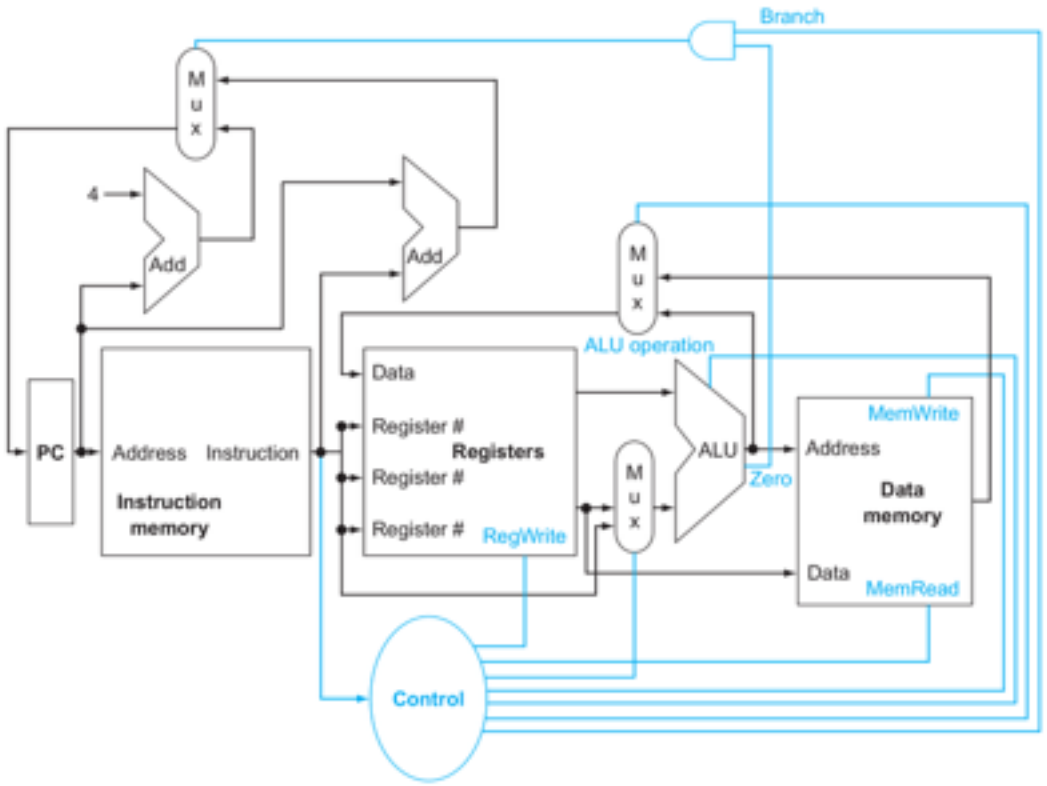

FIGURE 4.2 The basic implementation of the LEGv8 subset, including the necessary multiplexors and control lines. The

Refer to Figure 4.2 from your textbook (copied above) to answer the problems below.

Instructions used in Chapter 4 problems are the subset of LEGv8 as follows

Memory Reference:	LDUR	Rt, [Rn, Offset]	Load Register
	STUR	Rt, [Rn, Offset]	Store Register
Arithmetic/Logic:	ADD	Rd, Rm, Rn	Addition
	SUB	Rd, Rm, Rn	Subtraction
	AND	Rd, Rm, Rn	Bitwise Logical AND
	ORR	Rd, Rm, Rn	Bitwise Logical OR
Branch:	CBZ	Rt, Target	Compare, Branch on Zero

Rm and Rn are source register operands (value is read from the register file)
Rd is a destination register operand (value is written to the register file
Rt is source register operand for CBZ and STUR, destination register operand for LDUR
Offset is a signed constant value added to the base register Rn

Target is normally given as an assembly language label (symbolic name attached to the instruction that we want to branch to if the condition is true.)

