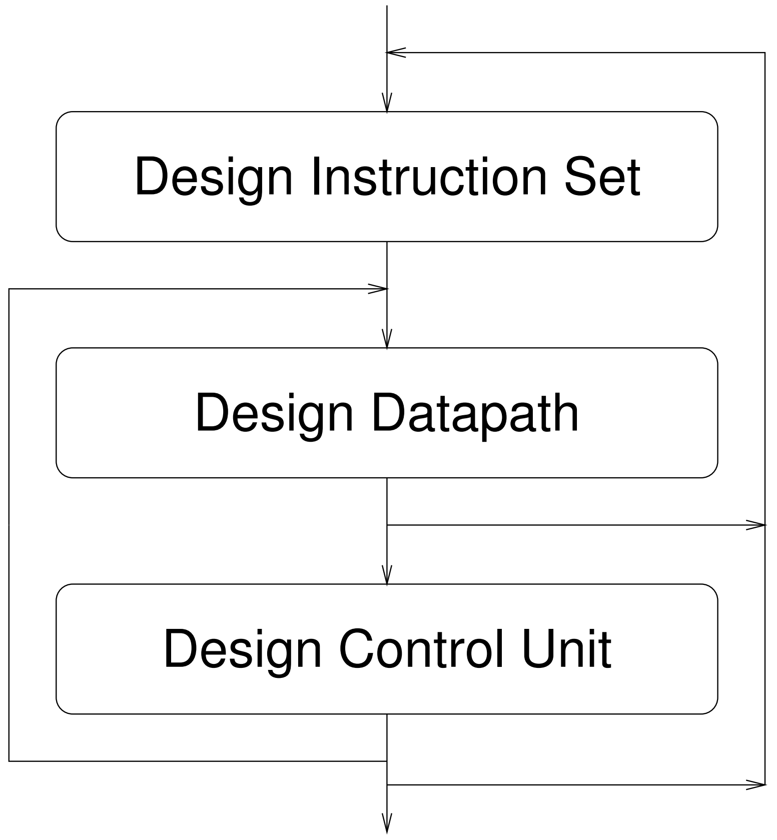
Basic Processor Design



This lecture deals with Instruction Set Design.

Instruction Set Terminology

Mnemonic (Instruction Name)

SUBI

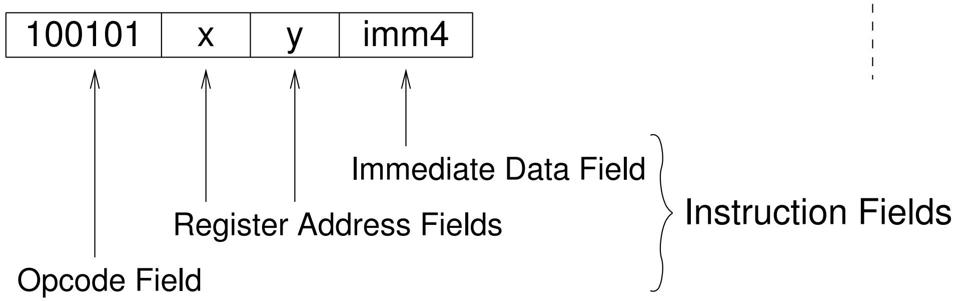
Syntax (Language Structure) *Assembly Language*

`SUBI Rx, Ry, imm4`

Semantics (Meaning) *Register Transfer Language*

$Rx \longleftarrow Ry - imm4$

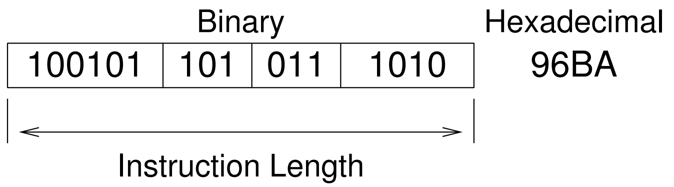
Coding *Machine Code*



Example

`SUBI R5, R3, 10`

$R5 \longleftarrow R3 - 10$



The *instruction set* for a processor is the set of all the instructions supported by that processor.

