Basic Processor Design – RISC

- All instructions are the same length
  - typically 1 word (i.e. 16 bits)
- Load/Store Architecture
  - All arithmetic and logic instructions deal only with registers and immediate values[1]

    e.g. `ADD R5,R3,R2` R5 ← R3 + R2
    `ORI R5,13` R5 ← R5 | 13
  - Separate instructions are needed for access to locations in memory.

    e.g. `LW R7,13(R4)` R7 ← *mem*(R4 + 13)
    `SW R7,13(R4)` *mem*(R4 + 13) ← R7

    mem(nnn) is shorthand for the data location in memory with address nnn.
  - Instruction set is maximally orthogonal

[1] an immediate (or literal) value is a data value encoded in the instruction word.

Basic Processor Design – RISC

Q1$_{RISC}$
How many Register Addresses in an Arithmetic/Logic Instruction?

- Usually 2 or 3 for RISC

2: `ADD Rx,Ry` Rx ← Rx + Ry

3: `ADD Rz,Rx,Ry` Rz ← Rx + Ry

Q2$_{RISC}$
How many General Purpose Registers?

- Usually $2^n - 1$ for RISC

  This gives 2^n addressable registers including the dummy register, R0[2].

  We then need n bits per register address in the instruction.

  With a 16 bit instruction length, $n = 2$ (i.e. 3 registers + R0) or $n = 3$ (i.e. 7 registers + R0) are sensible values.

[2] R0 is always zero

Basic Processor Design – RISC

Q3$_{RISC}$
How many Bits do we use for Short Immediates?

- Used in instructions like

  `ADDI R2,5` R2 ← R2 + 5
  `SW R7,13(R4)` *mem*(R4 + 13) ← R7

  Sensible values for a 16 bit instruction length are in the range $4 \le s \le 9$
  giving 2's complement values in the range $-2^{s-1} \le imm \le 2^{s-1} - 1$

Q3A$_{RISC}$
Do we support All Arithmetic/Logic instructions and All Load/Store instructions in both Register-Register and Register-Immediate forms?

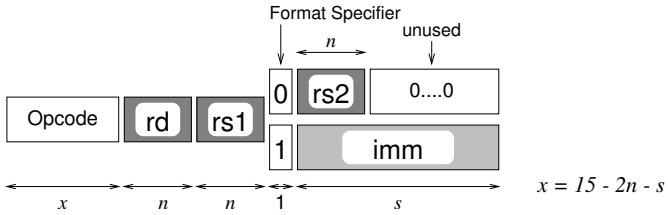
- Some RISC processors support only Register-Immediate form for Load/Store instructions.
- Some RISC processors support Register-Register form for all Arithmetic/Logic instructions and Register-Immediate form for a subset of these instructions.

Basic Processor Design – RISC

Q4$_{RISC}$
What Instruction Fields do we provide? How are they arranged?

- RISC instruction coding is highly orthogonal - any instruction may use any registers.
- Requirement for maximum length short immediate makes RISC coding tight.

Assume Q1$_{RISC}$: 3, Q2$_{RISC}$: $2^n - 1$, Q3$_{RISC}$: s, Q3A$_{RISC}$: YES
A suitable coding for Arithmetic/Logic and Load/Store instructions is:



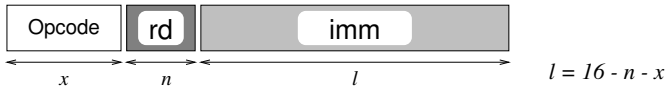
Example: If $n = 2$ and $s = 7$ then $x = 4$ giving up to 2^x (=16) instructions.

Most RISC processors support an instruction to set the upper bits of a register. The MIPS processor calls it LUI (load upper immediate) while the SPARC processor calls it SETHI.
For MIPS, the sequence of instructions required to set upper and lower parts of a register is:

`LUI Rx,200` Rx ← 200 $\times 2^{16}$
`ADDI Rx,5` Rx ← Rx + 5

Note that the $\times 2^{16}$ (i.e. shift left by 16) value comes from the MIPS word length (32) less the length of the long immediate used for LUI (16). In our example it will be $\times 2^{16-l}$ where l is the length of our long immediate.

To code a LUI or SETHI instruction we need fewer fields:

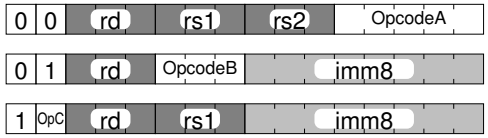


l = 16 - n - x

Example: If $n = 2$ and $x = 4$ then $l = 10$.
Note: $s + l \geq 16$ for the LUI/ADDI sequence to produce a 16 bit result.

