ISA II

The Instruction Circle:

A sequence of steps or phases

1. FETCH
2. DECODE
3. EVALUATE ADDRESS
4. FETCH OPERANDS
5. EXECUTE
6. STORE RESULT

Not all instructions require the whole six phases

For example:

LDR does not require EXECUTE
ADD does not require EVALUATE ADDRESS

FETCH:

Obtain the instruction from memory and load it into the IR

this phase is common to every instruction type

Complete description:

Step 1: Load the MAR with the contents of the PC, and simultaneously increment the PC (Access)

Step 2: Interrogate memory. This results in the Instruction being placed in the MDR by memory.

Step 3: Load the IR with the contents of the MDR

3 clock circles

DECODE

Indentify the Instruction

Also generate the set of control signals to process the identified instruction in later phases of the IC

Take the decoder as an example:

A 4-to-16 decoder identifies which of the 16 opcodes is going to be processed.

The input is the four bits: IR[15:12]

The remaining 12 bits identify what else is needed to process the instruction.

EVALUATE ADDRESS

Compute the address of the memory location that is needed to process the instruction.

This is necessary in LDR

It computes the address of the data word that is to be read from memory

By adding an offset to the content of a reg.

But not necessary in ADD

LDR calculates the address by adding a reg and an immediate.

FETCH OPERANDS

Obtain the source operands needed to process the instruction.

In LDR:

Step 1: Load MAR with the address calculated in EVALUATE ADDRESS

Step 2: Read memory, placing source operands in MDR

In ADD:

Obtain the source operands from the register file.

In some microprocessors, operand fetch from register file can be done at the same time the instruction is being decoded.

EXECUTE

Execute the instruction

For example:

In ADD, it performs addition in the ALU
In XOR, it performs bitwise XOR in the ALU

STORE RESULT

Write the result to the designated destination (register)

Once STORE RESULT is completed, a new IC (starts with the FETCH phase)

The ISA specifies:

The memory organization
- Address space (MIPS: 2^{32})
- Addressability (MIPS: 8 bits)
  - word/byte addressable

The register set
- 32 registers in MIPS

The instruction set
- Opcodes
- Data types
- Addressing modes
- Length and format of instructions

Opcodes

A large or small set of opcodes could be defined

Tradeoffs are involved

For example:

Hardware complexity vs. software complexity

Latency 延迟 of simple vs. complex instructions

(It depends on the design but it is obvious that complex instruction will take up less address space)

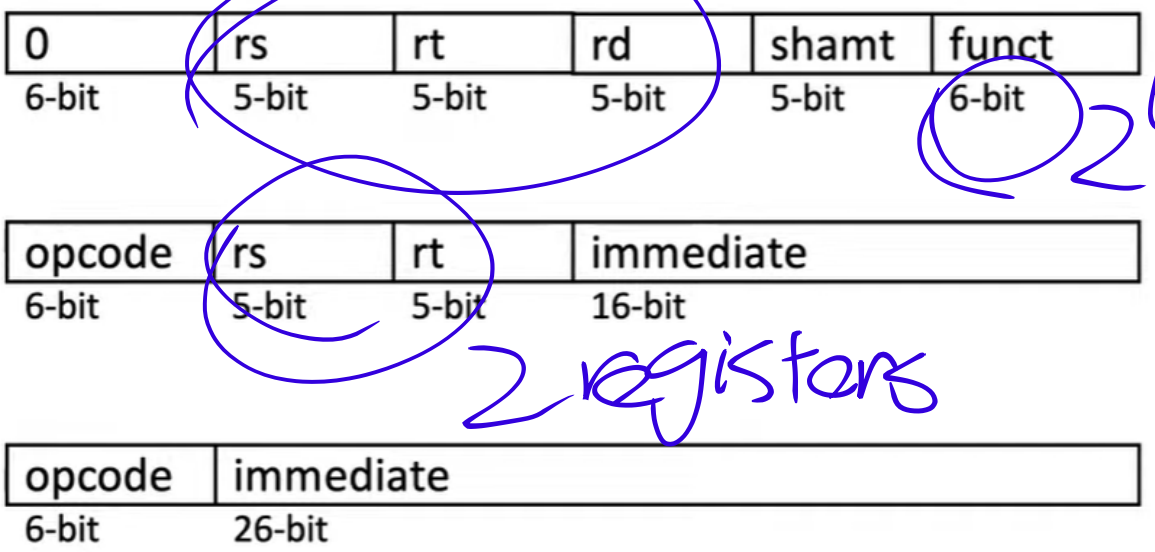
Three types of opcodes:

Operate

Data movement

Control

3 reg = 2 source reg + 1 destination reg



Opcode is 0 in MIPS R-Type instructions.

