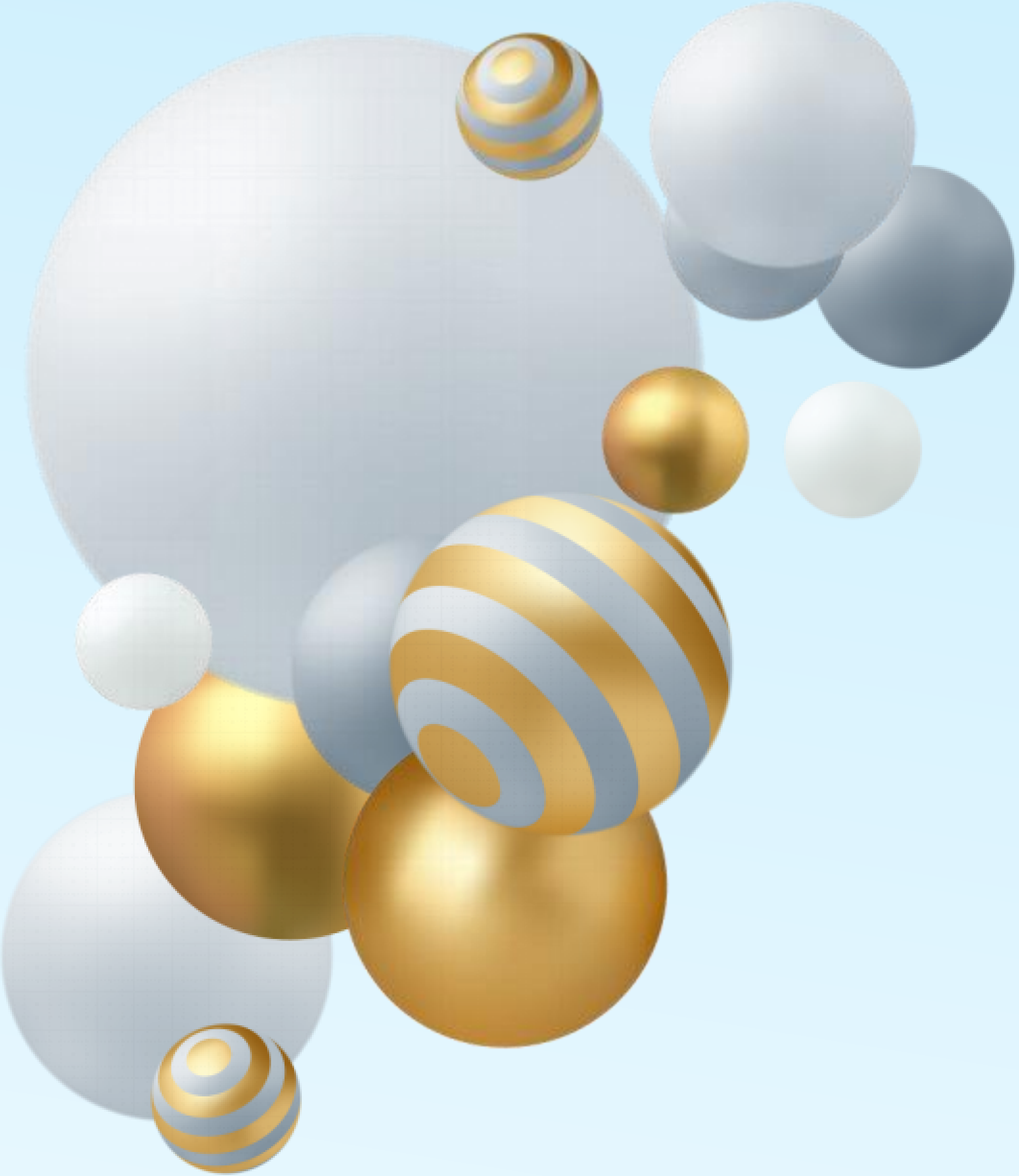
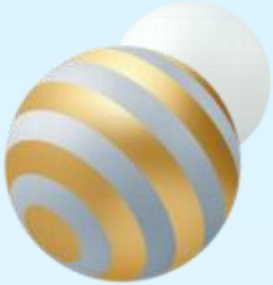


Computer organization

Lab3 RISC-V instructions(1)

