RISC V ARCHITECTURE

Rajeshwari B
Department of Electronics and Communication Engineering

PES UNIVERSITY ONLINE

RISC V ARCHITECTURE

Instructions: The Language of Computer –Part-B

Rajeshwari B
Department of Electronics and Communication Engineering

The Rest of the RISC-V Instruction Set

RISC V Additional Instructions of Base Architecture

To make an instruction set architecture suitable for a wide variety of computers, the RISC-V architects partitioned the instruction set into a base architecture and several extensions

Additional Instructions in RISC-V Base Architecture

Instruction	Name	Format	Description
Add upper immediate to PC	`auipc`	U	Add 20-bit upper immediate to PC; write sum to register
Set if less than	`slt`	R	Compare registers; write Boolean result to register
Set if less than, unsigned	`sltu`	R	Compare registers; write Boolean result to register
Set if less than, immediate	`slti`	I	Compare registers; write Boolean result to register
Set if less than immediate, unsigned	`sltiu`	I	Compare registers; write Boolean result to register

The Rest of the RISC-V Instruction Set

AUIPC (Add Upper Immediate to PC)

(AUIPC) adds the 20-bit immediate value to the upper 20 bits of the program counter (pc) and stores the result in the destination register (rd).

AUIPC forms a 32-bit temporary offset, by adding the 20-bit immediate value to the upper 20 bits of temporary offset, filling in the lower 12 bits with zeros.

The temporary offset is added to the pc, to form the pc-relative address. The result is placed in the destination register (rd).

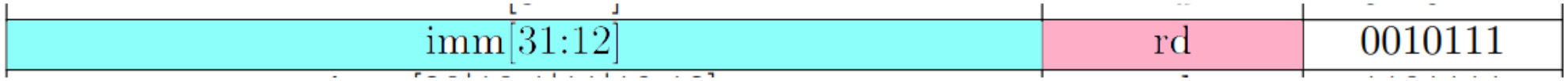
Syntax

auipc rd, imm

where, rd destination register
imm immediate value

The Rest of the RISC-V Instruction Set

Instruction Format



Example:

Assuming pc is at 0x800000ff.

auipc x5, 0x00110

imm = 0x00110

x5 ←– 0x00110000 + 0x800000ff

A.E X5= e 0x801100ff.

The Rest of the RISC-V Instruction Set

SLT(Set Less Than)

Syntax
slt rd, rs1, rs2

Performs the signed comparison between (rs1) and (rs2)
if rs1 < rs2,
rd=1
else rd=0

Example:

```
li x5, 3          # x5 <-- 3
li x3, 5          # x3 <-- 5
slt x1, x5, x3    # x1 <-- x5 < x3
```

x1 will have a value 1.

The Rest of the RISC-V Instruction Set

SLTU (Set Less Than Unsigned)

Syntax

sltu rd, rs1, rs2

Description:Performs the unsigned comparison between (rs1) and (rs2)

if rs1 < rs2,

rd=1

else rd=0

Example:1

```
li x5, 0x80000000          # x5 <-- 0x80000000
li x3, 0xffffffff          # x3 <-- 0xffffffff
slt x1, x5, x3             # x1 <-- x5 < x3
```

x1 will have a value 1.

2) SLTU rd, x0, rs2 sets rd to 1 if rs2 is not equal to zero, otherwise sets rd to zero

SLTI (Set Less than Immediate)

Syntax
slti rd, rs1, imm

Description :A SLTI performs signed comparison of contents of register (rs1) and Immediate data (imm). If the value in register is less than the immediate value, value 1 is stored in destination register, otherwise, value 0 is stored in the destination register.

Example: slti x5, x1, 2
x5=1 if x1<2

SLTIU (Set Less Than Immediate Unsigned)

Syntax
sltiu rd, rs1, imm

Description:does unsigned comparison between register contents (rs1) and Immediate data (imm).
If the value in register is less than the immediate value, the value 1 is stored in destination Register, otherwise, the value 0 is stored in destination register.

Example: **sltiu x5, x1, 2** x5=0 if x1=0xf0000000

sltiu rd, rs1, 1 sets rd to 1 if rs1 equals zero, otherwise sets rd to 0

The Rest of the RISC-V Instruction Set

Example:
For general signed addition, overflow checking requires three additional instructions after the addition to identify that the sum should be less than one of the operands if and only if the other operand is negative.

```
add t0, t1, t2
slti t3, t2, 0
slt t4, t0, t1
bne t3, t4, overflow
```

The Rest of the RISC-V Instruction Set

RISC V Extensions

RISC-V Base and Extensions

Mnemonic	Description	Insn. Count
I	Base architecture	51
M	Integer multiply/divide	13
A	Atomic operations	22
F	Single-precision floating point	30
D	Double-precision floating point	32
C	Compressed instructions	36

The **first**, M, adds instructions to multiply and divide integers.
Next unit will introduce several instructions in the M extension.

