CSC 411

Computer Organization (Spring 2023)
Lecture 10: Conditionals and loops

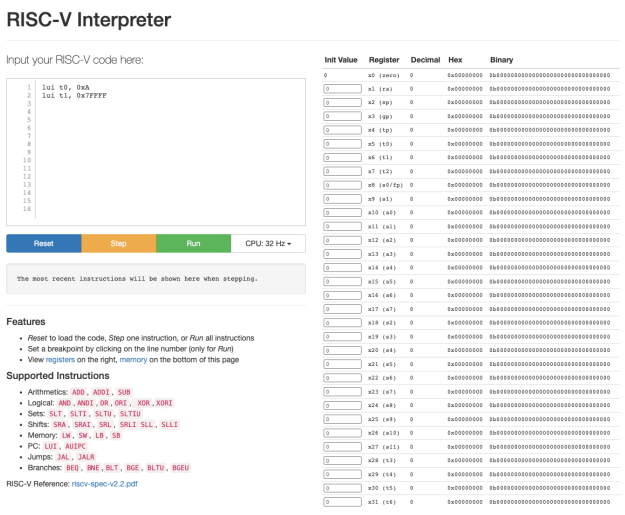
Prof. Marco Alvarez, University of Rhode Island

Example

- What is the value of x9?

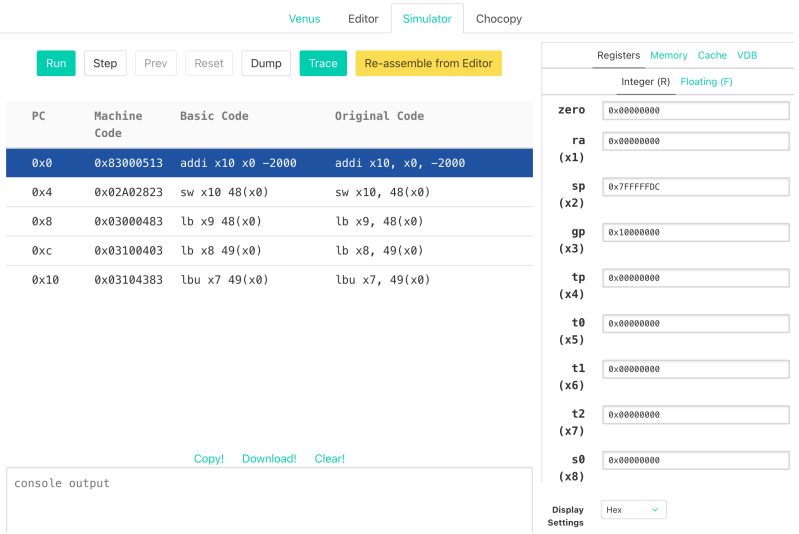
```
addi x10, x0, -2000
sw x10, 48(x0)
lb x9, 48(x0)

lb x8, 49(x0)
lbu x7, 49(x0)
```

RISC-V interpreter



https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/

Venus simulator



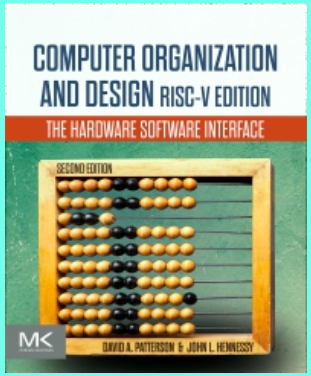
https://venus.cs61c.org/

Disclaimer

Some of the following slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)

The Hardware/Software Interface



More instructions

Character data

- Byte encoded character sets
  - **ASCII**: 128 characters (95 graphic, 33 control)
  - **Latin-1**: 256 characters (ASCII, +96 more graphic characters)
- **Unicode**: 32-bit character set
  - used in Java, C++ wide characters, …
  - most of the world's alphabets, plus symbols
  - UTF-8, UTF-16, UTF-32: variable-length encodings

ASCII representation

ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

Byte/halfword/word operations

- Load byte/halfword/word (sign extend to 64 bits in **rd**)
  - `lb rd, offset(rs1)`
  - `lh rd, offset(rs1)`
  - `lw rd, offset(rs1)`
- Load byte/halfword/word unsigned (zero extend to 64 bits in **rd**)
  - `lbu rd, offset(rs1)`
  - `lhu rd, offset(rs1)`
  - `lwu rd, offset(rs1)`
- Store byte/halfword/word (store rightmost 8/16/32 bits)
  - `sb rs2, offset(rs1)`
  - `sh rs2, offset(rs1)`
  - `sw rs2, offset(rs1)`

32-bit constants

- Most constants are small
  - 12-bit immediate is sufficient
- For the occasional 32-bit constant

```
lui <rd>, <constant>
```