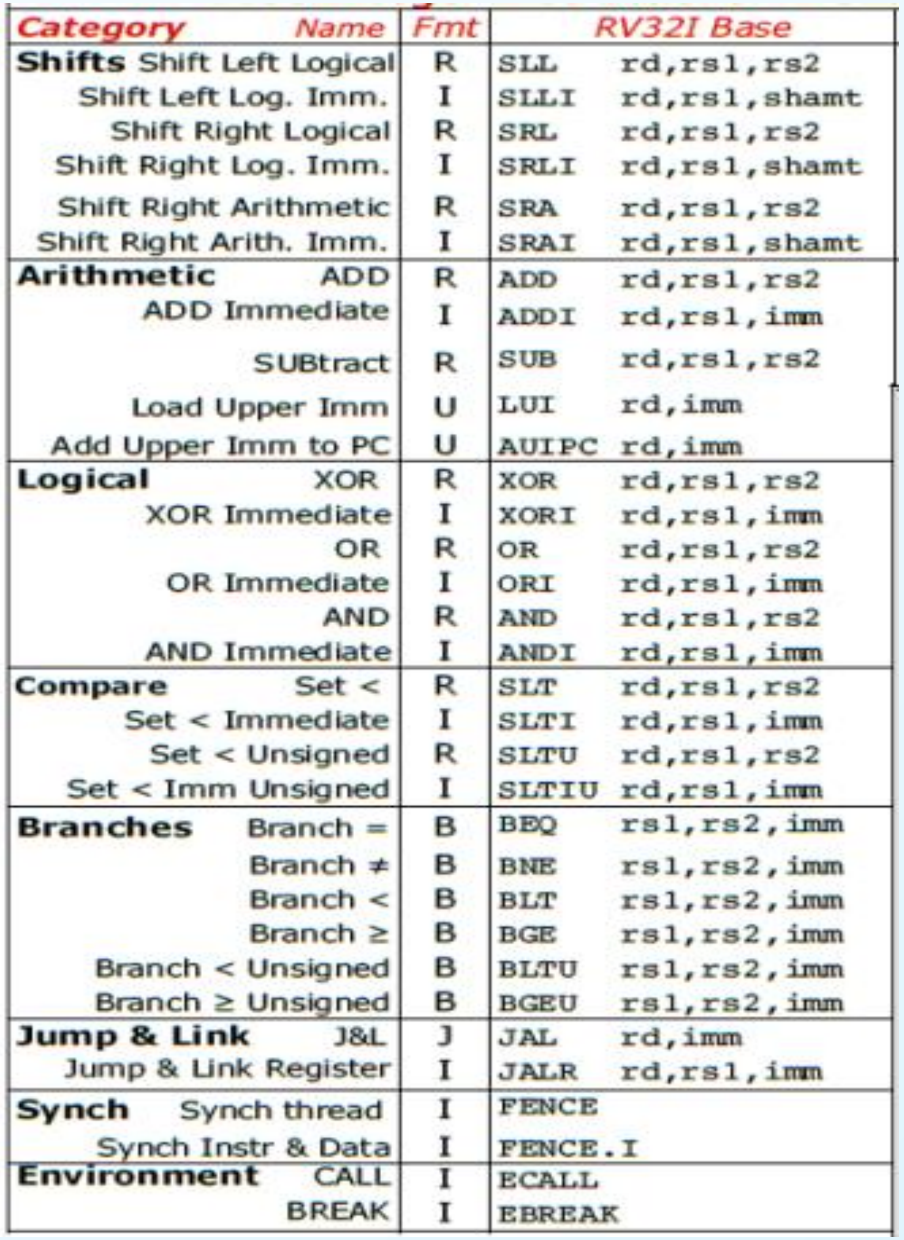
Logical & branch



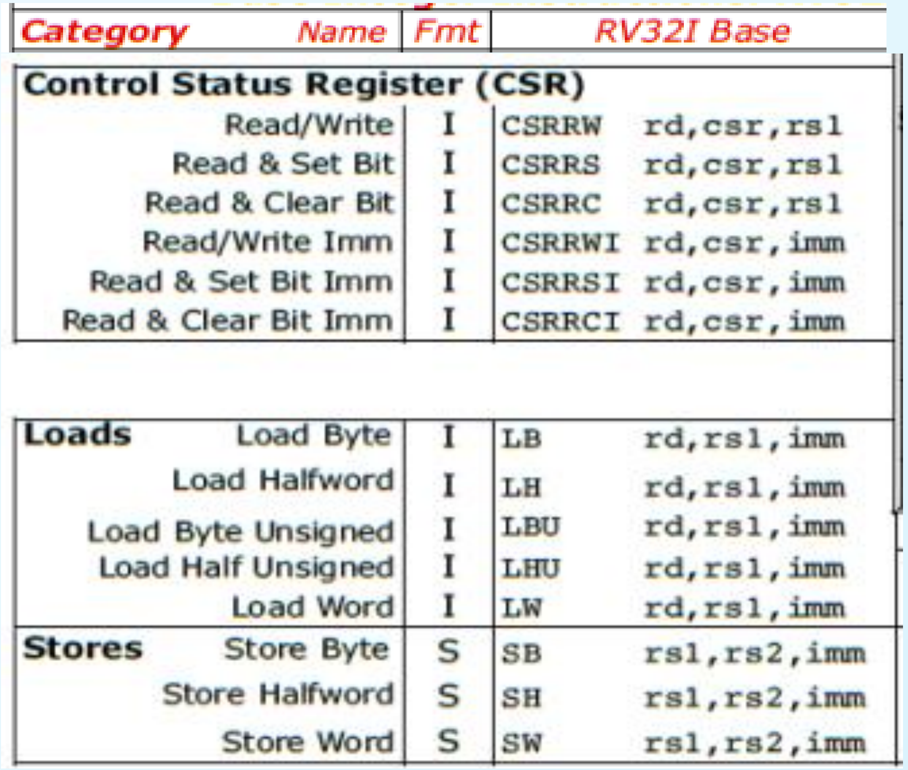
Topics

- RISC-V Instructions
  - ✓ Data transfer (load, store)
  - ✓ Calculation(arithmetic, logical, shift)
  - ✓ Jump instructions(conditional branch, unconditional branch)
- Instruction execution order
  - ✓ PC register
  - ✓ PC updating
- Practice

