Assembly Language & ISA

Sparsh Mittal

Courtesy for some slides: Smruti Ranjan Sarangi

Note: This PPT discusses general concepts. Some of the syntax discussed in this PPT is not applicable to RISC-V.

Levels of Interpretation: Instructions

```
for (i = 0; i < 10; i++)
        printf("go cucs");
```

```
main:   addi x2, x0, 10
        addi x1, x0, 0
loop:   slt x3, x1, x2
        ...
```

10 x2 x0 op=addi

```
00000000101000010000000000010011
00100000000000010000000000010000
00000000001000100001100000101010
```

High Level Language

- C, Java, Python, ADA, …
- Loops, control flow, variables

Assembly Language

No symbols (except labels)

One operation per statement

"human readable machine language"

Machine Language

Binary-encoded assembly

Labels become addresses

The language of the CPU

What is an Instruction Set

- The repertoire of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Simple instruction sets leads to simplified hardware implementation

Designing an ISA

* Important questions that need to be answered :
  * How many instructions should we have ?
  * What should they do ?
  * How complicated should they be ?

Two different paradigms : RISC and CISC

RISC
(Reduced Instruction Set Computer)

CISC
(Complex Instruction Set Computer)

RISC vs CISC

A reduced instruction set computer (RISC) implements simple instructions that have a simple and regular structure. The number of instructions is typically a small number (64 to 128). Examples: ARM, IBM PowerPC, HP PA-RISC, RISC-V

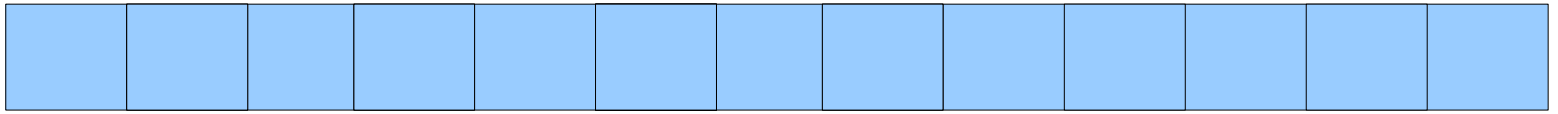
A complex instruction set computer (CISC) implements complex instructions that are highly irregular, take multiple operands, and implement complex functionalities. Secondly, the number of instructions is large (typically 500+). Examples: Intel x86, VAX

Survey of Instruction Sets

ISA	Type	Year	Vendor	Bits	Endianness	Registers
VAX	CISC	1977	DEC	32	little	16
SPARC	RISC	1986	Sun	32	big	32
	RISC	1993	Sun	64	bi	32
PowerPC	RISC	1992	Apple,IBM,Motorola	32	bi	32
	RISC	2002	Apple,IBM	64	bi	32
PA-RISC	RISC	1986	HP	32	big	32
	RISC	1996	HP	64	big	32
m68000	CISC	1979	Motorola	16	big	16
	CISC	1979	Motorola	32	big	16
MIPS	RISC	1981	MIPS	32	bi	32
	RISC	1999	MIPS	64	bi	32
Alpha	RISC	1992	DEC	64	bi	32
x86	CISC	1978	Intel,AMD	16	little	8
	CISC	1985	Intel,AMD	32	little	8
	CISC	2003	Intel,AMD	64	little	16
ARM	RISC	1985	ARM	32	bi(little default)	16
	RISC	2011	ARM	64	bi(little default)	31

Assembly Language

View of Memory



- Memory
  - One large array of bytes
  - Each location has an address
  - The address of the first location is 0, and increases by 1 for each subsequent location
- The program is stored in a part of the memory
- The program counter contains the address of the current instruction

Registers

- Registers → named storage locations
  - in ARM : r0, r1, … r15
  - in x86 : eax, ebx, ecx, edx, esi, edi
  - in RISC-V: x0 to x31
- Registers with special functions :
  - stack pointer
  - program counter
  - return address

Structure of a Statement

* Typically, each assembly statement has two parts: (1) an instruction code that is a mnemonic for a basic machine instruction, and (2) and a list of operands.

