and jumping

- In assembly code, can set a **label** on any line:

```
MyLabel: add $s0, $s1, 1
```

Labels and jumping

- ▶ In assembly code, can set a **label** on any line:

```
MyLabel: add $s0, $s1, 1
```

- ▶ Some instructions use labels to refer to a location in the code

Labels and jumping

- ▶ In assembly code, can set a **label** on any line:

```
MyLabel: add $s0, $s1, 1
```

- ▶ Some instructions use labels to refer to a location in the code
- ▶ E.g. the **j** instruction simply jumps (backwards or forwards) to the specified line:

```
j MyLabel
```

Branching

Branching

- ▶ **Branching** is conditional jumping

Branching

- **Branching** is conditional jumping

```
beq $s, $t, Label
```

Branching

- ▶ **Branching** is **conditional jumping**

```
beq $s, $t, Label
```

- ▶ This jumps to `Label` **if and only if** the value of `$s` equals the value of `$t`

Branching

- **Branching** is **conditional jumping**

```
beq $s, $t, Label
```

- This jumps to `Label` **if and only if** the value of `$s` equals the value of `$t`

```
bne $s, $t, Label
```

Branching

- **Branching** is **conditional jumping**

```
beq $s, $t, Label
```

- This jumps to `Label` **if and only if** the value of `$s` equals the value of `$t`

```
bne $s, $t, Label
```

- This jumps to `Label` **if and only if** the value of `$s` does not equal the value of `$t`

Conditionals

Conditionals

- ▶ Branching allows us to implement **if statements**

Conditionals

- Branching allows us to implement **if statements**

```
if s0 != 0:  
    s1 += 1  
else:  
    s2 += 1
```


Conditionals

- Branching allows us to implement **if statements**

```
if s0 != 0:  
    s1 += 1  
else:  
    s2 += 1
```

```
beq $s0, $zero, Else  
addi $s1, $s1, 1  
j End  
Else: addi $s2, $s2, 1  
End:
```

Loops

Loops

- ▶ Branching allows us to implement **while loops**

Loops

- Branching allows us to implement **while loops**

```
i = 0
total = 0
limit = 10

while i != limit:
    total += i
    i += 1
#end while
```

Loops

- Branching allows us to implement **while loops**

```
i = 0
total = 0
limit = 10
```

```
while i != limit:
    total += i
    i += 1
#end while
```

```
addi $s0, $zero, 0
addi $s1, $zero, 0
addi $s2, $zero, 10
```

```
Loop: beq $s0, $s2, LoopEnd
add $s1, $s1, $s0
addi $s0, $s0, 1
j Loop
LoopEnd:
```

Function calls

Function calls

- ▶ Function calls can be implemented using the jump instruction:

Function calls

- ▶ Function calls can be implemented using the jump instruction:
 - ▶ To call the function: save the address of the instruction after the current one, then jump to the function

Function calls

- ▶ Function calls can be implemented using the jump instruction:
 - ▶ To call the function: save the address of the instruction after the current one, then jump to the function
 - ▶ To return from the function: jump to the previously saved address

Function calls

- ▶ Function calls can be implemented using the jump instruction:
 - ▶ To call the function: save the address of the instruction after the current one, then jump to the function
 - ▶ To return from the function: jump to the previously saved address
- ▶ MIPS has `j``al` and `j``r` instructions and `$ra` register for this purpose

Function calls

- ▶ Function calls can be implemented using the jump instruction:
 - ▶ To call the function: save the address of the instruction after the current one, then jump to the function
 - ▶ To return from the function: jump to the previously saved address
- ▶ MIPS has `j` and `jal` instructions and `$ra` register for this purpose
- ▶ Socratic `FALCOMPED`: why save the return address? Why not just hard-code it into the program?