RISC-V instructions: Common operations



Category	Name	Fmt	RV32I Base	
Shifts	Shift Left Logical	R	SLL	rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI	rd,rs1,shamt
	Shift Right Logical	R	SRL	rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI	rd,rs1,shamt
Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm
Logical	XOR	R	XOR	rd,rs1,rs2
	XOR Immediate	I	XORI	rd,rs1,imm
	OR	R	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches	Branch =	B	BEQ	rs1,rs2,imm
	Branch ≠	B	BNE	rs1,rs2,imm
	Branch <	B	BLT	rs1,rs2,imm
	Branch ≥	B	BGE	rs1,rs2,imm
	Branch < Unsigned	B	BLTU	rs1,rs2,imm
	Branch ≥ Unsigned	B	BGEU	rs1,rs2,imm
Jump & Link	J&L	J	JAL	rd,imm
	Jump & Link Register	I	JALR	rd,rs1,imm
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
Environment	CALL	I	ECALL	
	BREAK	I	EBREAK	



Category	Name	Fmt	RV32I Base	
Control Status Register (CSR)				
	Read/Write	I	CSRRW	rd,csr,rs1
	Read & Set Bit	I	CSRRS	rd,csr,rs1
	Read & Clear Bit	I	CSRRC	rd,csr,rs1
	Read/Write Imm	I	CSRRWI	rd,csr,imm
	Read & Set Bit Imm	I	CSRRSI	rd,csr,imm
	Read & Clear Bit Imm	I	CSRRCI	rd,csr,imm
Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	LHU	rd,rs1,imm
	Load Word	I	LW	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	SW	rs1,rs2,imm

RISC-V instructions: Logical operation

C and Java logical operators and their corresponding RISC-V instructions

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

- To operate on fields of bits within a word or even on individual bits.
- Examples:
  - ✓ andi with 0x000000FF isolates the least significant byte
  - ✓ sll 2 bits to achieve the operation of multiplying by 4
- Question: there is no NOT in RISC-V, why? How to implement NOT?

RISC-V instructions: Logical operation demo(1)

- Run the demos on right hand and answer the questions.
  - ✓ Q1: Are the outputs of two demos the same?
  - ✓ Q2: If use 5 instead of 4 as the initial value on dvalue2, are the outputs of two demos the same?
  - ✓ Q3: On which situation could use 'and' operation to get the remainder instead of division?
  - ✓ Q4: Do the logical operations work quicker than arithmetic operations?

```
# Piece 3-1

.data
        dvalue1: .byte 27
        dvalue2: .byte 4
.text
        lb t0, dvalue1
        lb t1, dvalue2

        rem a0, t0, t1

        li a7, 1
        ecall

        li a7, 10
        ecall
```

```
# Piece 3-2

.data
        dvalue1: .byte 27
        dvalue2: .byte 4
.text
        lb t0, dvalue1
        lb t1, dvalue2

        addi t1, t1, -1
        and a0, t0, t1

        li a7, 1
        ecall

        li a7, 10
        ecall
```

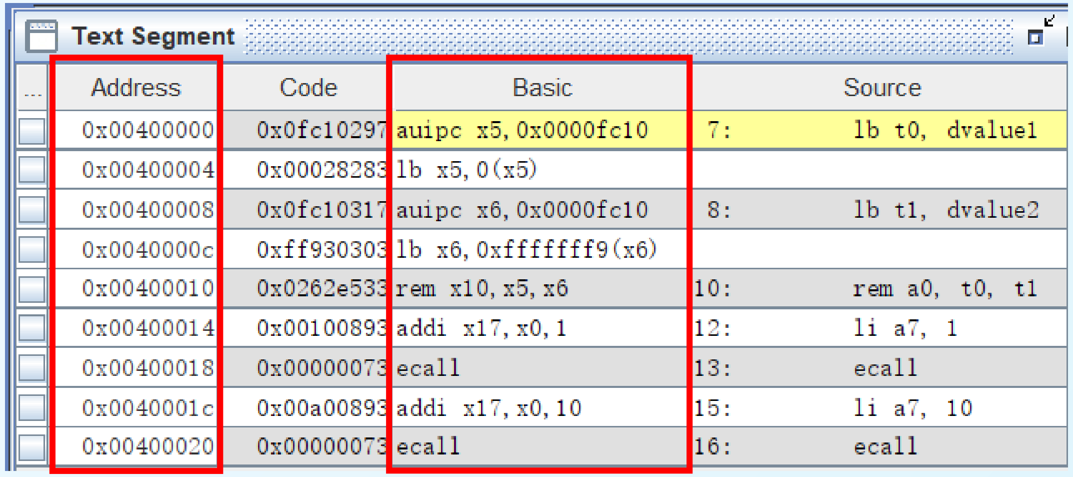
RISC-V instructions: Logical operation demo(2)

