University of Nottingham
UK | CHINA | MALAYSIA

MIPS Instruction Sets

Dr. Heng Yu

AY2023-24, Spring Semester
COMP1047: Systems and Architecture
Week 4

Learning Objectives

- Understand MIPS instruction set and design principle
- Translate a MIPS instruction into machine code

Recap: MIPS ISA

MIPS ISA

- We introduced the MIPS architecture.
  - Developed by John Hennessy and his colleagues at Stanford and in the 1980's.
  - Used in many commercial systems, including Silicon Graphics, Nintendo, and Cisco.
- RISC ISA: Reduced instruction set computing
  - As opposed to CISC (e.g. x86)
- Simple, elegant, easy to implement
  - That's why we choose it in COMP1047.
  - Once you've learned one architecture, it's easy to learn others.
- Designed with decades of continuous ISA design efforts
  - The prototype of a lot of modern ISAs

Think about it:

What's the relationship between von Neumann Architecture and MIPS ISA?

Image Link

University of Nottingham UK | CHINA | MALAYSIA

Recap: MIPS ISA

Instruction Set Architecture

- Defines the set of operations that a computer/processor can execute

- The contract between the hardware and software
  - "Contract": given an ISA, your sw and hw must be designed for the ISA! A glue for high and low levels of the system!

- Example ISAs:
  - x86: intel Xeon, intel Core i7/i5/i3, intel atom, AMD Athlon/Opteron, AMD FX, AMD A-series
  - ARM: Apple A-Series, Qualcomm Snapdragon, TIOMAP, NVidia Tegra
  - MIPS: Sony/Toshiba Emotion Engine, MIPS R-4000(PSP)
  - DEC Alpha, PowerPC, IA-64, SPARC and many more …

Assembly Language vs Machine Code

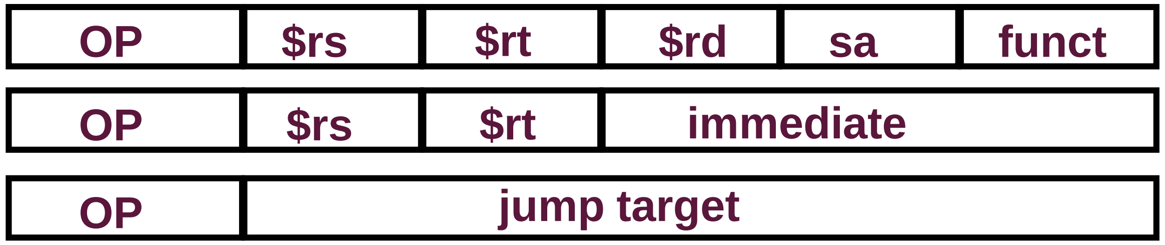
- We've been using assembly language
- Recall that an assembler is a program that translates a symbolic version of instructions into the binary versions
- MIPS32: Each machine instruction is 32-bit long and contains several fields

MIPS instruction	add \$s0, \$s1, \$s2
Hexadecimal (machine code)	$0232\ 8020_{16}$
Binary (machine code)	$0000\ 0010\ 0011\ 0010\ 1000\ 0000\ 0010\ 0000_2$

Instructions as numbers

- Currently we worked with words (32-bit blocks):
  - Each register is a word.
  - `lw` and `sw` both access memory one word at a time.
- So how do we represent instructions?
  - Design principle: Simplicity favors regularity
  - MIPS wants simplicity: since data is in words, make instructions be in words too…
  - Best case: define a single format for all types of instructions – too restrictive.
  - In practice: Defined 3 basic types of instruction formats: R-format, I-format, J-format.
- One instruction is in 32 bits
  - Divide the instruction word into "fields".
  - Each "field" tells computer the info about the operands and operations.

3 Instruction Formats: all 32 bits wide



MIPS ISA as an Example

- All instructions are 32 bits
- 32 32-bit registers
  - $zero is always 0
- 50 opcodes
  - Arithmetic/Logic operations
  - Load/store operations
  - Branch/jump operations
- 3 instruction formats
  - R-type: all operands are registers
  - I-type: one of the operands is an immediate value
  - J-type: non-conditional, nonrelative branches

3 Instruction Formats: all 32 bits wide

OP	$rs	$rt	$rd	sa	funct

OP	$rs	$rt	immediate

OP	jump target

Simplicity favors regularity

R-format instructions

- Define the following fields:

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

  - `opcode`: partially specifies what instruction it is (Note: '0' for all R-Format instructions)
  - `funct`: combined with `opcode` to specify the instruction
    - E.g. add: op(0), funct(32). sub: op(0), funct(34).
  - `rs` (Source Register): used to specify register containing first operand
  - `rt` (S-next Register): used to specify register containing second operand
  - `rd` (Destination Register): used to specify register which will receive result of calculation
  - `shamt` (shift amount): contains the amount a shift instruction will shift by. Set to zero in other cases.