- instruction
  - Name of a machine instruction
- operand
  - constant (also known as an immediate)
  - register
  - memory location

Examples of Instructions

```
sub r3, r1, r2
mul r3, r1, r2
```

r3= r1-r2

- subtract the contents of *r2* from the contents of *r1*, and save the result in *r3*
- multiply the contents of *r2* with the contents of *r1*, and save the results in *r3*

Nature of Operands

- Classification of instructions
  - If an instruction takes n operands, then it is said to be in the n-address format
  - Example : add r1, r2, r3 (3 address format)
- Addressing Mode
  - The method of specifying and accessing an operand in an assembly statement is known as the addressing mode.

Understanding Addressing modes AND Instructions that use them

We will discuss 4 addressing modes (relevant for RISC-V)

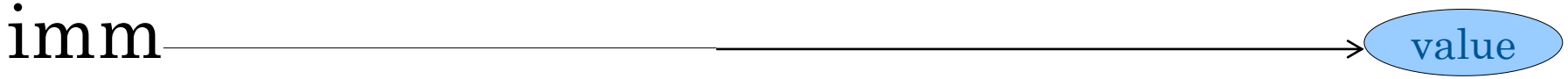
- Immediate
- Register direct
- Register indirect
- Base-offset

- NOTE: Register indirect is a special case of base-offset. If we set offset as zero, we get register indirect mode

1. Immediate addressing mode

- Value ← imm

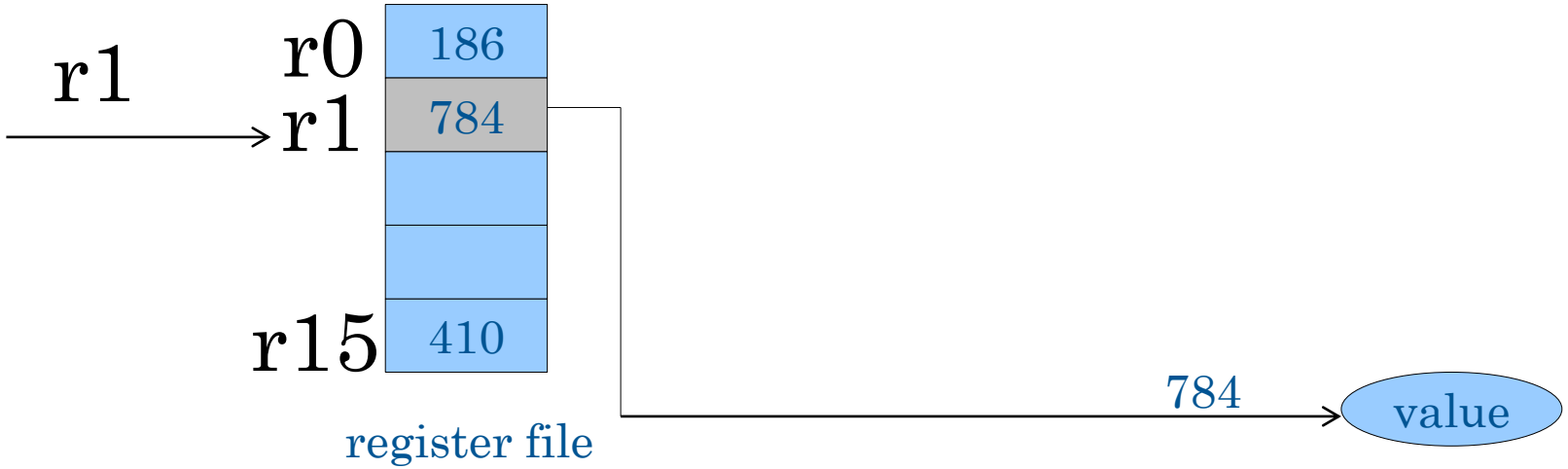
The value (e.g., 4, 8, 0x13, -3 etc) is available in the instruction itself. No need to access register file



2. Register Direct Mode

- Value ← r1

The value is obtained from the register directly.