- Run the demos on right hand and answer the questions.
  - ✓ Q1: What’s the value of t2 after executing slli instruction?
  - ✓ Q2: What’s the value of t3 after executing srli instruction?
  - ✓ Q3: What is the function of this piece of codes?
  - ✓ Q4: If we use srai instruction instead of “srli t3, t1, 16”, will the result be same?
  - ✓ Q5: If we change the value of dvalue1 to 0x12345678, what will be the answer to Q4?

```
# Piece 3-3
.include "macro_print_str.asm"
.data
	dvalue1: .word 0x87654321
.text
	lw a0, dvalue1
	li a7, 34
	ecall

	mv t1, a0
	slli t2, t1, 16
	srli t3, t1, 16
	or t1, t2, t3

	print_string("\n")
	mv a0, t1
	li a7, 34
	ecall
	end
```

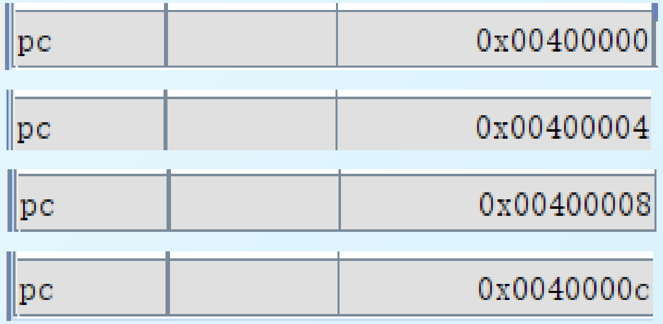
Instruction execution order: PC register

- The CPU takes the value of the **PC** register as the address and fetches the corresponding instruction from the memory.
  - ✓ PC register is **32** bit wide.
  - ✓ PC register maintains the address of the instruction to be executed.
  - ✓ **After** the current instruction is executed, the value of the **PC** register will be **updated** to determine the next instruction to be executed.



Text Segment

...	Address	Code	Basic	Source
□	0x00400000	0x0fc10297	auipc x5, 0x0000fc10	7: lb t0, dvalue1
□	0x00400004	0x00028283	lb x5, 0(x5)	
□	0x00400008	0x0fc10317	auipc x6, 0x0000fc10	8: lb t1, dvalue2
□	0x0040000c	0xff930303	lb x6, 0xfffffff9(x6)	
□	0x00400010	0x0262e533	rem x10, x5, x6	10: rem a0, t0, t1
□	0x00400014	0x00100893	addi x17, x0, 1	12: li a7, 1
□	0x00400018	0x00000073	ecall	13: ecall
□	0x0040001c	0x00a00893	addi x17, x0, 10	15: li a7, 10
□	0x00400020	0x00000073	ecall	16: ecall



pc		0x00400000
pc		0x00400004
pc		0x00400008
pc		0x0040000c

Instruction execution order: PC updating

- Check if the current instruction is non-jump
  - ✓ If the current instruction is non-jump instruction: **PC = PC+4**
  - ✓ If the current instruction is jump instruction
    - If the current instruction is unconditional jump: **PC = destination address**
    - If the current instruction is conditional jump
      - If the condition is **met**： **PC = destination address**
      - If the condition is **not met**: **PC = PC + 4**

Instruction execution order: Instructions

Some RISC-V jump instructions

Category	Instruction	Example	Meaning	Comments
Conditional branch	branch if equal	beq x5, x6, label	if(x5 == x6) go to label	PC-relative branch if registers equal
	branch if not equal	bne x5, x6, label	if(x5 != x6) go to label	PC-relative branch if registers not equal
	branch if less than	blt x5, x6, label	if(x5 < x6) go to label	PC-relative branch if registers less
	branch if greater or equal	bge x5, x6, label	if(x5 >= x6) go to label	PC-relative branch if registers greater or equal
	branch if less than, unsigned	bltu x5, x6, label	if(x5 < x6) go to label	PC-relative branch if registers less, unsigned
	branch if greater or equal, unsigned	bgeu x5, x6, label	if(x5 >= x6) go to label	PC-relative branch if registers greater or equal, unsigned
Unconditional branch	jump and link	jal x1, label	x1 = PC + 4; go to label	PC-relative procedure call
	jump and link register	jalr x1, 100(x5)	x1 = PC + 4; go to x5+100	Procedure return; indirect call
Pseudo instructions	jump	j label1	PC = address of label1	Jump to statement at label
	jump register	jr	PC = t0 (ra / x5)	Jump to address in t0