Function calls

- ▶ Function calls can be implemented using the jump instruction:
 - ▶ To call the function: save the address of the instruction after the current one, then jump to the function
 - ▶ To return from the function: jump to the previously saved address
- ▶ MIPS has `j` and `jal` instructions and `$ra` register for this purpose
- ▶ Socratic `FALCOMPED`: why save the return address? Why not just hard-code it into the program?
- ▶ Nested function calls require a **stack** of return addresses

MIPS machine code



MIPS instructions

MIPS instructions

- ▶ Each line of MIPS assembly code can be translated into a **machine code instruction**

MIPS instructions

- ▶ Each line of MIPS assembly code can be translated into a **machine code instruction**
- ▶ 1 line of assembly = 1 instruction

MIPS instructions

- ▶ Each line of MIPS assembly code can be translated into a **machine code instruction**
- ▶ 1 line of assembly = 1 instruction
- ▶ Each instruction is a **32 bit** value

MIPS instructions

- ▶ Each line of MIPS assembly code can be translated into a **machine code instruction**
- ▶ 1 line of assembly = 1 instruction
- ▶ Each instruction is a **32 bit** value
- ▶ First 6 bits specify the **opcode**; how the remaining 26 bits are interpreted depends on which opcode it is

Anatomy of an instruction

Anatomy of an instruction

R-type instruction:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Anatomy of an instruction

R-type instruction:

																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

- **opcode** and **function** together specify the operation to execute

Anatomy of an instruction

R-type instruction:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- ▶ **opcode** and **function** together specify the operation to execute
 - ▶ E.g. **add** has opcode 000000 and function 100000

Anatomy of an instruction

R-type instruction:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- ▶ **opcode** and **function** together specify the operation to execute
 - ▶ E.g. **add** has opcode 000000 and function 100000
 - ▶ E.g. **sub** has opcode 000000 and function 100010

Anatomy of an instruction

R-type instruction:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- ▶ **opcode** and **function** together specify the operation to execute
 - ▶ E.g. **add** has opcode 000000 and function 100000
 - ▶ E.g. **sub** has opcode 000000 and function 100010
- ▶ Some instructions specify a **shift** amount

Anatomy of an instruction

R-type instruction:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- ▶ **opcode** and **function** together specify the operation to execute
 - ▶ E.g. **add** has opcode 000000 and function 100000
 - ▶ E.g. **sub** has opcode 000000 and function 100010
- ▶ Some instructions specify a **shift** amount
 - ▶ For **add sub** etc these 5 bits are ignored

Anatomy of an instruction

R-type instruction:

																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

- ▶ **opcode** and **function** together specify the operation to execute
 - ▶ E.g. **add** has opcode 000000 and function 100000
 - ▶ E.g. **sub** has opcode 000000 and function 100010
- ▶ Some instructions specify a **shift** amount
 - ▶ For **add sub** etc these 5 bits are ignored
- ▶ Registers are identified by a 5-bit number

Anatomy of an instruction

R-type instruction:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- ▶ **opcode** and **function** together specify the operation to execute
 - ▶ E.g. **add** has opcode 000000 and function 100000
 - ▶ E.g. **sub** has opcode 000000 and function 100010
- ▶ Some instructions specify a **shift** amount
 - ▶ For **add sub** etc these 5 bits are ignored
- ▶ Registers are identified by a 5-bit number
 - ▶ E.g. **\$zero** → 00000, **\$s0** → 01000, **\$s1** → 01001

Anatomy of an instruction

R-type instruction:

[illegible]

- ▶ **opcode** and **function** together specify the operation to execute
 - ▶ E.g. **add** has opcode 000000 and function 100000
 - ▶ E.g. **sub** has opcode 000000 and function 100010
- ▶ Some instructions specify a **shift** amount
 - ▶ For **add sub** etc these 5 bits are ignored
- ▶ Registers are identified by a 5-bit number
 - ▶ E.g. **\$zero** → 00000, **\$s0** → 01000, **\$s1** → 01001
 - ▶ There are 32 registers

Example

```
add $s0, $s0, $s1
```

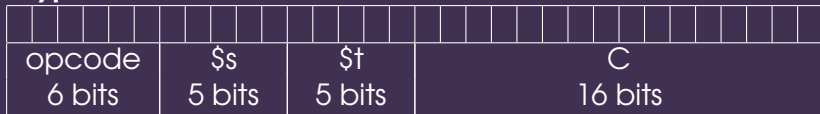


opcode	s	t	d	shift	function
000000	01000	01001	01001	00000	100000

Anatomy of an instruction

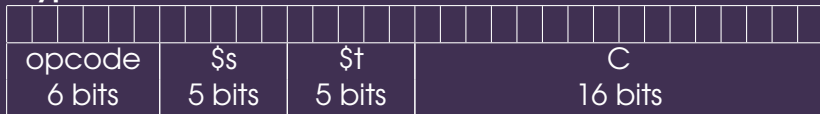
Anatomy of an instruction

I-type instruction:



