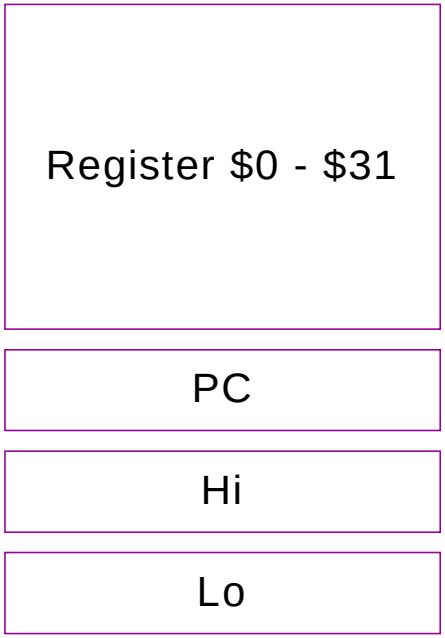
Chapter 2: Instructions: Language of the Machine

- Chapter overview:
  - RISC-V instruction set architecture
    - instruction type
    - instruction format
    - addressing modes
  - Other instruction set architectures

RISC-V Instruction Set Architecture: Registers

- Registers--32 general purpose registers
  - Value in register 0 : 0
  - Each one: 64 bits
- 3 special purpose registers
    - PC: program counter
    - Hi, Lo for multiply and divide
- A remark:
  - register 2: x2 (value) vs 2 (address)
- Byte: 8 bits, word: 32 bits; doubleword: 64 bits



Memory Operands

- Memory is byte addressed
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Instruction address must be a multiple of 4
  - doubleword address must be a multiple of 8

RISC-V ISA: Instruction Categories

- Arithmetic & logic (AL)

```
add x1, x2, x3          //  x1  =  x2 + x3
sub x1, x2, x3          //  x1  =  x2 - x3
```

  – each AL inst. has exactly 3 operands

- Load/store (double word) -- data transfer

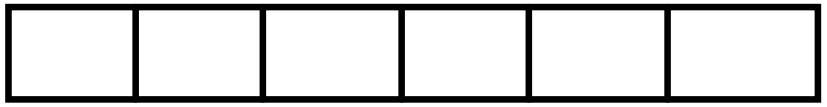
```
ld x1, b(x2)            // x1 = memory [x2+b]
sd x1, b(x2)            // memory[x2+b]=x1
```

- Control purpose--Jump, branch

```
beq x2, x3, L1      //branch to f(L1) when ($2) = ($3)
bne x2, x3, L1      //branch to f(L1) when ($2) ≠ ($3)
jal x5  L1          //goto f(L1), save returning address to x5
```

Representing Inst. in Computer

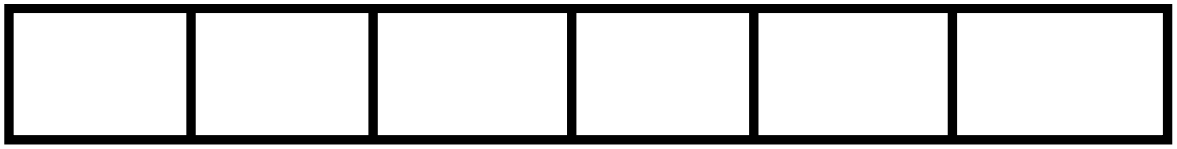
- Instruction format: layout of the instruction
- 5 types of formats:
  R-type (regular)
  I-type (Immediate)
  S-type (sd)
  SB-type (branch)
  UJ-type (jal)



- R-type: 6 fields
  - op: operation code
  - rs1: 1st register source operand
  - rs2: 2nd register source operand
  - rd: register destination
  - funct7: function code
  - funct3: additional function code (for shifting operation)