Alternative Example Coding #2

An alternative solution is based on the following assumptions:
Q1$_{RISC}$: 3 (or 2 for register-immediate instructions), Q2$_{RISC}$: 7 ($n = 3$),
Q3$_{RISC}$: 8 ($s = 8$ and $l = 8$), Q3A$_{RISC}$: NO



OpcodeA allows for up to 32 Register-Register Arithmetic/Logic instructions.

OpcodeB allows for up to 8 Register-Immediate Arithmetic/Logic instructions.

OpcodeC allows for up to 2 Register-Immediate Load/Store instructions[3].

[3]in this case we know that one of the two will be load and the other will be store

Q5$_{RISC}$
What Instructions will we support?

Type	Function	Mnemonic	Function	Mnemonic
Arithmetic	Add	ADD/ADDI	Subtract	SUB
Logic	Bitwise AND	AND/ANDI	Bitwise OR	OR/ORI
	Bitwise Exclusive OR	XOR	Shift Right Logical	SRL
Data Movement	Load Word	LW	Store Word	SW
	Load Upper Immediate	LUI		
Control Transfer	Branch if Equals Zero	BEZ	Jump and Link	JAL
	Jump and Link Register	JALR		

This set has been chosen based on a maximum of 32+8+2 instructions (to match example coding #2), with support for a complete set of common arithmetic and logical functions (note that multiply is too complex to be included, while shift left is accomplished by adding a number to itself).
The control transfer functions are a minimum set to support subroutines. BEZ provides a conditional PC relative branch (unconditional if R0 is tested). JAL provides a PC relative branch which stores the calling address in a register. JALR provides the ability to return from a subroutine and also the ability to jump a long way.

Q5A$_{RISC}$ Define Assembly Language Syntax[4] and Semantics.

	Mnemonic	Format	Syntax	Semantics
Arithmetic	ADD	A	`ADD Rd,Rs1,Rs2`	$Rd \leftarrow Rs1 + Rs2$
	SUB	A	`SUB Rd,Rs1,Rs2`	$Rd \leftarrow Rs1 - Rs2$
Logic	AND	A	`AND Rd,Rs1,Rs2`	$Rd \leftarrow Rs1$ & $Rs2$
	OR	A	`OR Rd,Rs1,Rs2`	$Rd \leftarrow Rs1$ | $Rs2$
	XOR	A	`XOR Rd,Rs1,Rs2`	$Rd \leftarrow Rs1$ ^ $Rs2$
	SRL	A	`SRL Rd,Rs1`	$Rd \leftarrow Rs1$ >>1
Immediate	ADDI	B	`ADDI Rd,imm8`	$Rd \leftarrow Rd + imm8$
	ANDI	B	`ANDI Rd,imm8`	$Rd \leftarrow Rd$ & $imm8$
	ORI	B	`ORI Rd,imm8`	$Rd \leftarrow Rd$ | $imm8$
	LUI	B	`LUI Rd,imm8`	$Rd \leftarrow imm8$ <<8
Data Movement	LW	C	`LW Rd,imm8(Rs1)`	$Rd \leftarrow mem(Rs1 + imm8)$
	SW	C	`SW Rd,imm8(Rs1)`	$mem(Rs1 + imm8) \leftarrow Rd$
Control Transfer	BEZ	B	`BEZ Rd,imm8`	$if\ (Rd = 0)\ then\ PC \leftarrow PC + imm8$
	JAL	B	`JAL Rd,imm8`	$Rd \leftarrow PC$; $PC \leftarrow PC + imm8$
	JALR	A	`JALR Rd,Rs1+Rs2`	$Rd \leftarrow PC$; $PC \leftarrow Rs1 + Rs2$

[4]operand order follows MIPS convention

*no flags for this processor

Q5B$_{RISC}$
What Opcodes will be Assigned?

Format A

OpA[4:3] \ OpA[2:0]	000	001	011	010	100	101	111	110
00	ADD	OR	–	JALR	–	–	–	–
01	–	AND	–	–	–	–	–	–
11	SUB	XOR	–	–	–	–	–	–
10	–	LSR	–	–	–	–	–	–

Format B

OpB[2] \ OpB[1:0]	00	01	11	10
0	ADDI	ORI	BEZ	JAL
1	LUI	ANDI	–	–

Format C

OpC[1:0]	0	1
	LW	SW

Instructions are grouped so that decoding is simple (this set show has plenty of room for expansion).