Anatomy of an instruction

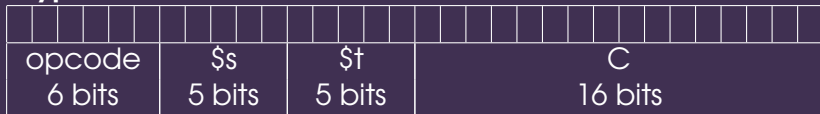
I-type instruction:



- ▶ **opcode** specifies the operation to execute

Anatomy of an instruction

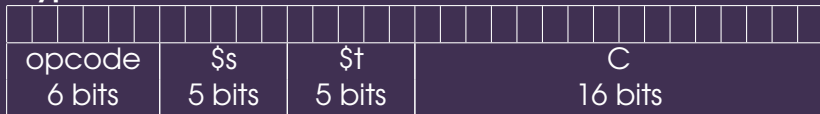
I-type instruction:



- ▶ **opcode** specifies the operation to execute
 - ▶ E.g. **addi** has opcode 001000

Anatomy of an instruction

I-type instruction:



- ▶ **opcode** specifies the operation to execute
 - ▶ E.g. **addi** has opcode 001000
- ▶ **c** is specified as a 16-bit number

Example

```
addi $s0, $s1, 123
```



opcode	s	t	C
001000	01001	01000	0000000001111011

Anatomy of an instruction

Anatomy of an instruction

J-type instruction:



Anatomy of an instruction

J-type instruction:



- ▶ **opcode** specifies the operation to execute

Anatomy of an instruction

J-type instruction:



- ▶ **opcode** specifies the operation to execute
 - ▶ E.g. `J` has opcode `000010`

Anatomy of an instruction

J-type instruction:



- ▶ **opcode** specifies the operation to execute
 - ▶ E.g. `J` has opcode `000010`
- ▶ **address** is specified as a 26-bit number

Representing numbers



Powers of 10

Powers of 10

$$10^6 = 1 \underbrace{000000}_{6 \text{ zeroes}}$$

Powers of 10

$$10^6 = 1 \underbrace{000000}_{6 \text{ zeroes}}$$

$$10^1 = 10$$

Powers of 10

$$10^6 = 1 \underbrace{000000}_{6 \text{ zeroes}}$$

$$10^1 = 10$$

$$10^0 = 1$$

Powers of 10

$$10^6 = 1 \underbrace{000000}_{6 \text{ zeroes}}$$

$$10^1 = 10$$

$$10^0 = 1$$

$$10^{-1} = 0.1$$

Powers of 10

$$10^6 = 1 \underbrace{000000}_{6 \text{ zeroes}}$$

$$10^1 = 10$$

$$10^0 = 1$$

$$10^{-1} = 0.1$$

$$10^{-6} = 0.\underbrace{000000}_{5 \text{ zeroes}}1$$

Scientific notation

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers
- ▶ $a \times 10^b$, where

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers
- ▶ $a \times 10^b$, where
 - ▶ a ($1 \leq |a| < 10$) is the **mantissa**

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers
- ▶ $a \times 10^b$, where
 - ▶ a ($1 \leq |a| < 10$) is the **mantissa**
 - ▶ (a is a positive or negative number with a single non-zero digit before the decimal point)

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers
- ▶ $a \times 10^b$, where
 - ▶ a ($1 \leq |a| < 10$) is the **mantissa**
 - ▶ (a is a positive or negative number with a single non-zero digit before the decimal point)
 - ▶ b (an integer) is the **exponent**

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers
- ▶ $a \times 10^b$, where
 - ▶ a ($1 \leq |a| < 10$) is the **mantissa**
 - ▶ (a is a positive or negative number with a single non-zero digit before the decimal point)
 - ▶ b (an integer) is the **exponent**
- ▶ E.g. 1 light year = 9.461×10^{15} metres