Instruction execution order: Branch

- Run the two demos on right hand and answer the questions.
  - ✓ Q1: Are the running results of two demos the same?
  - ✓ Q2: Observe values of PC when executing.
  - ✓ Q3: Modify them without changing the results by using **blt** instead.

```
# Piece 3-5

.include "macro_print_str.asm"
.text
	print_string("please input your score (0~100):")
	li a7, 5
	ecall
	mv t0, a0
case1:
	li t1, 60
	bge t0, t1, passLable
case2:
	j failLable

passLable:
	print_string("\nPASS (exceed or equal to 60) ")
	j caseEnd
failLable:
	print_string("\nFail (less than 60)")
	j caseEnd
caseEnd:
	end
```

```
# Piece 3-6

.include "macro_print_str.asm"
.text
	print_string("please input your score (0~100):")
	li a7, 5
	ecall
	mv t0, a0
case1:
	li t1, 60
	bge t0, t1, passLable
	j case2
case2:
	j failLable

passLable:
	print_string("\nPASS (exceed or equal to 60) ")
	j caseEnd
failLable:
	print_string("\nFail (less than 60)")
	j caseEnd
caseEnd:
	end
```

Instruction execution order: Loop

- Compare the operations of loop which calculates the sum from 1 to 10 in java and RISC-V.

Code in Java

```
public class CalculateSum{
    public static void main(String [] args){
        int i = 0;
        int sum = 0;
        for(i=0;i<=10;i++)
                sum = sum + i;
        System.out.print("The sum from 1 to 10 : " + sum );
    }
}
```

Code in RISC-V

```
# Piece 3-7

.include "macro_print_str.asm"
.data
        #....
.text
        add t1, zero, zero
        addi t0, zero, 0
        addi t3, zero, 10
calcu:
        addi t0, t0, 1      # i++
        add t1, t1, t0      # sum+=i
        blt t0, t3, calcu  #if(t0 < t3)  add i to sum

        print_string ("The sum from 1 to 10: ")
        mv a0, t1
        li a7, 1
        ecall

        end
```

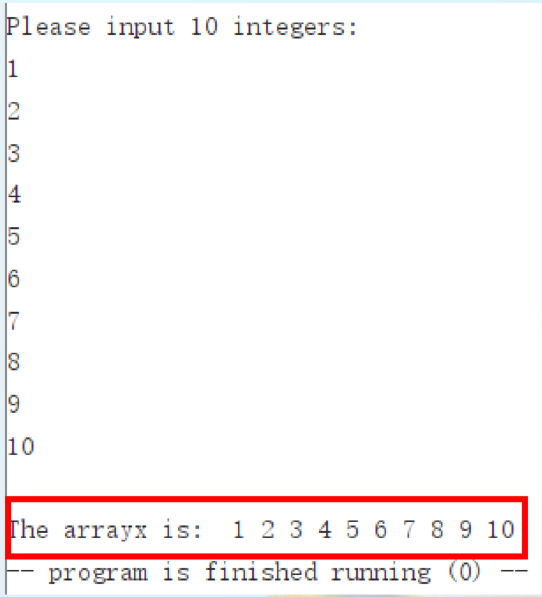
Instruction execution order: Demo(1)

- The following program is expected to get 10 integers from the input device, and print it as the following sample, Will the code get desired result? If not, what happened? Please modify the codes.

```
# Piece 3-8-1

.include "macro_print_str.asm"
.data
        arrayx:     .space      10
        str:        .asciz      "\nThe arrayx is:  "
.text
main:
        print_string("Please input 10 integers: \n")
        add t0, zero, zero
        addi t1, zero, 10
        la t2, arrayx
```

```
#Piece 3-8-2

loop_r:
        li a7, 5
        ecall
        sw a0, (t2)
        addi t0, t0, 1
        addi t2, t2, 4
        bne t0, t1, loop_r

        la a0, str
        li a7, 4
        ecall
        addi t0, zero, 0
        la t2, arrayx
```

```
#Piece 3-8-3

loop_w:
        lw a0, (t2)
        li a7, 1
        ecall
        print_string(" ")
        addi t2, t2, 4
        addi t0, t0, 1
        bne t0, t1, loop_w

        end
```



```
Please input 10 integers:
1
2
3
4
5
6
7
8
9
10

The arrayx is:  1 2 3 4 5 6 7 8 9 10
-- program is finished running (0) --
```

Instruction execution order: Demo(2)

- The function of following codes is to get 5 integers from input device, and find the min value and max value among them. There are 4 sections of code, reorganize the sequence of each section. Can the program find the real min and max values? If not, please modify the codes.

```
#Piece 3-9
#section A
        print_string("\nMin : ")
        mv a0, t0
        li a7, 1
        ecall
        print_string("\nMax : ")
        mv a0, t1
        li a7, 1
        ecall
        end
```