Funct defines the operation

Table B.2 R-type instructions, sorted by funct field

Funct	Name	Description	Operation
000000 (0)	`sll rd, rt, shamt`	shift left logical	`[rd] = [rt] << shamt`
000010 (2)	`srl rd, rt, shamt`	shift right logical	`[rd] = [rt] >> shamt`
000011 (3)	`sra rd, rt, shamt`	shift right arithmetic	`[rd] = [rt] >>> shamt`
000100 (4)	`sllv rd, rt, rs`	shift left logical variable	`[rd] = [rt] << [rs]4:0`
000110 (6)	`srlv rd, rt, rs`	shift right logical variable	`[rd] = [rt] >> [rs]4:0`
000111 (7)	`srav rd, rt, rs`	shift right arithmetic variable	`[rd] = [rt] >>> [rs]4:0`
001000 (8)	`jr rs`	jump register	`PC = [rs]`
001001 (9)	`jalr rs`	jump and link register	`$ra = PC + 4, PC = [rs]`
001100 (12)	`syscall`	system call	`system call exception`
001101 (13)	`break`	break	`break exception`
010000 (16)	`mfhi rd`	move from hi	`[rd] = [hi]`
010001 (17)	`mthi rs`	move to hi	`[hi] = [rs]`
010010 (18)	`mflo rd`	move from lo	`[rd] = [lo]`
010011 (19)	`mtlo rs`	move to lo	`[lo] = [rs]`
011000 (24)	`mult rs, rt`	multiply	`{[hi], [lo]} = [rs] × [rt]`
011001 (25)	`multu rs, rt`	multiply unsigned	`{[hi], [lo]} = [rs] × [rt]`
011010 (26)	`div rs, rt`	divide	`[lo] = [rs]/[rt], [hi] = [rs]%[rt]`
011011 (27)	`divu rs, rt`	divide unsigned	`[lo] = [rs]/[rt], [hi] = [rs]%[rt]`

(continued)

Table B.2 R-type instructions, sorted by funct field—Cont'd

Funct	Name	Description	Operation
100000 (32)	`add rd, rs, rt`	add	`[rd] = [rs] + [rt]`
100001 (33)	`addu rd, rs, rt`	add unsigned	`[rd] = [rs] + [rt]`
100010 (34)	`sub rd, rs, rt`	subtract	`[rd] = [rs] - [rt]`
100011 (35)	`subu rd, rs, rt`	subtract unsigned	`[rd] = [rs] - [rt]`
100100 (36)	`and rd, rs, rt`	and	`[rd] = [rs] & [rt]`
100101 (37)	`or rd, rs, rt`	or	`[rd] = [rs] | [rt]`
100110 (38)	`xor rd, rs, rt`	xor	`[rd] = [rs] ^ [rt]`
100111 (39)	`nor rd, rs, rt`	nor	`[rd] = ~([rs] | [rt])`
101010 (42)	`slt rd, rs, rt`	set less than	`[rs] < [rt] ? [rd] = 1 : [rd] = 0`
101011 (43)	`sltu rd, rs, rt`	set less than unsigned	`[rs] < [rt] ? [rd] = 1 : [rd] = 0`

Data Types

LC-3 only supports 2's complement integers
二进制补码

while MIPS supports:
- 2's complement integers
- Unsigned integers
- Floating point

Tradeoffs are also involved.

Addressing Modes

Methods for specifying where an operand is located.

There are 5 addressing modes in LC-3

① Immediate or literal (constant)

The operand is in some bits of the Instruction

② Register

The operand is in one of R0 to R7 registers

③ 3 memory addressing modes (+5)

i. PC-relative

ii. Indirect

iii. Base+offset

MIPS has pseudo-direct addressing (for j and jal) additionally, but doesn't have indirect addressing

Operate Instructions

There is no `not` instruction and `nor` instead

There is no `subi` instruction and `addi` instead.

when dealing with high-level code `a = b - 3`

Instructions with one Literal in MIPS

- I-type MIPS Instructions
  - 2 register operands and immediate
- Some operate and data movement instructions

opcode	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

  - opcode = operation
  - rs = source register
  - rt =
    - destination register in some instructions (e.g., `addi`, `lw`)
    - source register in others (e.g., `sw`)
  - imm = Literal or immediate

ADD with one Literal in MIPS

- Add immediate

MIPS assembly

```
addi $s0, $s1, 5
```

Field Values

op	rs	rt	imm
8	17	16	5

rt ← rs + sign-extend(imm)

Machine Code

op	rs	rt	imm
001000	10001	10010	0000 0000 0000 0101

0x22300005