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers
- ▶ $a \times 10^b$, where
 - ▶ a ($1 \leq |a| < 10$) is the **mantissa**
 - ▶ (a is a positive or negative number with a single non-zero digit before the decimal point)
 - ▶ b (an integer) is the **exponent**
- ▶ E.g. 1 light year = 9.461×10^{15} metres
- ▶ E.g. Planck's constant = 6.626×10^{-34} joules

Scientific notation

- ▶ A way of writing **very large** and **very small** numbers
- ▶ $a \times 10^b$, where
 - ▶ a ($1 \leq |a| < 10$) is the **mantissa**
 - ▶ (a is a positive or negative number with a single non-zero digit before the decimal point)
 - ▶ b (an integer) is the **exponent**
- ▶ E.g. 1 light year = 9.461×10^{15} metres
- ▶ E.g. Planck's constant = 6.626×10^{-34} joules
- ▶ Socrative FALCOMPED

Scientific notation in code

Scientific notation in code

Instead of writing $\times 10$, write e (no spaces)

Scientific notation in code

Instead of writing $\times 10$, write `e` (no spaces)

```
lightYear = 9.461e15  
plancksConstant = 6.626e-34
```

Floating point numbers

Floating point numbers

- ▶ Similar to scientific notation, but **base 2** (binary)

Floating point numbers

- ▶ Similar to scientific notation, but **base 2** (binary)
- ▶ $\pm \text{mantissa} \times 2^{\text{exponent}}$

Floating point numbers

- ▶ Similar to scientific notation, but **base 2** (binary)
- ▶ $\pm \text{mantissa} \times 2^{\text{exponent}}$
- ▶ Sign is stored as a single bit: 0 = +, 1 = -

Floating point numbers

- ▶ Similar to scientific notation, but **base 2** (binary)
- ▶ $\pm \text{mantissa} \times 2^{\text{exponent}}$
- ▶ Sign is stored as a single bit: 0 = +, 1 = -
- ▶ Mantissa is a binary number with a 1 before the point; only the digits after the point are stored

Floating point numbers

- ▶ Similar to scientific notation, but **base 2** (binary)
- ▶ $\pm \text{mantissa} \times 2^{\text{exponent}}$
- ▶ Sign is stored as a single bit: 0 = +, 1 = -
- ▶ Mantissa is a binary number with a 1 before the point; only the digits after the point are stored
- ▶ Exponent is a signed integer, stored with a **bias**

IEEE 754 floating point formats

Type	Sign	Exponent	Mantissa	Total
Single precision	1 bit	8 bits	23 bits	32 bits
Double precision	1 bit	11 bits	52 bits	64 bits

IEEE 754 floating point formats

Type	Sign	Exponent	Mantissa	Total
Single precision	1 bit	8 bits	23 bits	32 bits
Double precision	1 bit	11 bits	52 bits	64 bits

Exponent is stored with a **bias**:

- ▶ Single precision: store exponent + 127
- ▶ Double precision: store exponent + 1023

IEEE 754 floating point formats

Type	Sign	Exponent	Mantissa	Total
Single precision	1 bit	8 bits	23 bits	32 bits
Double precision	1 bit	11 bits	52 bits	64 bits

Exponent is stored with a **bias**:

- ▶ Single precision: store exponent + 127
- ▶ Double precision: store exponent + 1023
- ▶ Python uses double precision

IEEE 754 floating point formats

Type	Sign	Exponent	Mantissa	Total
Single precision	1 bit	8 bits	23 bits	32 bits
Double precision	1 bit	11 bits	52 bits	64 bits

Exponent is stored with a **bias**:

- ▶ Single precision: store exponent + 127
- ▶ Double precision: store exponent + 1023
- ▶ Python uses double precision
- ▶ Other languages have `float` (single) and `double` types