Think about it:

Why are only 5 bits needed for `rs`, `rt`, or `rd`?

Why 5 for `shamt`?

University of Nottingham
UK | CHINA | MALAYSIA

MIPS Instructions

Example

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

R	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$$00000010001100100100000000100000_2 = 02324020_{16}$$

MIPS Instructions

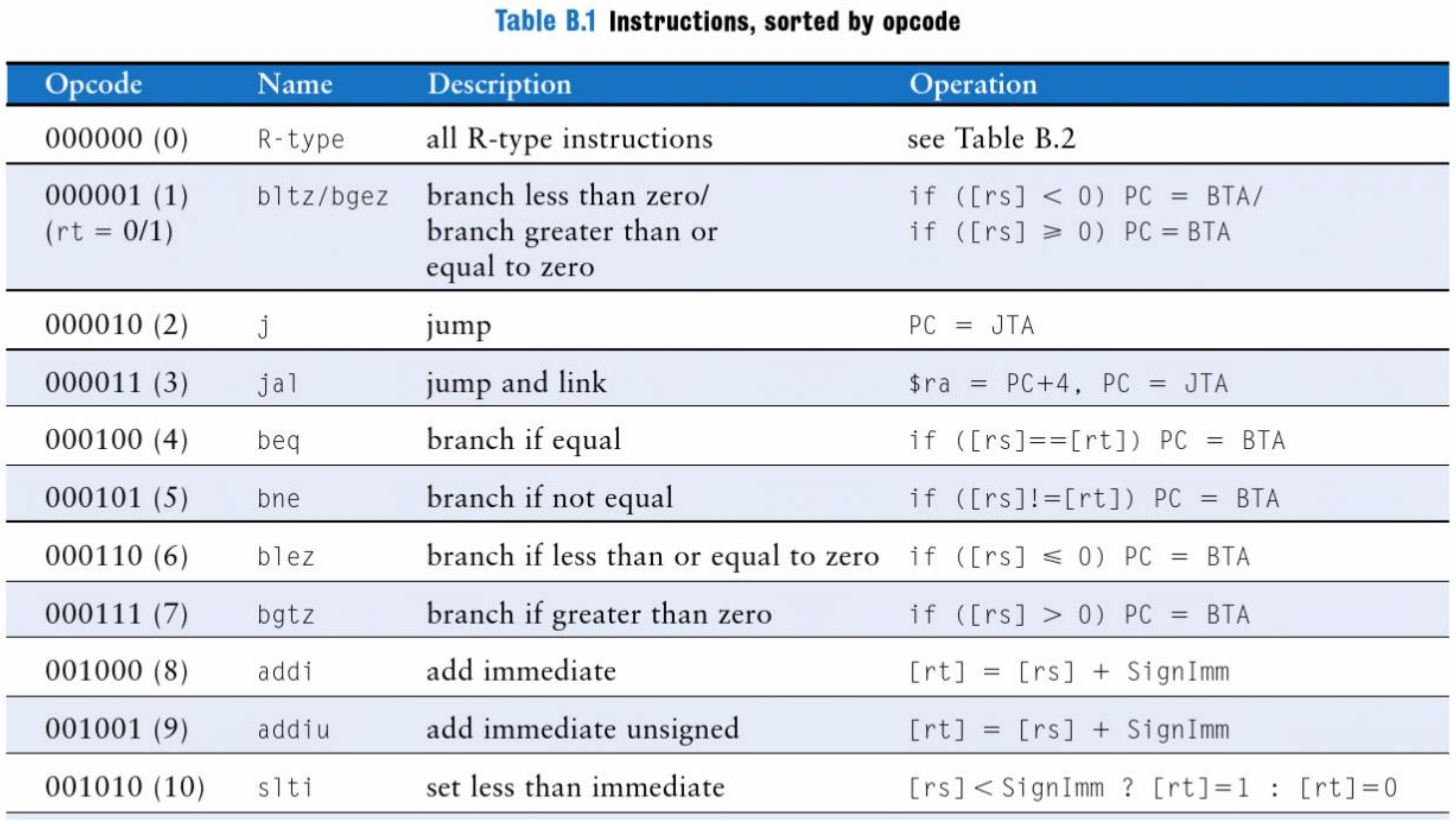


Table B.1 Instructions, sorted by opcode

Opcode	Name	Description	Operation
000000 (0)	R-type	all R-type instructions	see Table B.2
000001 (1) (rt = 0/1)	bltz/bgez	branch less than zero/ branch greater than or equal to zero	if ([rs] < 0) PC = BTA/ if ([rs] ⩾ 0) PC = BTA
000010 (2)	j	jump	PC = JTA
000011 (3)	jal	jump and link	$ra = PC+4, PC = JTA
000100 (4)	beq	branch if equal	if ([rs]==[rt]) PC = BTA
000101 (5)	bne	branch if not equal	if ([rs]!=[rt]) PC = BTA
000110 (6)	blez	branch if less than or equal to zero	if ([rs] ⩽ 0) PC = BTA
000111 (7)	bgtz	branch if greater than zero	if ([rs] > 0) PC = BTA
001000 (8)	addi	add immediate	[rt] = [rs] + SignImm
001001 (9)	addiu	add immediate unsigned	[rt] = [rs] + SignImm
001010 (10)	slti	set less than immediate	[rs] < SignImm ? [rt]=1 : [rt]=0

