WELLESLEY

Practice problems

For Exam 1: HW

Exam 1: Computational Building Blocks (HW) + Memory

Lectures

- Digital Logic
- Data as Bits
- Integer Representation
- Combinational Logic
- Arithmetic Logic
- Sequential Logic
- A Simple Processor
- Programming with memory

Labs

- 1: Transistors to Gates
- 2: Data as Bits
- 3: Combinational Logic & Arithmetic
- 4: ALU & Sequential Logic
- 5: Processor Datapath
- 6: Pointers in C

Topics

- Transistors, digital logic gates
- Data representation with bits, bit-level computation
- Number representations, arithmetic
- Combinational and arithmetic logic
- Sequential (stateful) logic
- Computer processor architecture overview
- Basic C programming with pointers

Assignments

- Gates
- Zero
- Bits
- Arch
- Pointers

Mid-semester Exam 1: HW

March 9

Short answer practice problems:

ex

1. How does a D-latch differ in behavior from a D-flip-flop?

2. How are instructions stored in the HW ISA? How does the HW ISA processor know what instruction to execute next?

3. How many bits are needed to choose a register if the register file has 32 entries?

4. What does it mean for a gate to be universal?

Short answer practice problems: solution

ex

1. How does a D-latch differ in behavior from a D-flip-flop?

The output of a D-latch changes based on input D the entire time input C = 1. The output of a D flip-flop changes based on input D only when the input C goes from 1 to 0 (falling edge).

2. How are instructions stored in the HW ISA? How does the HW ISA processor know what instruction to execute next?

Each instruction is encoded in 16 bits (as outlined in the instruction encoding table) and stored in a separate, byte-addressable instruction memory. The PC registrar holds the address of the next instruction and is incremented by 2 in the processor loop.

3. How many bits are needed to choose a register if the register file has 32 entries?

5. To choose between N options, we need $\log_2 N$ bits. $2^5 = 32$.

4. What does it mean for a gate to be universal?

Any circuit or boolean function can be implemented using only that gate.

Bit manipulation practice problem

ex

```
/*
 * absVal - Return the absolute value of x
 *   Examples: absVal(-1) = 1
 *             absVal(240) = 240
 *   You may assume -TMax <= x <= TMax
 *   Legal ops: ! ~ & ^ | + << >>
 */
int absVal(int x) {









}
```

Bit manipulation practice problem: solution 1

ex

```
/*
 * absVal - Return the absolute value of x
 *   Examples: absVal(-1) = 1
 *             absVal(240) = 240
 *   You may assume -TMax <= x <= TMax
 *   Legal ops: ! ~ & ^ | + << >>
 */
int absVal(int x) {
    int mask = x >> 31; // All 0's if positive, all 1's if negative
    return (x & ~mask) | ((~x + 1) & mask);
}
```

Bit manipulation practice problem: solution 2

ex

```
int absVal(int x) {
    // All 0's if positive, all 1's if negative
    int mask = x >> 31;

    // XOR with mask:
    //   no-op if mask is all 0's (0 XOR 0 = 0, 0 XOR 1 = 1)
    //   ~x if if mask is all 1's (0 XOR 1 = 1, 1 XOR 1 = 0)
    // Subtract mask:
    //   no-op if mask is all 0's
    //   +1 (subtract -1 = add 1) if mask is all 1's
    // Together:
    //   no-op if x is positive
    //   ~x + 1 = -x if x is negative
    return (x ^ mask) - mask;
}
```

Bit addition practice problem

ex

What is the result of the following computation on 8-bit two's complement numbers?

```
0b110100101 + 0b011001111
```

Does it overflow? Justify your answer without converting to binary numbers.

Consider the same computation on unsigned numbers. What is the result? Does it overflow?

Bit addition practice problem: solution

ex

What is the result of the following computation on 8-bit two's complement numbers?

```
  1 1     1 1 1 1
  110100101
+ 011001111
-----------
  001110100
```

Does not overflow. Justification:

- Inputs have different sign bits (overflow when sign bits are the same and output sign bit differs)
- OR: Carry in and carry out of the most significant bit are the same

Unsigned: same sum (001110100).

- Overflows because carry out of the most significant bit is dropped.

Building block choice practice problem

ex

Draw a circuit to implement a switching network. If S=1, the network is in pass-through mode: C=A and D=B . If S=0 , the network is in crossing mode: C=B , and D=A .