Basic Processor Design

Start with RISC[1] assumptions since we want a simple processor.

- Fixed instruction length
  - keeps control simple (very useful for pipelining[2])
- Load/store architecture & a large number of general purpose registers
  - avoids slow memory access
- Very few addressing modes
  - complex addressing modes are seldom used
- No complex instructions
  - keep everything simple

[1]RISC = Reduced Instruction Set Computer

[2]RISC machines were initially designed to increase performance through pipelining - your processor will not use pipelining

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[3]

    e.g. `ADD R3,R2,R5` $R5 \leftarrow R3 + R2$

    `OR R3,13,R5` $R5 \leftarrow R3 \mid 13$
  - Separate instructions are needed for access to locations in memory.

    e.g. `LD [R4+13],R7` $R7 \leftarrow \textit{mem}(R4 + 13)$

    `ST R7,[R4+13]` $\textit{mem}(R4 + 13) \leftarrow R7$

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

[3] 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 Rx,Ry,Rz`	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[4].

  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.

[4]R0 is always zero

Basic Processor Design – RISC

Q3_{RISC}
How many Bits do we use for Short Immediates?

- Used in instructions like

`ADD R3,5,R2`	R2 $\leftarrow$ R3 + 5
`ST R7,[R4+13]`	*mem*(R4 + 13) $\leftarrow$ R7

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

Q3A_{RISC}
Do we support All Arithmetic/Logic instructions and All Load/Store instructions in both Register-Register and Register-Immediate forms?

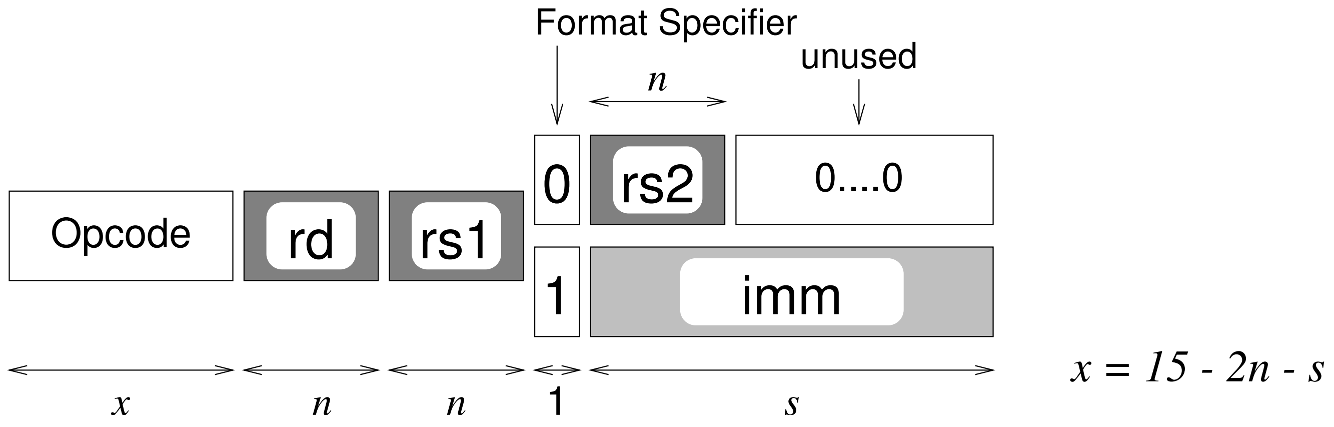
- Some RISC processors support only Register-Immediate form for Load/Store instructions.
- Some RISC processors support Register-Register form for all Arithmetic/Logic instructions and Register-Immediate form for a subset of these instructions.

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.