  - copies 20-bit constant to bits [31:12] of **<rd>**
  - clears bits [11:0] of **<rd>** to 0
- Example:

```
lui x19, 976  // 0x003D0
addi x19,x19,1280  // 0x500
```

Logical operations

- Instructions for bitwise manipulation

Logical operations	C operators	Java operators	RISC-V instructions
Shift left	<<	<<	sll, slli
Shift right	>>	>>>	srl, srli
Shift right arithmetic	>>	>>	sra, srai
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit XOR	^	^	xor, xori
Bit-by-bit NOT	~	~	xori

```
xori x5, x6, -1
```

Branches

- A branch is a change of control flow
  - conditional branch — change control according to a condition
    - `beq, bne, blt, bge, bltu, bgeu`
  - unconditional branch — change control unconditionally
    - `j` (jump)

Branch instructions

Conditional branch	Branch if equal	`beq x5, x6, 100`	`if (x5 == x6) go to PC+100`	PC-relative branch if registers equal
	Branch if not equal	`bne x5, x6, 100`	`if (x5 != x6) go to PC+100`	PC-relative branch if registers not equal
	Branch if less than	`blt x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less
	Branch if greater or equal	`bge x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal
	Branch if less, unsigned	`bltu x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less, unsigned
	Branch if greater or equal, unsigned	`bgeu x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal, unsigned
Unconditional branch	Jump and link	`jal x1, 100`	`x1 = PC+4; go to PC+100`	PC-relative procedure call
	Jump and link register	`jalr x1, 100(x5)`	`x1 = PC+4; go to x5+100`	Procedure return; indirect call

Conditional operations

- Jump/branch to a labeled instruction if a condition is `true`
  - otherwise, continue sequentially

```
// if equal, jump to label L1
beq <rs1>, <rs2>, L1

// if not equal, jump to label L1
bne <rs1>, <rs2>, L1
```

Example (if)

```
// assume f, g, h, i, j are in
// x19, x20, ...
if (i == j) {
    f = g + h;
} else {
    f = g - h;
}
```

```
main:
    // ... instructions
    bne x22, x23, label1
    add x19, x20, x21
    beq x0, x0, label2
label1:
    sub x19, x20, x21
label2:
    // ... instructions
```

Example (loop)

```
// assume i in x22, k in x24
// base address of save in x25
while (save[i] == k) {
    i += 1;
}
```

```
main:
    // ... instructions
label3:
    slli x10, x22, 2
    add  x10, x10, x25
    lw   x9, 0(x10)
    bne  x9, x24, label4
    addi x22, x22, 1
    beq  x0, x0, label3
label4:
    // ... instructions
```

More conditional operations

- Branch to instruction if condition is `true`

```
// branch to Label if rs1 less than rs2
blt <rs1>, <rs2>, Label
// branch to Label if rs1 greater or equal than rs2
bge <rs1>, <rs2>, Label


// example if (a > b) a += 1;
    // ... instructions
    bge  x23, x22, exit
    addi x22, x22, 1
exit:
```

Signed vs unsigned

- Signed comparison
  - `blt`, `bge`
- Unsigned comparison
  - `bltu`, `bgeu`
- Example

```
// assume x22 stores 0xFFFFFFFF
// assume x23 stores 0x00000001
// which instruction branches?
blt  x22, x23, Label
bltu x22, x23, Label
```

Examples

Example

```
// assume t0 holds the value 0x00101000
// what is the value of t2?
//
        addi t2, zero, 10
        blt x0, t0, if
        beq x0, x0, done
if:
        addi t2, t2, 2
done:
        addi t2, t2, 1
```

Example

```
// assume t1 holds the value 10 and s2
// is zero, what is the value of s2?
loop:
        bge zero, t1, done
        addi t1, t1, -1
        addi s2, s2, 2
        beq zero, zero, loop
done:
```

Example

```
// assume a, b, c, d
// are in s1, s2, s3, s4
// base address of data in t0
do {
   a = a + data[c];
   c = c + d;
} while (c != b);
```

Example

```
// translate the following loop into C
// assume that the C-level integer `i` is
// in register t1, s2 holds the C-level
// integer `result`, and s1 holds the
// base address of the integer `MemArray`
//
        addi t1, zero, 0
        addi t2, zero, 100
loop:
        lw s3, 0(s1)
        add s2, s2, s3
        addi s1, s1, 4
        addi t1, t1, 1
        blt t1, t2, loop
```

Example

```
// assume a, b, c, v are
// in s1, s2, s3, s4
switch (v) {
  case 0:
    a = b + c;
    break;
  case 1:
    a = b - c;
    break;
  case 2:
    a = b * c;
    break;
}
```