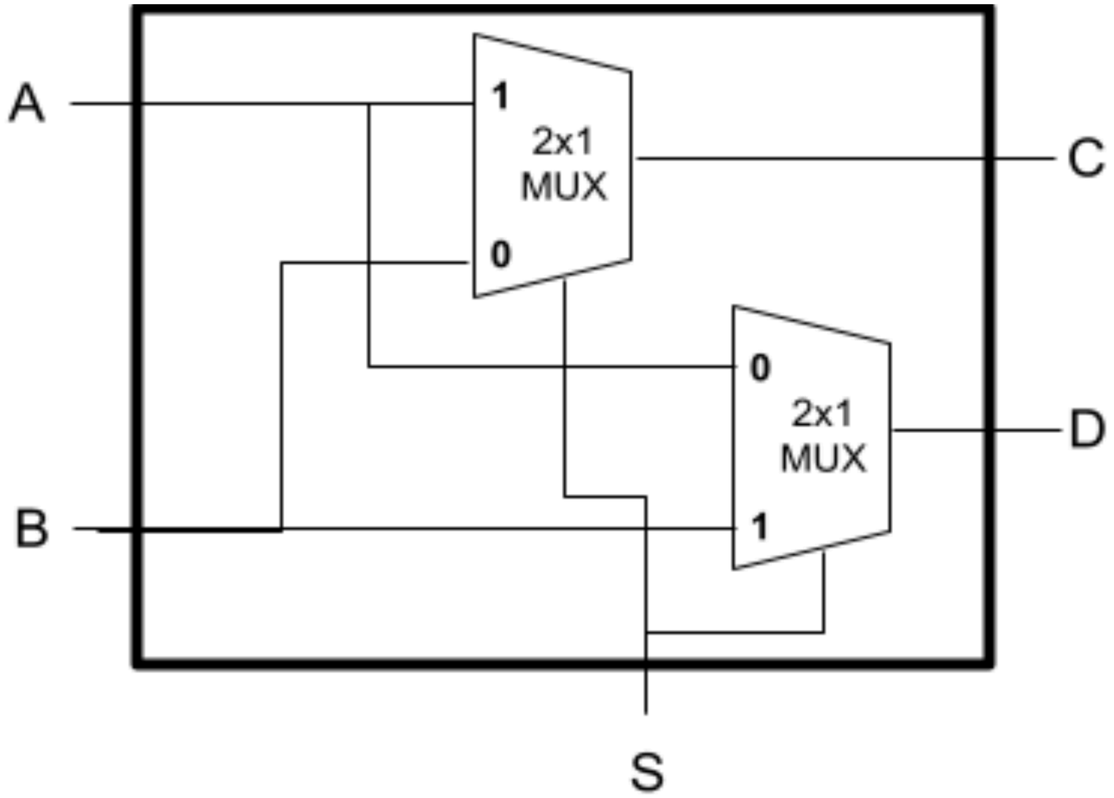
Use the most reasonable combinational building blocks or gates.

Building block choice practice problem: solution

ex

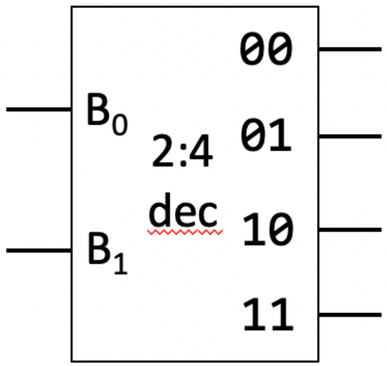
Draw a circuit to implement a switching network. If S=1, the network is in pass-through mode: C=A and D=B . If S=0 , the network is in crossing mode: C=B , and D=A .

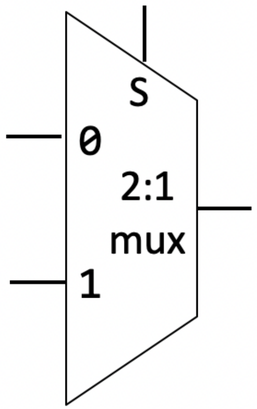
Use the most reasonable combinational building blocks or gates.