The Rest of the RISC-V Instruction Set

RISC V Extensions Contd

The **second** extension, **A**, supports atomic memory operations for multiprocessor synchronization.

The load-reserved (lr.w) and store-conditional (sc.w) instructions introduced earlier are members of the A extension.

The remaining 18 instructions are optimizations of common synchronization patterns, like atomic exchange and atomic addition.

The third and fourth extensions, **F and D**, provide operations on floating-point numbers, which are described in next unit

RISC V Extensions Contd

The last extension, **C,** provides no new functionality at all. Rather, it takes the most popular RISC-V instructions, like addi, and provides equivalent instructions that are only **16 bits in length**, rather than 32.

It thereby allows programs to be expressed in fewer bytes, which can reduce cost.

To fit in 16 bits, the new instructions have restrictions on their operands: for example, some instructions can only access some of the 32 registers, and the immediate fields are narrower.

The Rest of the RISC-V Instruction Set

Fallacies

Powerful instruction ⇒ higher performance

- Fewer instructions required
- 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

Use assembly code for high performance

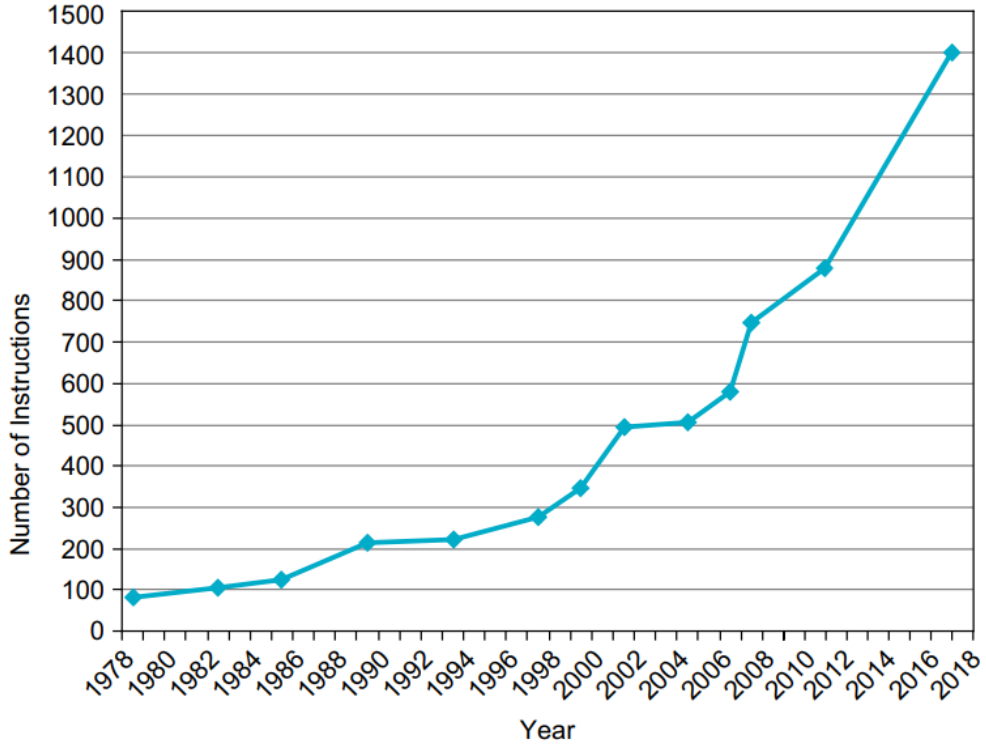
- But modern compilers are better at dealing with modern processors
- More lines of code ⇒ more errors and less productivity
- writing in assembly language are the protracted time spent coding and debugging, the loss in portability, and the difficulty of maintaining such code

The Rest of the RISC-V Instruction Set

Fallacies

Backward compatibility ⇒ instruction set doesn't change

But they do accrete more instructions



Growth of x86 instruction set over time.

Pitfalls

Sequential words are not at sequential addresses

- Increment by 4, not by 1!

Keeping a pointer to an automatic variable after procedure returns

- e.g., passing pointer back via an argument
- Pointer becomes invalid when stack popped

The Rest of the RISC-V Instruction Set

Concluding Remarks

The two principles of the **stored-program computer** are the use of instructions that are indistinguishable from numbers and the use of alterable memory for programs.

Design principles

1. Simplicity favors regularity
2. Smaller is faster
3. Good design demands good compromises

Make the common case fast

Layers of software/hardware

Compiler, assembler, hardware

RISC-V: typical of RISC ISAs compared to x86

THANK YOU

Rajeshwari B
Department of Electronics and Communication Engineering
rajeshwari@pes.edu