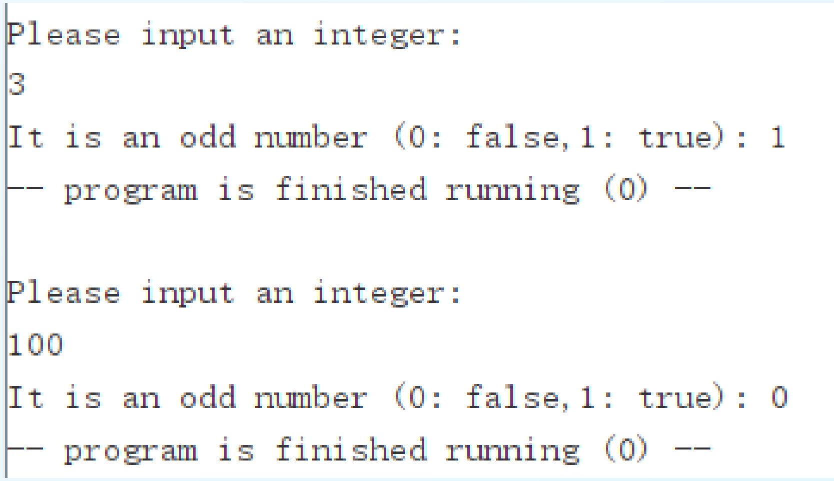
```
#section B
judge_times:
        addi t4, t4, 1
        bge t3, t4, loop
```

```
#section C
set_max:
        mv t1, a0
        j set_min
set_min:
       bge a0, t0, judge_times
        mv t0, a0
        j judge_times
```

```
#section D
.include "macro_print_str.asm"
.data
        min: .word 0
        max: .word 0
.text
        lw t0, min
        lw t1, max
        li t3, 4
        li t4, 0
        print_string("Please input 5
integer:\n")
loop:
        li a7, 5
        ecall
        bge a0, t1, set_max
        j set_min
```

Practice 1

- 1-1. Run the two demos in “Instruction execution order: Demo” part, and answer all the questions.
- 1-2. Here is a demo to meet the following function: get the integer from input, judge whether the data is odd or not, if it is odd then print 1, else print 0.
  - 1-2-1. Run the demo to see whether the function is ok? If not, please modify the code to meet the design expectations.
  - 1-2-2. Which is(are) basic instruction(s) in the following set: li, mv, and, ecall, end?

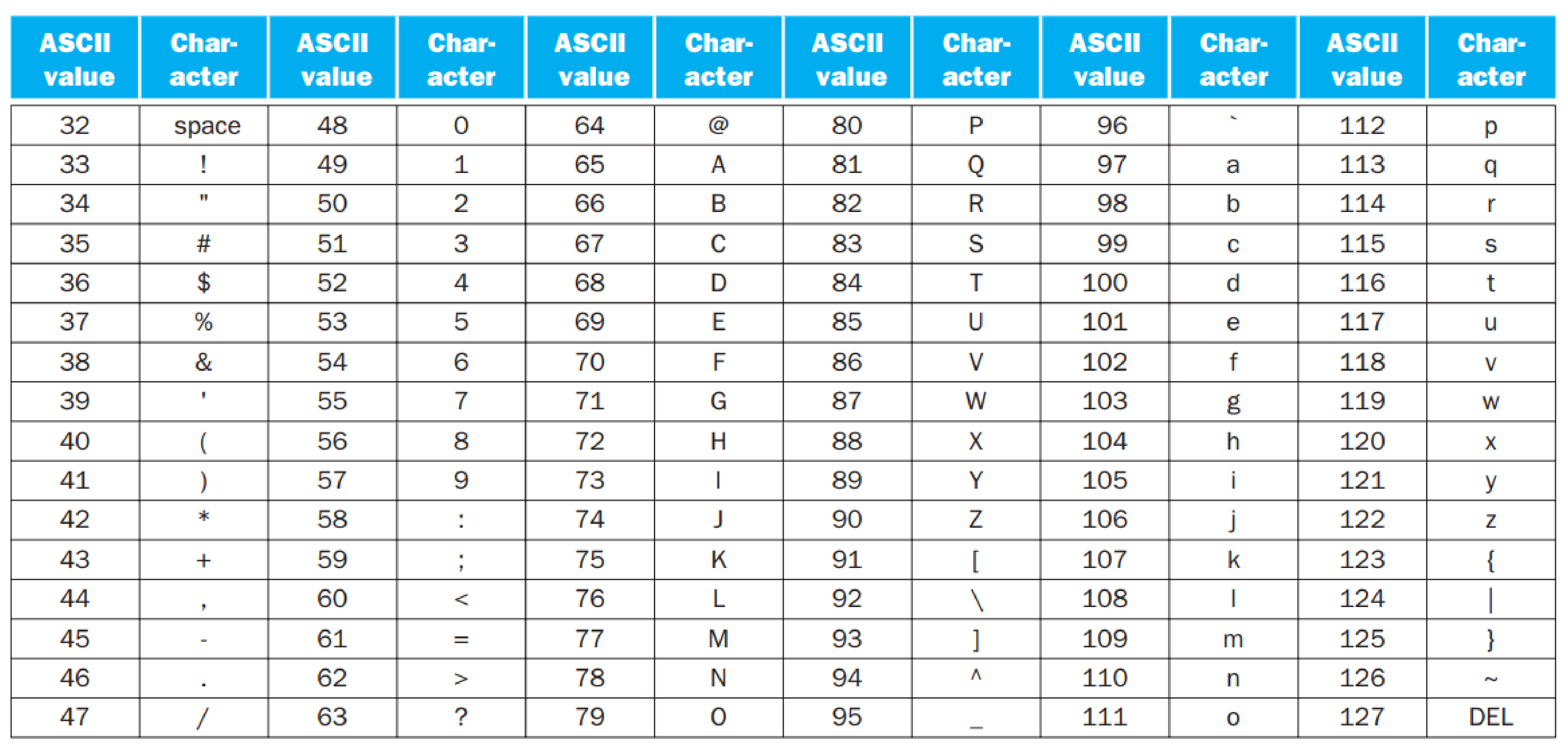


```
Please input an integer:
3
It is an odd number (0: false,1: true): 1
-- program is finished running (0) --

Please input an integer:
100
It is an odd number (0: false,1: true): 0
-- program is finished running (0) --
```

```
#Piece 3-10

.include "macro_print_str.asm"
.data
.text
main:
        print_string("Please input an integer: \n")
        li a7, 5
        ecall
        li t1, 1

        mv t0, a0
        and a0, t1, t0

        print_string("It is an odd number (0: false,1: true): ")
        mv a0, t0
        li a7, 1
        ecall

        end
```