Examples of instructions that use those modes

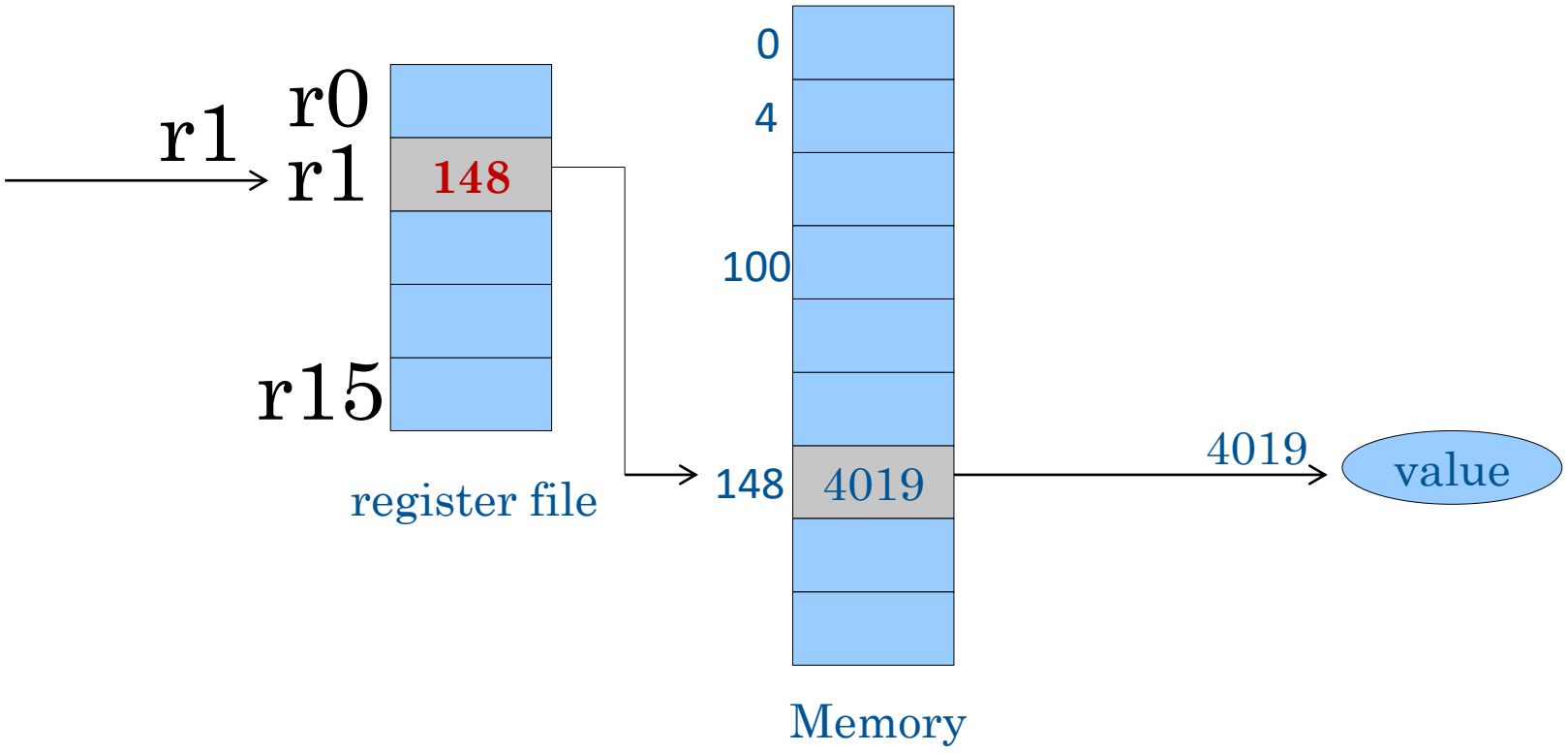
- *Register direct:* `sub r3, r1, r2`
- R1 and r2 values are fetched from registers. Result is stored in r3

- *Immediate:* `sub r3, r1, 500`
- Here, r1 and r3 are accessed from registers and 500 is the immediate value available in the instruction itself.
- RISC-V: at most one operand can be an immediate.