Table B.1 in Appendix B, in the Harris & Harris Book

MIPS Instructions

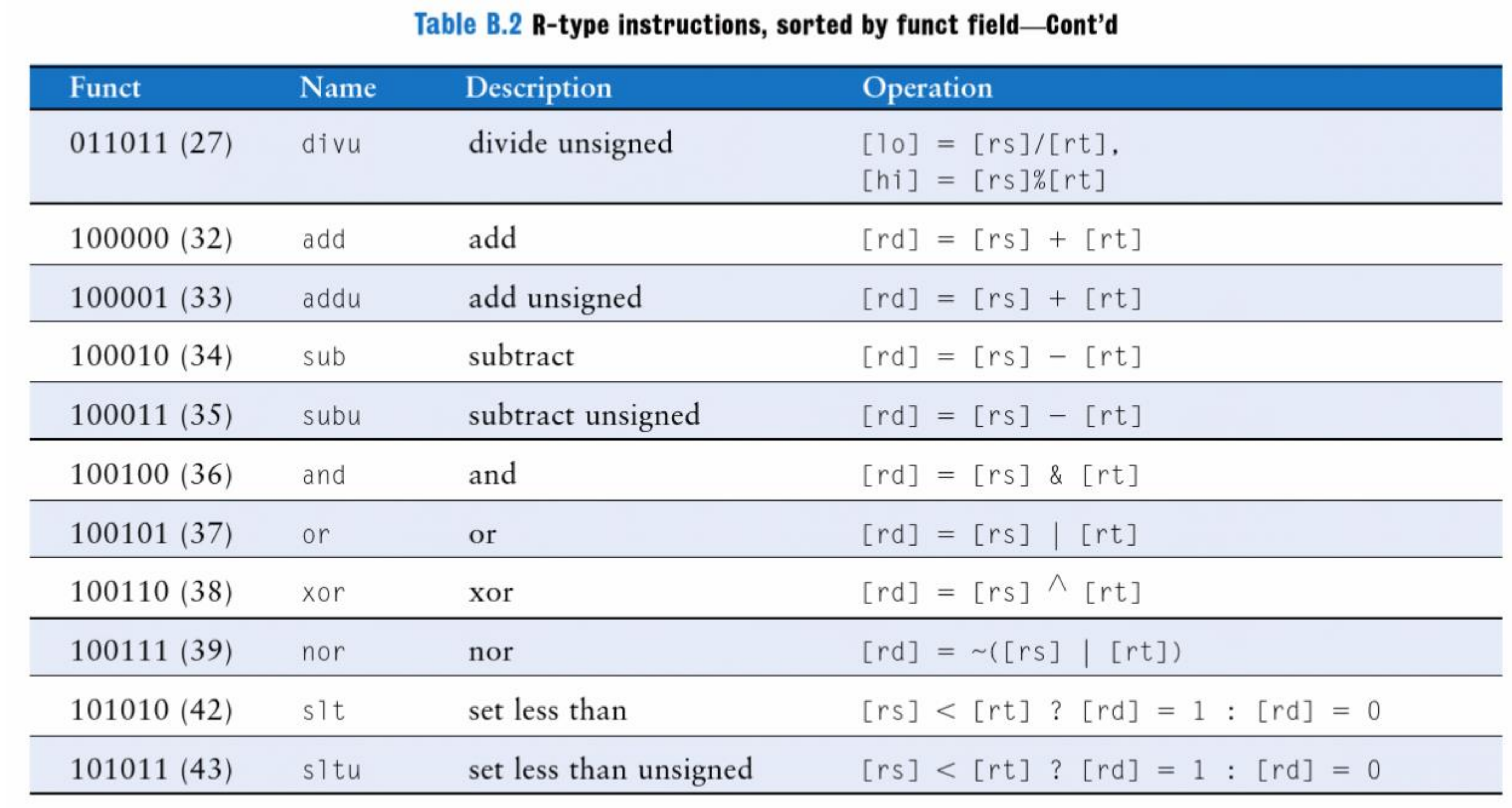


Table B.2 R-type instructions, sorted by funct field—Cont'd

Funct	Name	Description	Operation
011011 (27)	divu	divide unsigned	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
100000 (32)	add	add	[rd] = [rs] + [rt]
100001 (33)	addu	add unsigned	[rd] = [rs] + [rt]
100010 (34)	sub	subtract	[rd] = [rs] − [rt]
100011 (35)	subu	subtract unsigned	[rd] = [rs] − [rt]
100100 (36)	and	and	[rd] = [rs] & [rt]
100101 (37)	or	or	[rd] = [rs] | [rt]
100110 (38)	xor	xor	[rd] = [rs] ^ [rt]
100111 (39)	nor	nor	[rd] = ~([rs] | [rt])
101010 (42)	slt	set less than	[rs] < [rt] ? [rd] = 1 : [rd] = 0
101011 (43)	sltu	set less than unsigned	[rs] < [rt] ? [rd] = 1 : [rd] = 0

Table B.2 in Appendix B, in the Harris & Harris Book

University of Nottingham
UK | CHINA | MALAYSIA

MIPS Instructions

Question: translate the following instruction into machine code (in Hex format)

```
sub $s0, $s1, $s2
```

University of Nottingham UK | CHINA | MALAYSIA

MIPS Instructions

I-format instructions

- Define the following fields:

6	5	5	16
`opcode`	`rs`	`rt`	`immediate`

  - `opcode`: *uniquely* specifies an I-format instruction
  - `rs`: specifies the *only* register operand
  - `rt`: specifies the register which receives the result of calculation (*target register*)
  - `immediate`: 16-bit signed integer, can represent up to 2^{16} different immediate values.

