A blue and gold logo

AI-generated content may be incorrect.

**San Francisco Bay University**

**EE488 - Computer Architecture**

**Homework Assignment #6**

**Due day: 5/4/2025**

**Instruction:**

1. **The homework answer sheet should contain the original questions and corresponding answers.**
2. **The answer sheet must be in MS-Word file format with Github links for the programming questions. As follows is the answer sheet name format.**

***<course\_id>\_week<week\_number>\_StudentID\_FirstName\_LastName.pdf***

1. **The program name in Github must follow the format like   
   *<course\_id>\_week<week\_number>\_q<question\_number>\_StudentID\_FirstName\_LastName***
2. **Show screenshot of all running results, including the system date/time.**
3. **The calculation process must be typed if needed, handwriting can’t be accepted.**
4. **Only accept homework submission uploaded via Canvas.**
5. **Overdue homework submission can’t be accepted.**
6. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
7. Write Python def function to design 8-bits ALU based on the following opcodes.

Note: The input parameters for A and B need to be converted to binary number in def function

|  |  |
| --- | --- |
| Opcode | Operations |
| 0000 | Out = A + B |
| 0001 | Out = A - B |
| 0010 | Out = A \* B |
| 0011 | Out = A / B |
| 0100 | Out = A << 1 |
| 0101 | Out = A >> 1 |
| 0110 | Out = A rotated left by 1 |
| 0111 | Out = A rotated right by 1 |
| 1000 | Out = A and B |
| 1001 | Out = A or B |
| 1010 | Out = A xor B |
| 1011 | Out = A nor B |
| 1100 | Out = A nand B |
| 1101 | Out = A xnor B |
| 1110 | Out = 1 if A>B else 0 |
| 1111 | Out = 1 if A=B else 0 |

def alu(opcode: str, A: int, B: int) -> str:

    """

    8-bit ALU.

    opcode: 4-bit string, e.g. '0000'

    A, B: integer inputs (any range)

    Returns: 8-bit binary string of the result.

    """

    # --- convert inputs to 8-bit values ---

    a = int(format(A & 0xFF, '08b'), 2)

    b = int(format(B & 0xFF, '08b'), 2)

    # --- decode opcode and compute out (may exceed 8 bits) ---

    if   opcode == '0000':  # add

        out = a + b

    elif opcode == '0001':  # sub

        out = a - b

    elif opcode == '0010':  # mul

        out = a \* b

    elif opcode == '0011':  # div (integer)

        out = a // b if b != 0 else 0

    elif opcode == '0100':  # shift left logical

        out = (a << 1)

    elif opcode == '0101':  # shift right logical

        out = (a >> 1)

    elif opcode == '0110':  # rotate left by 1

        out = ((a << 1) & 0xFF) | ((a >> 7) & 0x01)

    elif opcode == '0111':  # rotate right by 1

        out = ((a >> 1) & 0x7F) | ((a & 0x01) << 7)

    elif opcode == '1000':  # and

        out = a & b

    elif opcode == '1001':  # or

        out = a | b

    elif opcode == '1010':  # xor

        out = a ^ b

    elif opcode == '1011':  # nor = ¬(A ∨ B)

        out = ~(a | b)

    elif opcode == '1100':  # nand = ¬(A ∧ B)

        out = ~(a & b)

    elif opcode == '1101':  # xnor = ¬(A ⊕ B)

        out = ~(a ^ b)

    elif opcode == '1110':  # A > B ?

        out = 1 if a > b else 0

    elif opcode == '1111':  # A == B ?

        out = 1 if a == b else 0

    else:

        raise ValueError(f"Unknown opcode '{opcode}'")

    # --- mask result to 8 bits ---

    out &= 0xFF

    # --- return as 8-bit binary string ---

    return format(out, '08b')

if \_\_name\_\_ == "\_\_main\_\_":