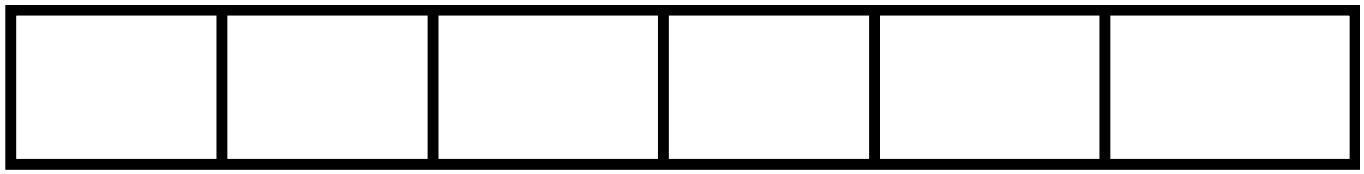
Instruction Format/R-Type

- Example: `add x9, x20, x21`



- Other R-type inst.
  `sub x1, x2, x3`

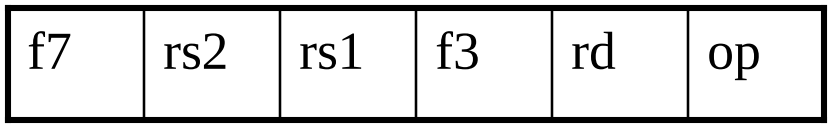
Example



- Sub x9, x10, x11
- Add x11, x9, x10
- Sub x11, x10, x9
- Sub x11, x9, x10

- Which instruction it represents?

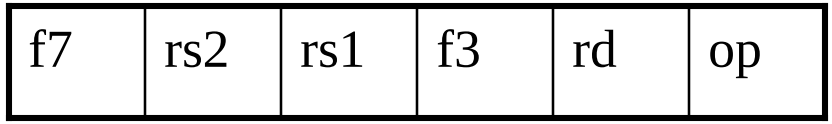
I-type



- Example: load doubleword -- `ld x9, 240(x22)`
  - rs2 is not used: use funct7 + rs2 to store the constant

- Summary
  - 5 fields
  - Immediate field = funct7 + rs2 : 12 bits, holds a constant
- Other I-type inst.
  add immediate -- `addi x8, x9, 100`

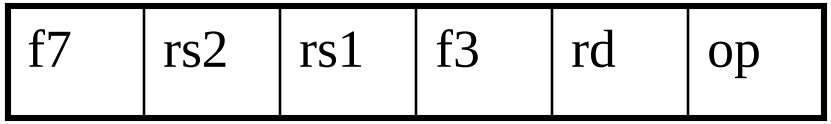
S-type for sd



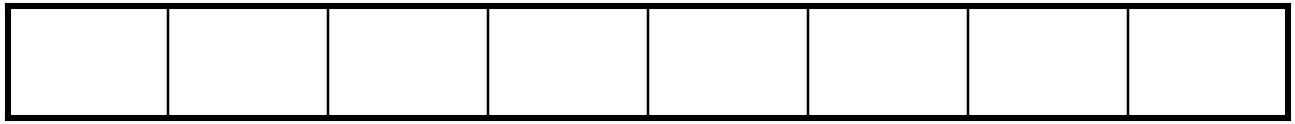
- Example: store doubleword -- `sd x1, 1000(x2)`
  - rd is not used: use funct7 + rd to store the constant

- Summary
  - 5 fields
  - Immediate field =funct7 + rd : 12 bits, holds a constant

SB-type: branches



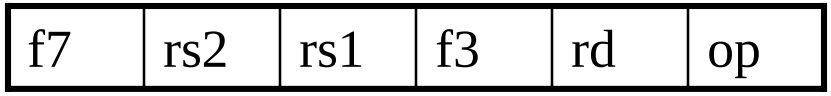
- Example: `bne x10, x11, 2000`
  - rd is not used, use rd+ funct7 to store the constant.