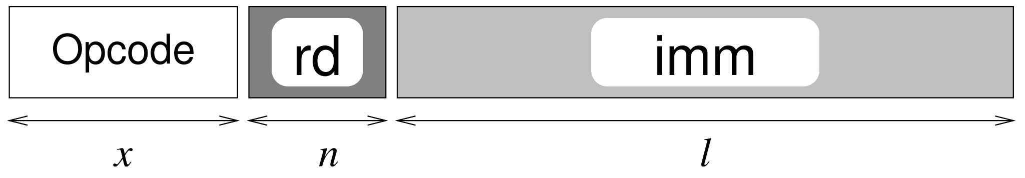
Basic Processor Design – RISC

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 SPARC, the sequence of instructions required to set upper and lower parts of a register is:

`SETHI 200,Rx`	$Rx \leftarrow 200 \times 2^{10}$
`ADD Rx,5,Rx`	$Rx \leftarrow Rx + 5$

Note that the $\times 2^{10}$ (i.e. shift left by 10) value comes from the SPARC word length (32) less the length of the long immediate used for SETHI (22). In our example it will be $\times 2^{16-l}$ where l is the length of our long immediate.

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

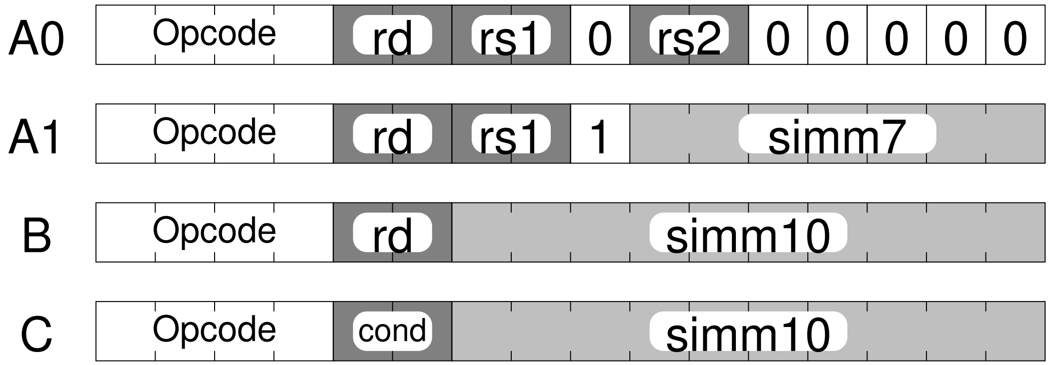


$l = 16 - n - x$

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

Example Coding #3

Assume Q1$_{RISC}$: 3, Q2$_{RISC}$: 3 ($n = 2$), Q3$_{RISC}$: 7 ($s = 7$), Q3A$_{RISC}$: YES
This gives $x = 4$ and $l = 10$ with the coding shown below[5]:



In this case up to 16 opcodes are supported, each of which supports either coding A (i.e. A0 and A1), coding B or coding C (which supports conditional branch).

This is just one of many possible codings (this one is loosely based on the SPARC instruction coding).

[5]note that in this example short and long immediates (`simm7` and `simm10`) are signed numbers

Basic Processor Design – RISC

$Q5_{RISC}$
What Instructions will we support?[6]

Type	Function	Mnemonic	Function	Mnemonic
Arithmetic	Add	ADD	Subtract	SUB
	Add with Carry	ADDX	Subtract with Borrow	SUBX
Logic	Bitwise AND	AND	Bitwise OR	OR
	Bitwise Exclusive OR	XOR	Logical Shift Right	LSR
Data Movement	Load	LD	Store	ST
	Set High	SETHI		

- 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).

- all arithmetic and logic functions update the condition flags:

  Zero (Z) is set if the result is zero and cleared otherwise.

  Carry (C) is updated by ADD/ADDX/SUB/SUBX/LSR and is set to zero by other logical instructions.