Worked Examples

Example 1:

Suppose that the 32-bit machine code instruction on the output of the **Instruction Memory** unit in the figure was created from the assembly language instruction below. Which of the following sets of Control unit outputs will appear?

```
ADD X1, X2, X3      // Assume the following register values before execution:
                    // X1 = 0xff, X2 = 0x0, X3 = 0x4
```

A	Branch = False	RegWrite = True	Zero = False	MemRead = False
B	Branch = False	RegWrite = True	Zero = False	MemRead = True
C	Branch = False	RegWrite = False	Zero = False	MemRead = False
D	Branch = True	RegWrite = False	Zero = False	MemRead = False
E	Branch = False	RegWrite = True	Zero = True	MemRead = False

Correct Answer: A

Branch - this is True if the instruction uses the output of the branch target adder unit for the next PC value, otherwise it is False the next PC value will be PC+4.

RegWrite - this is True if the instruction should overwrite the old value of the destination register with a new value, otherwise it is False because the instruction has no destination register operand.

Zero - this is True if the result out of the ALU is exactly equal to 64 zeros for the instruction, otherwise it is False. Note: the operation performed by the ALU is to pass the register value to the output for a CBZ instruction and addition for a memory access instruction (LDUR or STUR.)

MemRead - this is true if the instruction causes the **Data Memory** unit to read a value from memory, otherwise it is false. It is not desirable to unnecessarily read data from memory and then ignore that data because doing this wastes power and causes inefficient use of data caches. There are no equivalent RegRead signals on the **Registers** unit because it is more efficient to always read two register values and then ignore one or both values if the instruction has one or zero register source operands.

The **ADD X1, X2, X3** instruction is not a branch instruction (Branch = False,) does have a register destination operand (RegWrite = True,) is not a load instruction (MemRead = False,) and the result of adding the contents of registers X2 and X3 is 0x0 + 0x4 = 0x4 which is not 0x0 (Zero = False.)

Problems for Submission

On BlackBoard in an untimed quiz you will be randomly given specific problem numbers from the list below. Work those problems and submit answer letter (A, B, C, D, or E) for each problem in BlackBoard. You may retake the quiz with a new random selection once (for a total of two attempts.) Your score will the the highest of the attempt scores.

Problem 1: E

Same as Example 1 above, but use the following instruction.

```
ADD X1, X2, X3      // Assume the following register values before execution:
                    // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 2: E

Same as Example 1 above, but use the following instruction.

```
SUB X1, X2, X3      // Assume the following register values before execution:
                    // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 3: A

Same as Example 1 above, but use the following instruction.

```
SUB X1, X2, X3      // Assume the following register values before execution:
                    // X1 = 0xff, X2 = 0x1, X3 = 0x2
```

Problem 4: E

Same as Example 1 above, but use the following instruction.

```
AND X1, X2, X3      // Assume the following register values before execution:
                    // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 5: A

Same as Example 1 above, but use the following instruction.

```
AND X1, X2, X3      // Assume the following register values before execution:
                    // X1 = 0xff, X2 = 0x1, X3 = 0x1
```

Problem 6: E

Same as Example 1 above, but use the following instruction.

```
AND X1, X2, X3      // Assume the following register values before execution:
                    // X1 = 0xff, X2 = 0x2, X3 = 0x1
```

Problem 7: E

Same as Example 1 above, but use the following instruction.

```
ORR X1, X2, X3        // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 8: A

Same as Example 1 above, but use the following instruction.

```
ORR X1, X2, X3        // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x1, X3 = 0x0
```

Problem 9: D

Same as Example 1 above, but use the following instruction.

```
CBZ X1, TARGET        // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 10: D

Same as Example 1 above, but use the following instruction.

```
CBZ X1, TARGET        // Assume the following register values before execution:
                      // X1 = 0x01, X2 = 0x0, X3 = 0x0
```

Problem 11: B

Same as Example 1 above, but use the following instruction.

```
LDUR  X1, [X2, #8]    // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 12: B

Same as Example 1 above, but use the following instruction.

```
LDUR  X2, [X3, #16]   // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 13: B

Same as Example 1 above, but use the following instruction.

```
LDUR  X2, [X1, #0]    // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 14: C

Same as Example 1 above, but use the following instruction.

```
STUR  X2, [X3, #16]   // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 15: C

Same as Example 1 above, but use the following instruction.

```
STUR  X1, [X2, #8]    // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```

Problem 16: C

Same as Example 1 above, but use the following instruction.

```
STUR  X2, [X3, #16]   // Assume the following register values before execution:
                      // X1 = 0xff, X2 = 0x0, X3 = 0x0
```