Practice 2

- Read a character, judge whether the binary representation of the character's ASCII code is palindrome. For example, the ASCII code of 'f' (102 in decimal, 0110_0110 in binary) is a binary palindrome, the ASCII code of space (32 in decimal, 0010_0000 in binary) is not.



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

Tip: Big-endian vs Little-endian(1)

- The CPU's byte ordering scheme (or endian issues) affects memory organization and defines the relationship between address and byte position of data in memory.
  - ✓ a Big-endian system means byte 0 is always the most-significant (leftmost) byte.
  - ✓ a Little-endian system means byte 0 is always the least-significant (rightmost) byte.

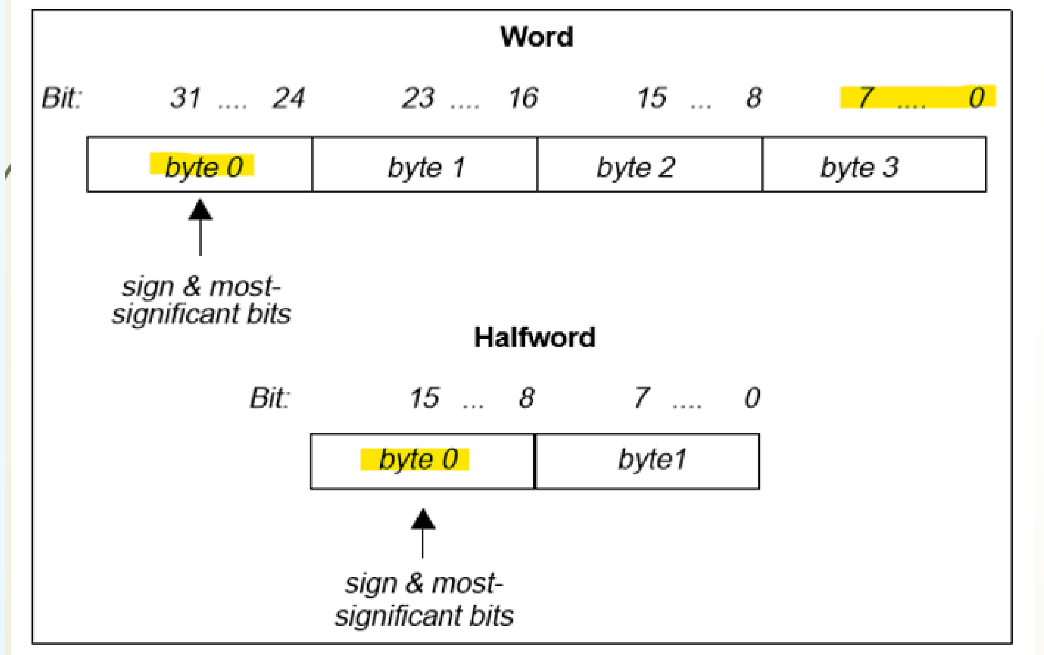


Figure 1-1: Big-endian Byte Ordering

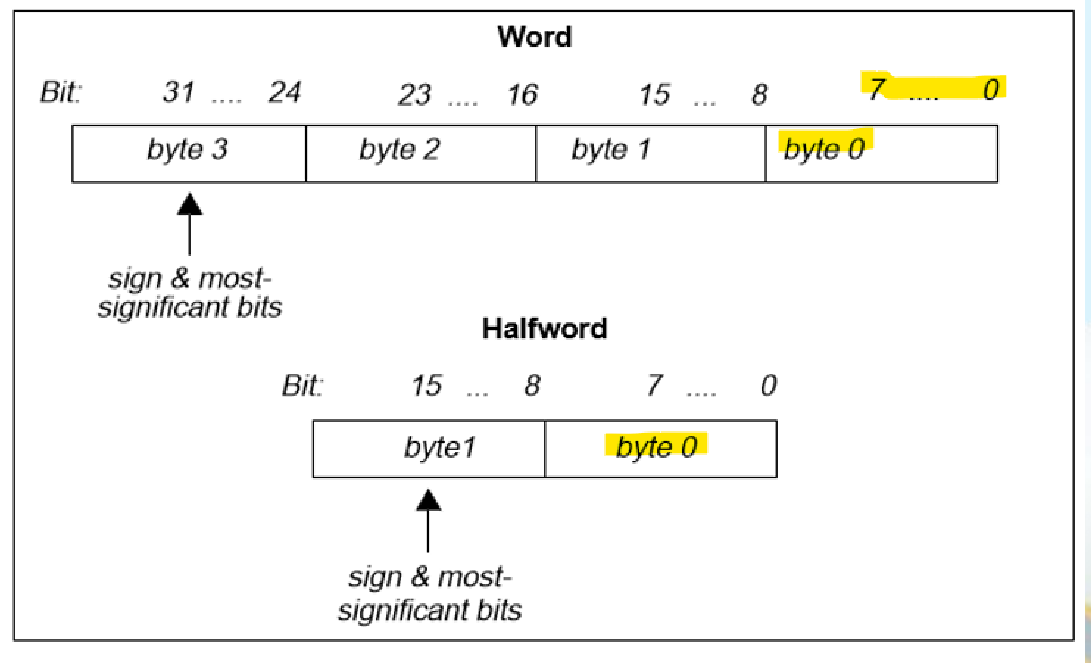


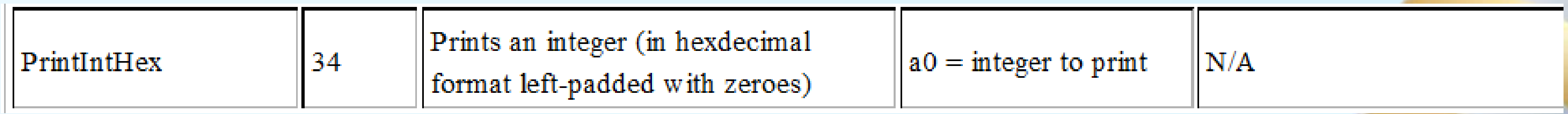
Figure 1-2: Little-endian Byte Ordering

Tip: Big-endian vs Little-endian(2)

- Run the demo to anwer the question : Does your simulator work on big-endian or little-endian, explain the reasons.

```
#Piece 3-11
.include "macro_print_str.asm"
.data
	tdata0: .byte   0x44,0x33,0x22,0x11
	tdata:   .word  0x44332211
.text
main:
	lb a0, tdata0
	li a7, 34
	ecall
	lb a0, tdata
	li a7, 34
	ecall
	end
```

```
#Piece 3-12
.include "macro_print_str.asm"
.data
	tdata0: .byte   0x11,0x22,0x33,0x44