    # test the specific cases you already have

    cases = [

        ('0000', 15, 5),

        ('0001', 15, 5),

        ('0110', 0x85, 0),

        ('1110', 7, 3),

        ('1010', 0b10101010, 0b01010101),

    ]

    for op, A, B in cases:

        out\_bin = alu(op, A, B)

        out\_int = int(out\_bin, 2)

        print(f"{op} | A={A:#04x} B={B:#04x} → int {out\_int}, bin {out\_bin}")

    print("\n-- Exhaustive opcode test --")

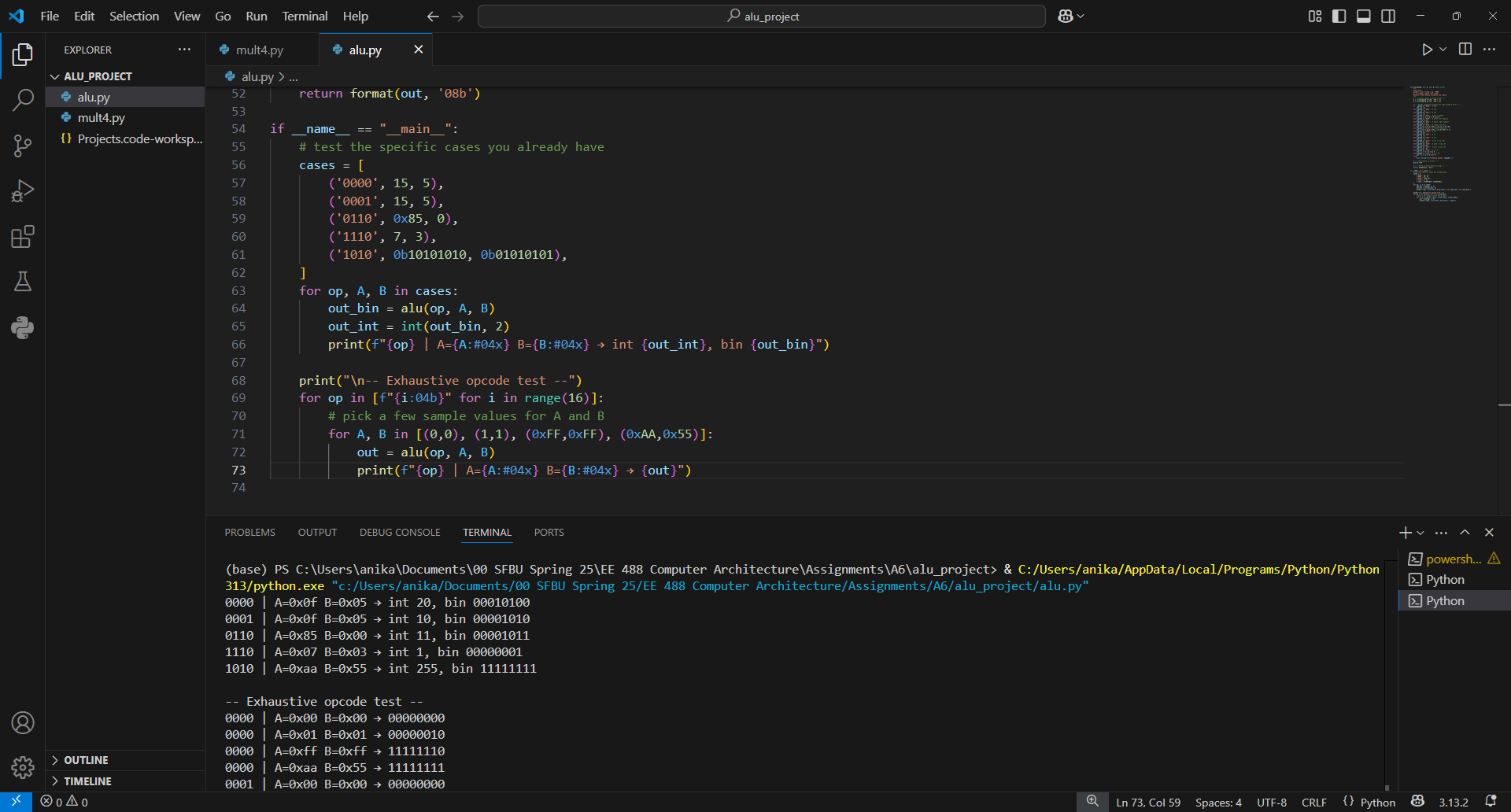
    for op in [f"{i:04b}" for i in range(16)]:

        # pick a few sample values for A and B

        for A, B in [(0,0), (1,1), (0xFF,0xFF), (0xAA,0x55)]:

            out = alu(op, A, B)

            print(f"{op} | A={A:#04x} B={B:#04x} → {out}")



1. Write Python programs to design a 4-bits multiplier which implements Booth’s algorithm and one of multiplication algorithms from 3 versions shown in the handout of *Lec06-alu.pdf,* respectively.
2. # mult4.py
3. def mult4\_shiftadd(x: int, y: int) -> int:
4. """
5. 4-bit unsigned shift-add multiplier (Version 1).
6. Implements:
7. for i in 0..3:
8. if y[i] == 1: product += (x << i)
9. shift y right each cycle.
10. Returns 8-bit product of x \* y (0..15 × 0..15 → 0..255).
11. """
12. mcand = x & 0xF     # 4-bit multiplicand
13. mult  = y & 0xF     # 4-bit multiplier
14. prod  = 0
15. for \_ in range(4):
16. if mult & 1:
17. prod += mcand
18. mcand <<= 1
19. mult  >>= 1
20. return prod & 0xFF  # mask to 8 bits
21. def mult4\_booth(x: int, y: int) -> int:
22. """
23. 4-bit signed Booth’s algorithm multiplier.
24. x, y are treated as signed 4-bit (–8..+7). Returns signed 8-bit Python int.
26. Algorithm (per Booth’s rules):
27. - Examine (Q0, Q−1):
28. 01 → add  M into A
29. 10 → sub  M from A
30. 00 or 11 → no op
31. - Arithmetic right shift of [A(5b), Q(4b), Q−1]
32. - Repeat 4 times.
33. """
34. def to\_u4(v): return v & 0xF
35. def to\_s4(u): return u - 0x10 if (u & 0x8) else u
36. M   = to\_s4(to\_u4(x))    # signed multiplicand
37. Q   = to\_u4(y)           # unsigned bits of multiplier
38. A   = 0
39. Q\_1 = 0
40. for \_ in range(4):
41. q0 = Q & 1
42. # Booth step :contentReference[oaicite:4]{index=4}:contentReference[oaicite:5]{index=5}
43. if   q0 == 1 and Q\_1 == 0:
44. A = (A - M) & 0x1F
45. elif q0 == 0 and Q\_1 == 1:
46. A = (A + M) & 0x1F
47. # pack [A(5b), Q(4b), Q−1] into 10 bits and arithmetic shift right by 1
48. combo = (A << 5) | (Q << 1) | Q\_1
49. msb   = (combo >> 9) & 1
50. combo = (combo >> 1) | (msb << 9)
51. A   = (combo >> 5) & 0x1F
52. Q   = (combo >> 1) & 0xF
53. Q\_1 = combo & 1
54. # combine and sign-extend to Python int
55. result = ((A & 0x1F) << 4) | Q
56. result &= 0xFF
57. if result & 0x80:
58. result -= 0x100
59. return result
60. if \_\_name\_\_ == "\_\_main\_\_":
61. # Demo of both multipliers
62. print("Unsigned shift-add (0..15 × 0..15):")
63. for a, b in [(3,6), (7,7), (15,15)]:
64. print(f"  {a:2d} × {b:2d} = {mult4\_shiftadd(a,b):3d}")
65. print("\nBooth’s algorithm (–8..+7 × –8..+7):")
66. for a, b in [(-7,3), (7,-3), (-8,7), (-8,-8)]:
67. print(f"  {a:3d} × {b:3d} = {mult4\_booth(a,b):4d}")