[6]This set has been chosen to match example coding #3

Basic Processor Design – RISC

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

Type	Function	Mnemonic	SubFunction	Mnemonic
Control Transfer	Branch if equal (Z==1)	BEQ	Branch if not equal (Z==0)	BNE
	Branch if carry set (C==1)	BCS	Branch if carry clear (C==0)	BCC
	Branch and Link	BAL	Jump and Link	JMPL

- This processor (like the SPARC on which it is based) uses condition code flags (C,Z) to support conditional branch[7].
- Branch instructions are PC relative with a limited range (+511/-512).
- BAL and JMPL support subroutine call by saving the old PC value in a link register. If R0 (dummy register) is specified as the link register, then BAL can be used as a simple unconditional PC relative branch while JMPL can be used for jumping a long way or for returning from a subroutine.

[7]other RISC processors such as MIPS save some complexity by not supporting condition code flags

Basic Processor Design – RISC

Q5A$_{RISC}$ Define Assembly Language Syntax[8] and Semantics.

	Mnemonic	Format	Syntax	Semantics
Arithmetic	ADD*	A	`ADD Rs1,Op2,Rd`	$Rd \leftarrow Rs1 + Op2$
			where `Op2` is either `Rs2` or `simm7`	
	SUB*	A	`SUB Rs1,Op2,Rd`	$Rd \leftarrow Rs1 - Op2$
	ADDX*	A	`ADDX Rs1,Op2,Rd`	$Rd \leftarrow Rs1 + Op2 + C$
	SUBX*	A	`SUBX Rs1,Op2,Rd`	$Rd \leftarrow Rs1 - Op2 - C$
Logic	AND*	A	`AND Rs1,Op2,Rd`	$Rd \leftarrow Rs1$ `&` $Op2$
	OR*	A	`OR Rs1,Op2,Rd`	$Rd \leftarrow Rs1$ `|` $Op2$
	XOR*	A	`XOR Rs1,Op2,Rd`	$Rd \leftarrow Rs1$ `^` $Op2$
	LSR*	A	`LSR Rs1,Rd`	$Rd \leftarrow Rs1$ `>>1`
Data Movement	LD	A	`LD [Rs1+Op2],Rd`	$Rd \leftarrow mem(Rs1 + Op2)$
	ST	A	`ST Rd,[Rs1+Op2]`	$mem(Rs1 + Op2) \leftarrow Rd$
	SETHI	B	`SETHI simm10,Rd`	$Rd \leftarrow simm10$ `<<6`
Control Transfer	BEQ	C	`BEQ simm10`	$if\ (Z == 1)\ then\ PC \leftarrow PC + simm10$
	BNE	C	`BNE simm10`	$if\ (Z == 0)\ then\ PC \leftarrow PC + simm10$
	BCS	C	`BCS simm10`	$if\ (C == 1)\ then\ PC \leftarrow PC + simm10$
	BCC	C	`BCC simm10`	$if\ (C == 0)\ then\ PC \leftarrow PC + simm10$
	BAL	B	`BAL simm10,Rd`	$Rd \leftarrow PC; PC \leftarrow PC + simm10$
	JMPL	A	`JMPL Rs1+Op2,Rd`	$Rd \leftarrow PC; PC \leftarrow Rs1 + Op2$

[8]operand order follows SPARC convention

*marked commands update the flags (C,Z)

Basic Processor Design – RISC

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

Op[3:2] \ Op[1:0]	00	01	11	10
00	ADD	ADDX	AND	OR
01	SUB	SUBX	XOR	LSR
11	LD	ST	–	JMPL
10	SETHI	–	B...	BAL

Format A

Format B/C

Cond[1] \ Cond[0]	0	1
0	BEQ	BNE
1	BCS	BCC

Instructions are grouped so that decoding is simple.

Unfortunately this example processor does not meet your specification...

Basic Processor Design

Consider CISC[9] features that might be included in the design.

- Variable instruction length
  - allows full 16-bit values to be included in your instructions
- Abilitity to act on data in memory
  - gives reduced instruction count
- Registers with specific roles
  - e.g. 8 registers but only 2 may act as address registers for load and store - leads to shorter register address fields
- More addressing modes
  - e.g. store with pre-decrement or load with post-increment
- More complex instructions
  - e.g. `CALL` and `RETURN` instuctions which store and retrieve the return address on the stack

[9]CISC = Complex Instruction Set Computer