	tdata:   .word  0x44332211
.text
main:
	lb a0, tdata0
	li a7, 34
	ecall
	lb a0, tdata
	li a7, 34
	ecall
	end
```



PrintIntHex	34	Prints an integer (in hexdecimal format left-padded with zeroes)	a0 = integer to print	N/A

Tip: Big-endian vs Little-endian(3)

- Run the demo to anwer the question :

  - ✓ *Q1. What's the output of this demo?*

    *A. **0x**00000044**0x**00000033**0x**00000022**0x**00000011**0x**55667788*

    *B.**0x**00000011**0x**00000022**0x**00000033**0x**00000044**0x**88776655*

    *C.**0x**00000055**0x**00000066**0x**00000077**0x**00000088**0x**11223344*

    *D.**0x**00000088**0x**00000077**0x**00000066**0x**00000055**0x**44332211*

  - ✓ *Q2. Does* your simulator work on big-endian or little-endian, explain the reasons.

```
#Piece 3-13

.include "macro_print_str.asm"
.data
        tdata0: .word   0x11223344, 0x55667788
.text
main:
        la t0, tdata0
        lb a0, (t0)
        li a7, 34
        ecall

        la t0, tdata0
        lb a0, 1(t0)
        ecall

        lb a0, 2(t0)
        ecall

        lb a0, 3(t0)
        ecall

        lw a0, 4(t0)
        ecall

        end
```

Practice 3

- Print out a 9*9 multiplication table.
  - ✓ Define a function to print a*b = c , the value of “a” is from parameter t0, the value of “b” is from parameter t1.
  - ✓ Less system call is better(more effective).