Decoder + mux practice problem

ex



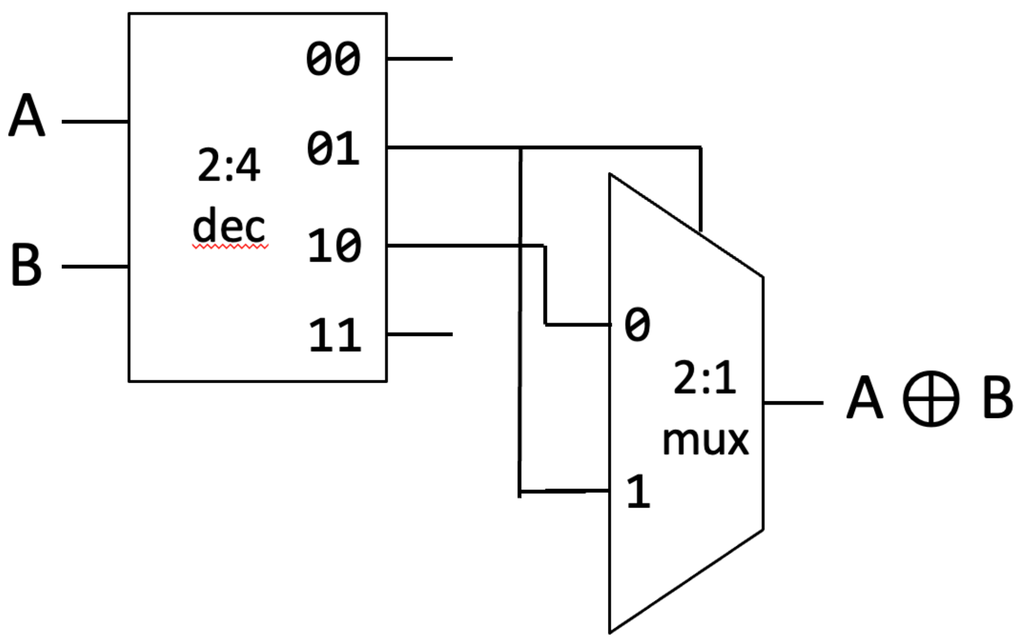


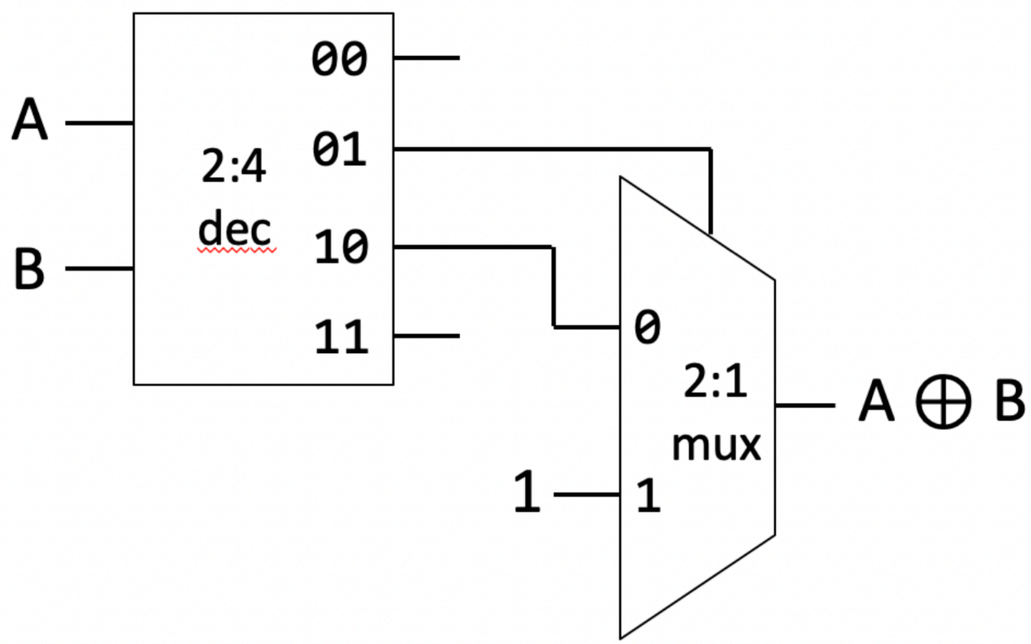
Use **one** 2:4 decoder and **one** 2:1 mux to implement A XOR B