Think about it:

How many different *I-format* instructions can be represented?

What is the maximum number that the `immediate` field can carry?

MIPS Instructions

Example

6	5	5	16
opcode	rs	rt	immediate

```
addi $21, $22, -50
```

decimal representation:

8	22	21	-50

binary representation:

001000	10110	10101	1111111111001110

look up opcode for ‘addi’ in Table B.1 in Appendix B, in the Harris & Harris Book

MIPS Instructions

Example

6	5	5	16
opcode	rs	rt	immediate

```
lw    $t0, 1200($t1)
```

decimal representation:

35	9	8	1200

binary representation:

100011	01001	01000	0000010010110000

look up opcode for 'lw' in Table B.1 in Appendix B, in the Harris & Harris Book

University of Nottingham
UK | CHINA | MALAYSIA

MIPS Instructions

Question: translate the following instruction into machine code (in Hex format)

```
addi $t0,$t1,100
```

MIPS Instructions

Question: translate the following instruction into machine code (in Hex format)

```
addi $21,$22,-50
```

MIPS Instructions

J-format instructions

- Define the following fields:

6	26
opcode	JTA

  - `opcode`: *uniquely* specifies an J-format instruction, so far, we've learnt `j` and `jal`
  - `JTA`: 26-bit jump target address, jump relative to the current PC value, jump in words not in bytes.

Summary

- MIPS instruction set
- {R, I, J}-type instructions

University of Nottingham
UK | CHINA | MALAYSIA

Stay Tuned.