CS2100 - L09 - MIPS : Encoding

Week 5

Instruction Format & Encoding

- 9.1 - R-Format
- 9.2 - I-Format
- 9.3 - J-Format

Summary

- Each MIPS instruction is **fixed-length** 32-bits
  - ➔ All relevant information for an operation must be encoded with these bits!
- Additional challenge:
  - To reduce the complexity of processor design, the instruction encodings should be as regular as possible
    - ➔ Small number of formats, i.e. as few variations as possible

R-format (Register format: `op $r1, $r2, $r3`)

- Instructions which use 2 source registers and 1 destination register
- `e.g. add, sub, and, or, nor, slt, etc`
- **Special cases:** `srl, sll`, etc

I-format (Immediate format: `op $r1, $r2, Immd`)

- Instructions which use 1 source register, 1 immediate value and 1 destination register
- `e.g. addi, andi, ori, slti, lw, sw, beq, bne`, etc

memory branch

J-format (Jump format: `op Immd`)

- `j` instruction uses only one immediate value

R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		
J	opcode	target address				

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
Arithmetic	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Three operands; data in registers
	add immediate	`addi $s1, $s2, 100`	`$s1 = $s2 + 100`	Used to add constants
	load word	`lw $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100	Word from memory to register
	store word	`sw $s1, 100($s2)`	Memory[`$s2` + 100] = $s1	Word from register to memory
Data transfer	load byte	`lb $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100	Byte from memory to register
	store byte	`sb $s1, 100($s2)`	Memory[`$s2` + 100] = $s1	Byte from register to memory
	load upper immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
	branch on equal	`beq $s1, $s2, 25`	if (`$s1 == $s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1, $s2, 25`	if (`$s1 != $s2`) go to PC + 4 + 100	Not equal test; PC-relative
Conditional branch	set on less than	`slt $s1, $s2, $s3`	if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for beq, bne
	set less than immediate	`slti $s1, $s2, 100`	if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0	Compare less than constant
	jump	`j 2500`	go to 10000	Jump to target address
Uncondi-	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
tional jump	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

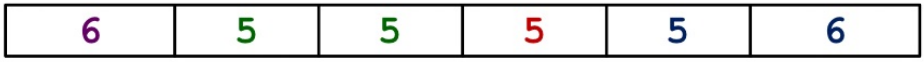
9.1 - R-Format

R-format (op $r1, $r2, $r3)

- Use 2 source registers and 1 destination register

extra space ∵ register values don't take too much space

- Define fields with the following number of bits each:
  - 6 + 5 + 5 + 5 + 5 + 6 = 32 bits



- Each field has a name:

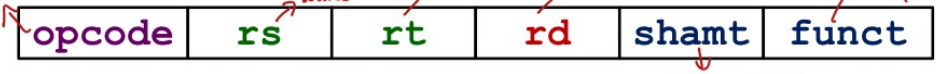
operation code

source

target

destination

expand opcode (distinguish further)



shift amount

- Each field is an independent 5- or 6-bit unsigned integer
  - 5-bit fields can represent any number 0-31
  - 6-bit fields can represent any number 0-63

add : opcode : 0_{10}, funct : 32_{10}

sll : opcode : 0_{10}, funct : 0_{10}

9.2 - I-format

I-format (op $r1, $r2, Immd)

- Use 1 source register, 1 immediate value and 1 destination register

- Define fields with the following number of bits each:
  - 6 + 5 + 5 + 16 = 32 bits

6	5	5	16

- Again, each field has a name:

opcode	rs	rt	immediate

- Only one field is inconsistent with R-format.
  - **opcode**, **rs**, and **rt** are still in the same locations

addi : opcode : 8

lw : opcode : 0x23

program counter (reg that keeps address of instruction in processor)

Branches - PC-relative addressing

- Specify target address relative to PC
- Target address : PC + 16-bit Imm field — signed 2s int
- Interpreted as no. of words — exclusive to branches

  ⇒ Can branch to $\pm 2^{15}$ words ⇒ 2^{17} bytes from PC

- Branch Calculation:

  If the branch is **not taken**:

  PC = PC + 4

  PC + 4 = address of next instruction

  If the branch is **taken**:

  PC = (PC + 4) + (immediate × 4)

9.3 - J-Format

J-format (`op Immd`)

- `j` instruction uses only one immediate value

- Define only two fields:

6 bits	26 bits

- As usual, each field has a name:

Specify exact address

opcode	target address

- Keep `opcode` field identical to **R-format** and **I-format** for consistency
- Combine all other fields to make room for **larger target address**

- **Summary:** Given a **Jump** instruction