Decoder + mux practice problem: solution

ex

Use **one** 2:4 decoder and **one** 2:1 mux to implement A XOR B



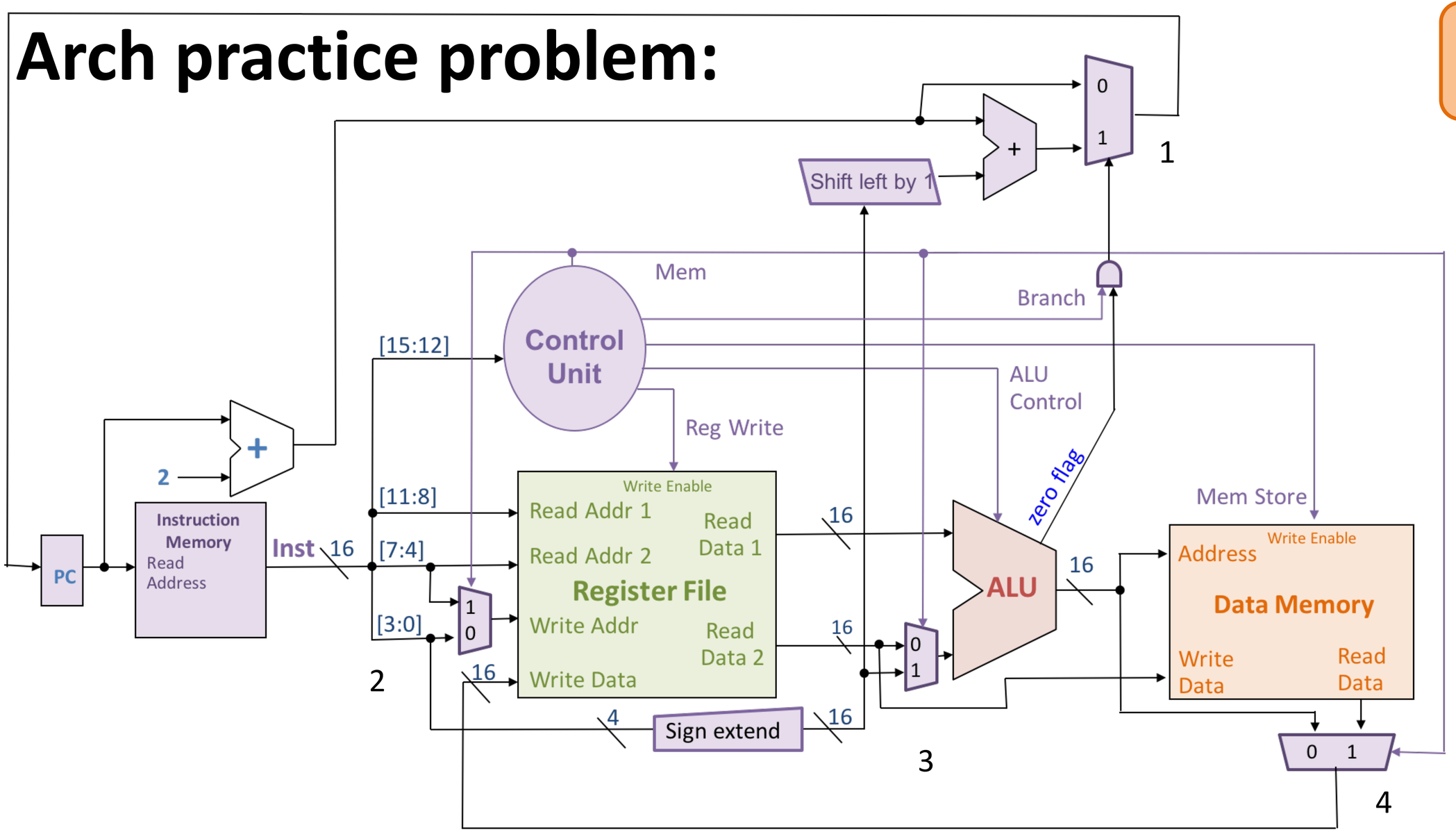


Note: these are just two of many valid solutions.

Insight: XOR should be true if AB is either 01 OR 10. We can use the MUX to implement OR on those two output lines of the decoder.

HW Arch practice problem:

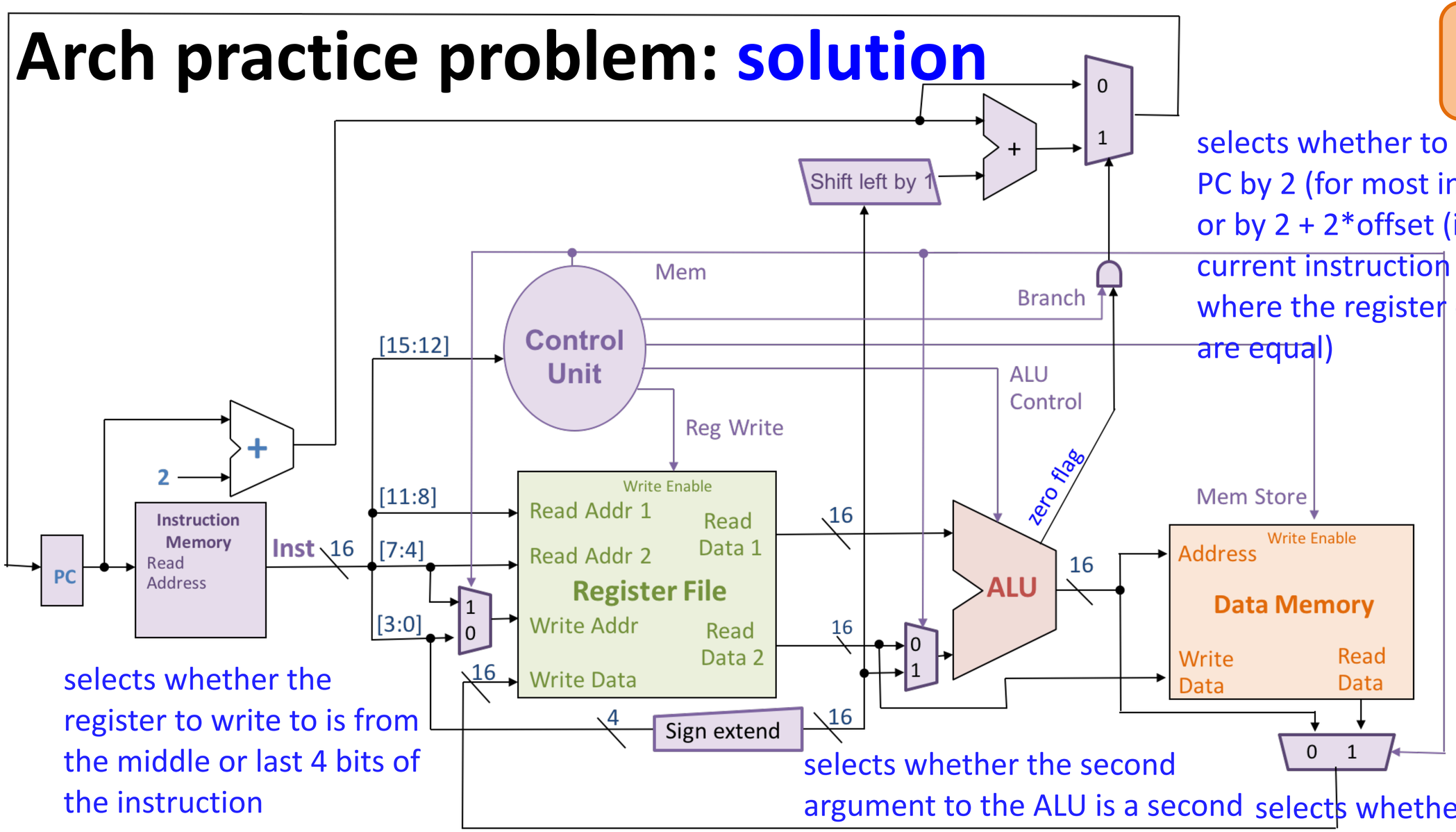
ex



1. What is the purpose of each of the 4 mux components in the HW architecture above?

HW Arch practice problem: solution

ex



selects whether to update the PC by 2 (for most instructions) or by 2 + 2*offset (if the current instruction is a BEQ where the register contents are equal)

selects whether the register to write to is from the middle or last 4 bits of the instruction

selects whether the second argument to the ALU is a second register's data or from sign-extending an offset

selects whether to write to the register file either the ALU result or a word from memory

C programming with pointers

ex

Write the following C function:

`double_string` should take in a null-terminated C string and return a newly-allocated null-terminated string that contains the same string, repeated twice in sequence. You do not need to check the result of `malloc`. You may use the `strlen` function.

For example, `double_string("abcd") = “abcdabcd"`.

C programming with pointers: solution

ex

```c
char *double_string(char *s) {
    int len = strlen(s);
    char *result = malloc((2*len+1)*sizeof(char));
    char *cursor = result;
    for (int i = 0; i < 2; i++) {
        char *from = s;
        while (*from != '\0') {
            *cursor = *from;
            cursor++;
            from++;
        }
    }
    *cursor = '\0';
    return result;
}
```

```c
// Not necessary for solution, but
// example of using this function
int main(void) {
    char *s1 = double_string("abc");
    // prints “abcabc”
    printf("%s\n", s1);
    free(s1);
    return 0;
}
```