Example

Example

0 10000001 101000000000000000000000

Example

0 10000001 101000000000000000000000

► Exponent: $129 - 127 = 2$

Example

0 10000001 101000000000000000000000

- ▶ Exponent: $129 - 127 = 2$
- ▶ Mantissa: binary 1.101

Example

0 10000001 101000000000000000000000

- ▶ Exponent: $129 - 127 = 2$
- ▶ Mantissa: binary 1.101
- ▶ $1 + \frac{1}{2} + \frac{1}{8} = 1.625$

Example

0 10000001 101000000000000000000000

- ▶ Exponent: $129 - 127 = 2$
- ▶ Mantissa: binary 1.101
- ▶ $1 + \frac{1}{2} + \frac{1}{8} = 1.625$
- ▶ $1.625 \times 2^2 = 6.5$

Example

0 10000001 101000000000000000000000

- ▶ Exponent: $129 - 127 = 2$
- ▶ Mantissa: binary 1.101
- ▶ $1 + \frac{1}{2} + \frac{1}{8} = 1.625$
- ▶ $1.625 \times 2^2 = 6.5$
- ▶ Alternatively: $1.101 \times 2^2 = 110.1$

Example

0 10000001 101000000000000000000000

- ▶ Exponent: $129 - 127 = 2$
- ▶ Mantissa: binary 1.101
- ▶ $1 + \frac{1}{2} + \frac{1}{8} = 1.625$
- ▶ $1.625 \times 2^2 = 6.5$
- ▶ Alternatively: $1.101 \times 2^2 = 110.1$
- ▶ $= 4 + 2 + \frac{1}{2} = 6.5$

Socratic FALCOMPED

Socrative FALCOMPED

What is the value of this number expressed in IEEE 754 single precision format?

0 01111100 100110000000000000000000

You have **5 minutes**, and you **may** use a calculator!
(Unless your calculator does IEEE 754 conversion...)

Precision of floating point numbers

Precision of floating point numbers

- ▶ Precision **varies** by **magnitude**

Precision of floating point numbers

- ▶ Precision **varies** by **magnitude**
- ▶ Numbers near 0 can be stored more accurately than numbers further from 0

Precision of floating point numbers

- ▶ Precision **varies** by **magnitude**
- ▶ Numbers near 0 can be stored more accurately than numbers further from 0
- ▶ Analogy: in scientific notation with 3 decimal places

Precision of floating point numbers

- ▶ Precision **varies** by **magnitude**
- ▶ Numbers near 0 can be stored more accurately than numbers further from 0
- ▶ Analogy: in scientific notation with 3 decimal places
 - ▶ Around 3.142×10^0 : can represent a difference of 0.001

Precision of floating point numbers

- ▶ Precision **varies** by **magnitude**
- ▶ Numbers near 0 can be stored more accurately than numbers further from 0
- ▶ Analogy: in scientific notation with 3 decimal places
 - ▶ Around 3.142×10^0 : can represent a difference of 0.001
 - ▶ Around 3.142×10^3 : can represent a difference of 1

Precision of floating point numbers

- ▶ Precision **varies** by **magnitude**
- ▶ Numbers near 0 can be stored more accurately than numbers further from 0
- ▶ Analogy: in scientific notation with 3 decimal places
 - ▶ Around 3.142×10^0 : can represent a difference of 0.001
 - ▶ Around 3.142×10^3 : can represent a difference of 1
 - ▶ Around 3.142×10^6 : can represent a difference of 1000

Range of floating point numbers

Type	Smallest value	Largest value
Single precision	$\pm 1.175 \times 10^{-38}$	$\pm 3.403 \times 10^{38}$
Double precision	$\pm 2.225 \times 10^{-308}$	$\pm 1.798 \times 10^{308}$

Rounding errors

Rounding errors

- ▶ Many numbers cannot be represented exactly in IEEE float

Rounding errors

- ▶ Many numbers cannot be represented exactly in IEEE float
 - ▶ Similar to how decimal notation cannot exactly represent $\frac{1}{3} = 0.333333 \dots$ or $\frac{1}{7} = 0.142857 \dots$

Rounding errors

- ▶ Many numbers cannot be represented exactly in IEEE float
 - ▶ Similar to how decimal notation cannot exactly represent $\frac{1}{3} = 0.3333333 \dots$ or $\frac{1}{7} = 0.142857 \dots$
- ▶ Decimal: can represent $\frac{a}{b}$ exactly iff $b = 2^m 5^n$

