



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
 3. The program name in Github must follow the format like
<course_id>_week<week_number>_q<question_number>_StudentID_FirstName_LastName
 4. Show screenshot of all running results, including the system date/time.
 5. The calculation process must be **typed** if needed, handwriting can't be accepted.
 6. Only accept homework submission uploaded via Canvas.
 7. Overdue homework submission can't be accepted.
 8. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)
-
1. 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}")

```

```

File Edit Selection View Go Run Terminal Help
alu_project
EXPLORER
  ALU PROJECT
    alu.py
    mult4.py
    Projects.code-workspace
  alu.py
    52 return format(out, '08b')
    53
    54 if __name__ == "__main__":
    55     # test the specific cases you already have
    56     cases = [
    57         ('0000', 15, 5),
    58         ('0001', 15, 5),
    59         ('0110', 0x85, 0),
    60         ('1110', 7, 3),
    61         ('1010', 0x10101010, 0x01010101),
    62     ]
    63     for op, A, B in cases:
    64         out_bin = alu(op, A, B)
    65         out_int = int(out_bin, 2)
    66         print(f"{op} | A={A:#04x} B={B:#04x} + int {out_int}, bin {out_bin}")
    67
    68     print("\n-- Exhaustive opcode test --")
    69     for op in ["{:04b}" for i in range(16)]:
    70         # pick a few sample values for A and B
    71         for A, B in [(0,0), (1,1), (0xFF,0xFF), (0xAA,0x55)]:
    72             out = alu(op, A, B)
    73             print(f"{op} | A={A:#04x} B={B:#04x} + {out}")
    74
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) PS C:\Users\anika\Documents\00 SFBU Spring 25\EE 488 Computer Architecture\Assignments\W6\alu_project & C:\Users\anika\AppData\Local\Programs\Python\Python
313\python.exe "C:/Users/anika/Documents/00 SFBU Spring 25/EE 488 Computer Architecture/Assignments/W6/alu_project/alu.py"
0000 | A=0x0f B=0x05 + int 20, bin 00010100
0001 | A=0x0f B=0x05 + int 10, bin 00001010
0110 | A=0x25 B=0x00 + int 11, bin 00001011
1110 | A=0x07 B=0x03 + int 1, bin 00000001
1010 | A=0xaa B=0x55 + int 255, bin 11111111

-- Exhaustive opcode test --
0000 | A=0x00 B=0x00 + 00000000
0000 | A=0x01 B=0x01 + 00000010
0000 | A=0xFF B=0xFF + 11111110
0000 | A=0xaa B=0x55 + 11111111
0001 | A=0x00 B=0x00 + 00000000

```

- 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.

```

3. # mult4.py
4.
5. def mult4_shiftadd(x: int, y: int) -> int:
6.     """
7.     4-bit unsigned shift-add multiplier (Version 1).
8.     Implements:
9.     for i in 0..3:
10.         if y[i] == 1: product += (x << i)
11.         shift y right each cycle.
12.     Returns 8-bit product of x * y (0..15 × 0..15 → 0..255).
13.     """
14.     mcand = x & 0xF      # 4-bit multiplicand
15.     mult = y & 0xF      # 4-bit multiplier
16.     prod = 0
17.     for _ in range(4):
18.         if mult & 1:
19.             prod += mcand
20.             mcand <<= 1
21.             mult >>= 1
22.     return prod & 0xFF  # mask to 8 bits
23.
24. def mult4_booth(x: int, y: int) -> int:
25.     """
26.     4-bit signed Booth's algorithm multiplier.

```

```

27.     x, y are treated as signed 4-bit (-8...+7). Returns signed 8-bit
        Python int.
28.
29.     Algorithm (per Booth's rules):
30.         - Examine (Q0, Q-1):
31.             01 → add M into A
32.             10 → sub M from A
33.             00 or 11 → no op
34.         - Arithmetic right shift of [A(5b), Q(4b), Q-1]
35.         - Repeat 4 times.
36.     """
37.     def to_u4(v): return v & 0xF
38.     def to_s4(u): return u - 0x10 if (u & 0x8) else u
39.
40.     M = to_s4(to_u4(x))    # signed multiplicand
41.     Q = to_u4(y)           # unsigned bits of multiplier
42.     A = 0
43.     Q_1 = 0
44.
45.     for _ in range(4):
46.         q0 = Q & 1
47.         # Booth step
:contentReference[oaicite:4]{index=4}:contentReference[oaicite:5]{index=5}
48.         if q0 == 1 and Q_1 == 0:
49.             A = (A - M) & 0x1F
50.         elif q0 == 0 and Q_1 == 1:
51.             A = (A + M) & 0x1F
52.
53.         # pack [A(5b), Q(4b), Q-1] into 10 bits and arithmetic shift
            right by 1
54.         combo = (A << 5) | (Q << 1) | Q_1
55.         msb = (combo >> 9) & 1
56.         combo = (combo >> 1) | (msb << 9)
57.
58.         A = (combo >> 5) & 0x1F
59.         Q = (combo >> 1) & 0xF
60.         Q_1 = combo & 1
61.
62.     # combine and sign-extend to Python int
63.     result = ((A & 0x1F) << 4) | Q
64.     result &= 0xFF
65.     if result & 0x80:
66.         result -= 0x100
67.     return result
68.

```

```
69. if __name__ == "__main__":
70.     # Demo of both multipliers
71.     print("Unsigned shift-add (0..15 × 0..15):")
72.     for a, b in [(3,6), (7,7), (15,15)]:
73.         print(f" {a:2d} × {b:2d} = {mult4_shiftadd(a,b):3d}")
74.
75.     print("\nBooth's algorithm (-8..+7 × -8..+7):")
76.     for a, b in [(-7,3), (7,-3), (-8,7), (-8,-8)]:
77.         print(f" {a:3d} × {b:3d} = {mult4_booth(a,b):4d}")
78.
```

```

23 def mult4_booth(x: int, y: int) -> int:
24     A = (combo >> 5) & 0x1f
25     Q = (combo >> 1) & 0xF
26     Q_1 = combo & 1
27
28     # combine and sign-extend to Python int
29     result = ((A & 0x1F) << 4) | Q
30     result &= 0xFF
31     if result & 0x80:
32         result -= 0x100
33     return result
34
35 if __name__ == "__main__":
36     # Demo of both multipliers
37     print("Unsigned shift-add (0..15 x 0..15):")
38     for a, b in [(3,6), (7,7), (15,15)]:
39         print(f"{a:2d} x {b:2d} = (mult4_shiftadd(a,b):3d)")
40
41     print("\nBooth's algorithm (-8..+7 x -8..+7):")
42     for a, b in [(-7,3), (7,-3), (-8,7), (-8,-8)]:
43         print(f"{a:3d} x {b:3d} = (mult4_booth(a,b):4d)")
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78

```

(base) PS C:\Users\anika\Documents\00 SFBU Spring 25\EE 488 Computer Architecture\Assignments\A6\alu_project & C:\Users\anika\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\anika\Documents\00 SFBU Spring 25\EE 488 Computer Architecture\Assignments\A6\alu_project\mult4.py"

Unsigned shift-add (0..15 x 0..15):

3 x 6 = 18

7 x 7 = 49

15 x 15 = 225

Booth's algorithm (-8..+7 x -8..+7):

-7 x 3 = -21

7 x -3 = -21

-8 x 7 = -56

-8 x -8 = 64