- Unusual encoding
  - Shift the constant to right by 1 to get imm[12:1]
  - Funct7=imm[12]+imm[10:5], rd=imm[4:1]+imm[11]

- Summary
  - 8 fields,
  - funct1 splits to 2 fields: imm[12] & imm[10:5]
  - rd splits to 2 fields: imm[4:1] & imm[11]

UJ-type: jal (jump and link)



- Example: jal x0, 2000
  - rd is used, use f7+rs2+rs1+f3 to store the 20 bits constant.
- Unusual encoding
  - Shift the constant to right by 1 to get imm[20:1]
  - f7+rs2+rs1+f3 =imm[20]+imm[10:1] +imm[11] +imm[19:12]

- Summary:
  - 6 fields: imm[20]+imm[10:1] +imm[11] +imm[19:12]+rd+opcode
  - The constant has 20 bits imm[20:1]

Instruction Format Summary

R	f7	rs2	rs1	f3	rd	opcode
I	imm[11:0]		rs1	f3	rd	opcode
S	imm[11:5]	rs2	rs1	f3	imm[4:0]	opcode
SB	imm[12][10:5]	rs2	rs1	f3	Imm[4:1][11]	opcode
UJ	imm[20][10:1][11][19:12]				rd	opcode

RISC-V Addressing Modes (p.118)

- Register addressing
  - operand is a register; e.g. `add x8, x19, x18`
- Base or displacement addressing
  - operand: at memory location reg. + constant (base)
    e.g. `ld x8, 240(x19)`
- Immediate addressing
  - operand: constant, in inst.; e.g. `addi x8, x8, 8`
- PC-relative addressing
  - branch address = PC + (constant in inst.)
    e.g., `bne x8, x21, 200`

Basic ISA Classes

- Accumulator:
  - 1 address add A acc ← acc + mem[A]
- Stack (special purpose register):
  - 0 address add tos ← tos + next
- General purpose register:
  - 2 address add A, B loc(A) ← loc(A) + loc(B)
  - 3 address add A,B,C loc(A) ← loc(B) + loc(C)
    - memory/memory
    - memory/register
    - register/register --load/store (our RISC-V)

Comparing Number of Inst.

Code sequence for C= A+B for 4 class of instruction sets:

Stack	Accumulator	GP Register Reg.-mem.	GP register load/store

Inst. count depends on the architecture

Evolution of Intel x86 (ch. 2.17)

- 1978: The Intel 8086 is announced (16 bit architecture): <100 instructions
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions (mostly designed for higher performance)
- 1997: 57 new "MMX" instructions are added, Pentium II
- 1999: The Pentium III added another 70 instructions (SSE)
- 2001: Another 144 instructions (SSE2)
- 2003: AMD extends the architecture to increase address space to 64 bits, widens all registers to 64 bits and other changes (AMD64)
- 2004: Intel capitulates and embraces AMD64 (calls it EM64T) and adds more media extensions
- ….
- 2008 … About 900 instructions
- 2011: Vector extension, 128 new instructions were added
- "This history illustrates the impact of the "golden handcuffs" of backward compatibility

  "adding new features as someone might add clothing to a packed bag"

  "an architecture that is difficult to explain and impossible to love"

Fallacies

- Powerful instruction ⇒ higher performance
  - Fewer instructions required in the code
  - But complex instructions are hard to implement
    - May slow down all instructions, including simple ones
  - Compilers are good at making fast code from simple instructions

Example

- A processor with the following CPI. Suppose we can add new powerful AL instructions such that
  - Reduce 25% AL instructions
  - 10% increase of cycle time
- Is this a good choice?

Instruction	CPI	I. count
AL	1	500m
Load/store	10	300m
Branch	3	100m

Chapter Summary

- RISC-V instruction set architecture
  - 3 categories: ALU, data transfer, branch and jump
  - 5 instruction types: R, I, S, SB, UJ
  - 4 addressing modes: register, base, immediate, PC relative,
- Other ISA