Rounding errors

- ▶ Many numbers cannot be represented exactly in IEEE float
 - ▶ Similar to how decimal notation cannot exactly represent $\frac{1}{3} = 0.3333333 \dots$ or $\frac{1}{7} = 0.142857 \dots$
- ▶ Decimal: can represent $\frac{a}{b}$ exactly iff $b = 2^m 5^n$
- ▶ Binary: can represent $\frac{a}{b}$ exactly iff $b = 2^n$

Rounding errors

- ▶ Many numbers cannot be represented exactly in IEEE float
 - ▶ Similar to how decimal notation cannot exactly represent $\frac{1}{3} = 0.3333333 \dots$ or $\frac{1}{7} = 0.142857 \dots$
- ▶ Decimal: can represent $\frac{a}{b}$ exactly iff $b = 2^m 5^n$
- ▶ Binary: can represent $\frac{a}{b}$ exactly iff $b = 2^n$
- ▶ In particular, IEEE float can't represent $\frac{1}{10} = 0.1$ exactly!

Rounding errors

- ▶ Many numbers cannot be represented exactly in IEEE float
 - ▶ Similar to how decimal notation cannot exactly represent $\frac{1}{3} = 0.3333333 \dots$ or $\frac{1}{7} = 0.142857 \dots$
- ▶ Decimal: can represent $\frac{a}{b}$ exactly iff $b = 2^m 5^n$
- ▶ Binary: can represent $\frac{a}{b}$ exactly iff $b = 2^n$
- ▶ In particular, IEEE float can't represent $\frac{1}{10} = 0.1$ exactly!
- ▶ This can lead to **rounding errors** with some calculations

Rounding errors

- ▶ Many numbers cannot be represented exactly in IEEE float
 - ▶ Similar to how decimal notation cannot exactly represent $\frac{1}{3} = 0.3333333 \dots$ or $\frac{1}{7} = 0.142857 \dots$
- ▶ Decimal: can represent $\frac{a}{b}$ exactly iff $b = 2^m 5^n$
- ▶ Binary: can represent $\frac{a}{b}$ exactly iff $b = 2^n$
- ▶ In particular, IEEE float can't represent $\frac{1}{10} = 0.1$ exactly!
- ▶ This can lead to **rounding errors** with some calculations
 - ▶ E.g. according to Python, $0.1 + 0.2 - 0.3 = 5.551 \times 10^{-17}$

Testing for equality

Testing for equality

- ▶ Due to rounding errors, using `==` or `!=` with floating point numbers is almost always a bad idea

Testing for equality

- ▶ Due to rounding errors, using `==` or `!=` with floating point numbers is almost always a bad idea
- ▶ E.g. in Python, `0.1 + 0.2 == 0.3` evaluates to `False`

Testing for equality

- ▶ Due to rounding errors, using `==` or `!=` with floating point numbers is almost always a bad idea
- ▶ E.g. in Python, `0.1 + 0.2 == 0.3` evaluates to `False`
- ▶ Better to check for **approximate equality**: calculate the difference between the numbers, and check that it's smaller than some threshold

Testing for equality

- ▶ Due to rounding errors, using `==` or `!=` with floating point numbers is almost always a bad idea
- ▶ E.g. in Python, `0.1 + 0.2 == 0.3` evaluates to `False`
- ▶ Better to check for **approximate equality**: calculate the difference between the numbers, and check that it's smaller than some threshold

```
THRESHOLD = 1e-5
def is_approx_equal(a, b):
    return abs(b - a) < THRESHOLD
```

Decimal types

Decimal types

- ▶ Python (and other languages) provide a `decimal` type

Decimal types

- ▶ Python (and other languages) provide a `decimal` type
- ▶ Uses base 10 rather than base 2, so avoids some of the gotchas with IEEE float

Decimal types

- ▶ Python (and other languages) provide a `decimal` type
- ▶ Uses base 10 rather than base 2, so avoids some of the gotchas with IEEE float
- ▶ ... however not natively supported by the CPU, hence much slower