Usage of the “charis.py”:

- Make sure you have the bitstring library(script tries to install it)
  - Install by running “pip install bitstring –user”
- Create a text file with all your instructions inside
- Run “./charis.py -a filename to create rom.data with the assembled instructions
- To disassemble a file run “./charis -d filename” to create filename + .dis with the assembly instructions.

Note: on windows run as: “python charis.py [...]”

charis_1 - The first program example provided by the lab staff(contains .asm and rom.data):

```
00: addi r5, r0, 8
04: ori r3, r0, 0xABCD
08: sw r3, 4(r0)
0C: lw r10, -4(r5)
10: lb r16, 4(r0)
14: nand r4, r10, r16
```

charis_2 – The second program example provided by the lab staff:

```
00: bne r5, r5, 8
04: b -2
08: addi r1, r0, 1
```

charis_3 – A program I wrote that has all 24 instructions inside:

```
li r1, 0xa
li r2, 0x900
li r3, 0xfff0
add r4, r2, r1
sub r4, r1, r2
and r4, r4, r0
not r4, r4
or r4, r0, r1
nand r4, r1, r3
nor r4, r1, r4
sra r4, r3
srl r4, r3
sll r4, r3
rol r4, r3
ror r4, r3
lui r4, 0xffff
addi r4, r1, 0xfff6
nandi r4, r1, 0x0
ori r4, r1, 0x0
li r1, 0xffff
li r2, 0xa
sw r1, 0x2(r2)
lw r2, 0x2(r2)
sb r1, 0x0(r0)
lb r2, 0x0(r0)
beq r0, r0, 0x4
b 0x1
bne r0, r2, 0xfffe
```

charis_4 is a reduced charis_3 program for test benching the DATAPATH

Important Note: If you are having issues (import errors, etc) with the project code please check out the archived project hosted on my GitHub repo. (ACE312 Folder, full code source also available)

CHARISA

CHAnia Risc Instruction Set Architecture

1.) 32, 32-bit Registers with the exception of R0 -> Hardwired to 0.
2.) 2 Types of Instructions always 32-bits in length.
3.) ALU/Logic operations include: add, sub, nand, not, or, sra, sll, srl, ror, rol, li, addi, nandi and ori.
4.) 3 Branching Instructions: b, bne, beq.
5.) 4 Memory Instructions: lw, sw, lb, sb. (see below for RLT notation)
6.) Byte addressable Memory

Types of Instructions:

R - Type

6-bits	**5-bits**	**5-bits**	**5-bits**	**5-bits**	**6-bits**
OpCode	**Rs**	**Rd**	**Rt**	**Not-used**	**func**

I - Type

6-bits	**5-bits**	**5-bits**	**16-bits**
OpCode	**Rs**	**Rd**	**Immediate**

List of Instructions with RTL notation:

Opcode	FUNC	Instruction	RTL Notation
100000	110000	add	RF[Rd] <- RF[Rs] + RF[Rt]
100000	110001	sub	RF[Rd] <- RF[Rs] - RF[Rt]
100000	110010	and	RF[Rd] <- RF[Rs] AND RF[Rt]
100000	110011	or	RF[Rd] <- RF[Rs] OR RF[Rt]
100000	110100	not	RF[Rd] <- !RF[Rs]
100000	110101	nand	RF[Rd] <- RF[Rs] NAND RF[Rt]
100000	110110	nor	RF[Rd] <- RF[Rs] NOR RF[Rt]
100000	111000	sra	RF[Rd] <- RF[Rs] >> 1
100000	111001	srl	RF[Rd] <- RF[Rs] >> 1 (Logical, Zfill MSB)
100000	111010	sll	RF[Rd] <- RF[Rs] << 1 (Logical, Zfill MSB)
100000	111100	rol	RF[Rd] <- Rotate_left(F[Rs])
100000	111101	ror	RF[Rd] <- Rotate_right(F[Rs])
111000	-	li	RF[Rd] <- SignExtend(Immed)
111001	-	lui	RF[Rd] <- Immed << 16 (Zfill)
110000	-	addi	RF[Rd] <- RF[Rs] + SignExtend(Immed)
110010	-	nandi	RF[Rd] <- RF[Rs] NAND SignExtend(Immed)
110011	-	ori	RF[Rd] <- RF[Rs] OR SignExtend(Immed)
111111	-	b	PC <- PC + 4 + (SignExtend(Immed)<<2)

000000	-	beq	If RF[Rd] == RF[Rs]
			PC <- PC + 4 + (SignExtend(Immed)<<2)
			Else
			PC <- PC + 4

000001	-	bne	If RF[Rd] != RF[Rs]
			PC <- PC + 4 + (SignExtend(Immed)<<2)
			Else
			PC <- PC + 4

000011	-	lb	RF[Rd] <- Zfill(31 downto 8) & MEM[RF[Rs] + SignExtend(immed)](7 downto 0)
000111	-	sb	MEM[RF[Rs] + SignExtend(immed)] <- Zfill(31 downto 8) & RF[Rd](7 downto 0)
001111	-	lw	RF[Rd] <- MEM[RF[Rs] + SignExtend(immed)]
011111	-	sw	MEM[RF[Rs] + SignExtend(immed)] <- RF[Rd]