3. Register Indirect Mode

- Value ← (r1)

(1) Read value of r1 from register file. This gives a memory address
(2) Read the value stored in memory at that address

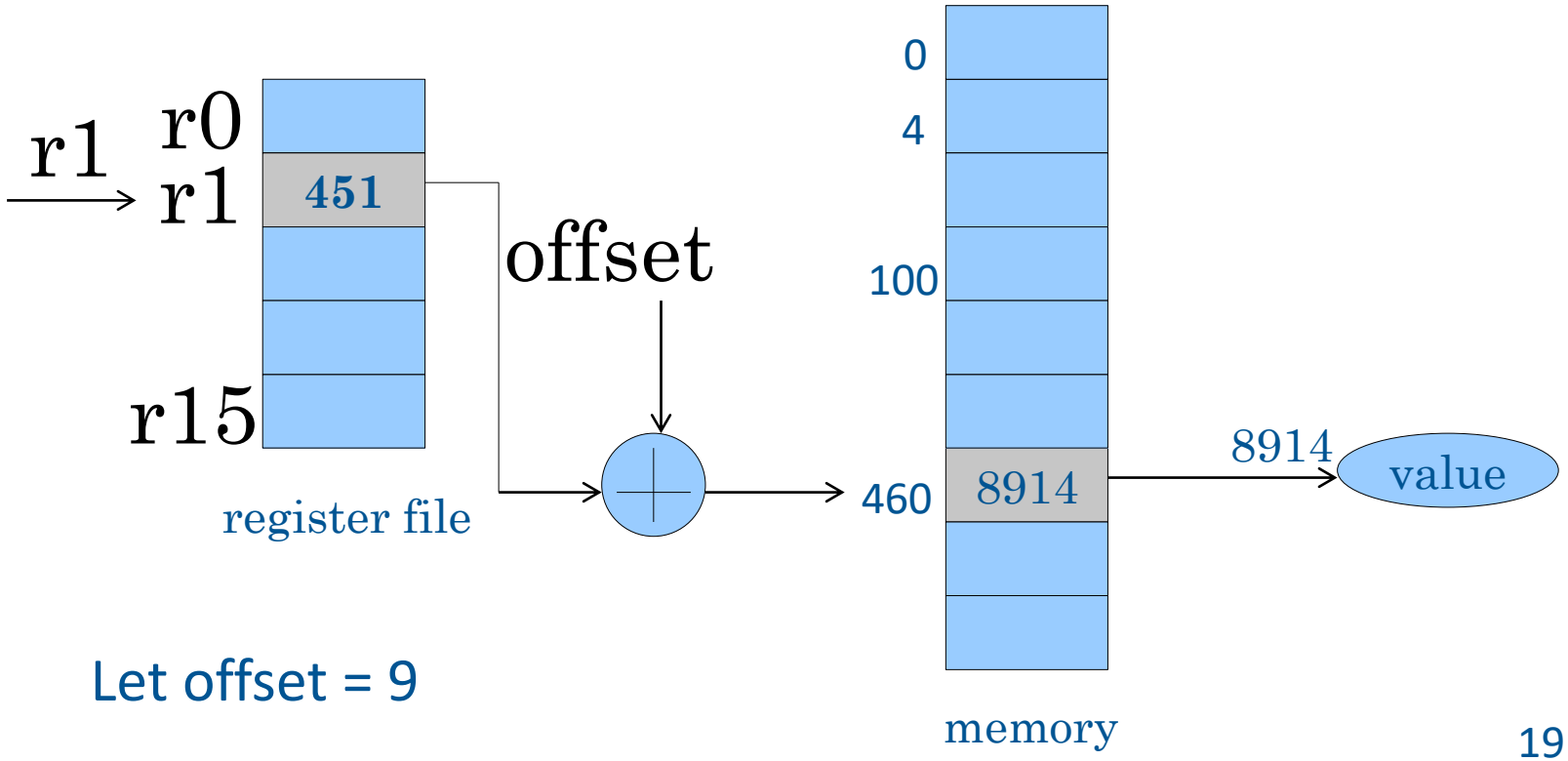


4. Base-offset Addressing Mode

- Value ← offset(r1)

(1) Read value of r1 from register file. Add offset to it. This gives a memory address

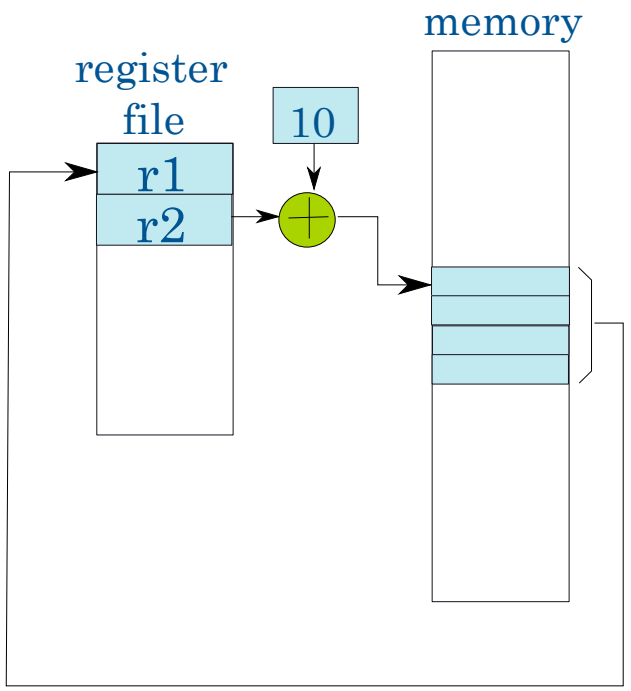
(2) Read the value stored in memory at that address



Examples of instructions that use those modes

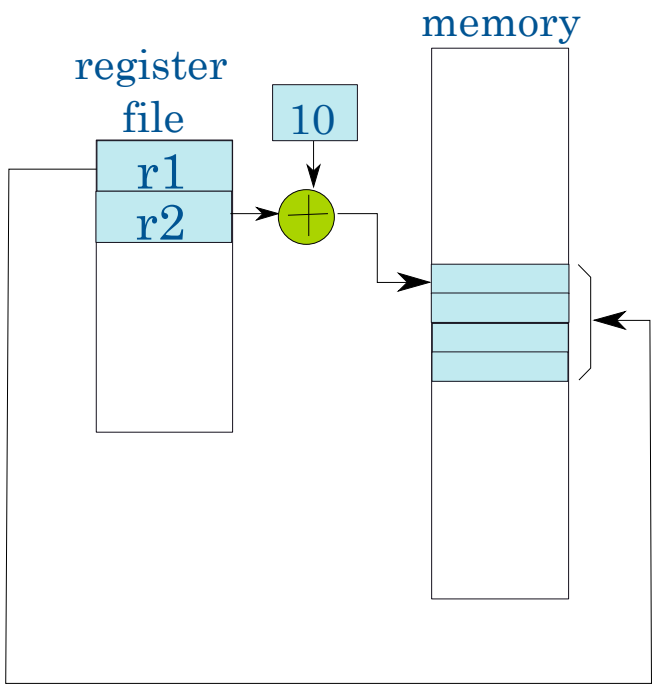
Load and store instructions use register indirect and base-offset addressing modes

lw r1, 10(r2)



(a)

sw r1, 10(r2)



(b)

Lw = load word
Sw= store word

Solved Example

Consider below instructions that are executed when the state of register file and memory is as given here.

ld x6, 24(x10)

sd x5, 16(x10)

Show only the changed values in RF and memory after these instructions are executed.

Register file

Index	Value
x3	29078901
x4	17808987
x5	3897409
x6	89545908
x7	38743091
x8	3331235
x9	1122334455
x10	000001016

Memory

Address	Value
1000	1234567
1008	9876543
1016	1357913
1024	2468024
1032	11335577
1040	67891234
1048	97654789
1056	102938456

Solution: x10+24 is 1040. ld instruction will load from 1040 and store that value in register x6. So, x6 will become 67891234
x10+16 is 1032. sd instruction will store the value of x5 at address 1032. Hence, memory address 1032 will change to 3897409