Assign 4 Solutions

4.3.1 LDUR + STUR = 25% + 10% = 35%  
4.3.2 All! That is where all instructions start.  
4.3.3 I + LDUR + STUR+ CBZ + B =76% or all but R instructions.  
4.3.4 Twiddling its thumbs! It is always computed, but used when needed.

4.5 The instructions decodes to STUR X2, [X3, 0x14]

Step 1, figure out what 0xf801 4062 means

1111 1000 **0000 0001 0100** 0000 0110 0010 (red is opcode), (green is address)(op2)(Rn=X3) (Rt =X2]

STUR X2,[X3 ,0x14]

4.5.1 Sign Extend input is 0 0001 0100 (0x0000 0014) becomes 0x0000 0000 0000 0014

Shift Left 2 input is 0x0000 0014 so output is 0x0000 0050

in binary we would have 0000 … 0001 0100 so, shift each bit to the left twice give 0101 0000 =(0x50)

4.5.2 (See page 281) ALUop = 00 use the opcode of 111 1100 0000 to reference the table to extract ALUop.

4.5.3 The new PC is the old +4. This signal goes from the PC, through the “PC+4” adder, through the “branch”, and back to the PC

4.5.4 **Reg2Loc** : Inputs: 1 and 2 and selector 1 ; Output: 2 (The “upper” input to this mux is bits 20:16 from the instruction. In STUR , bits 20:12 are the offset—in this case, 0x14 . Thus, bits 20:16 contain the second hex digit: 0x1 .)

**ALUsrc** : Inputs: Reg[X2] and 0x0000000000000014 ; Output:

0x0000000000000014

**MemToReg** : Inputs: Reg[X2] + 0x14 and <undefined> ; output:

<undefined>/

**Branch**: Inputs: PC+4 and 0x0000000000000050

**4.5.5** ALU inputs: Reg[X3] and 0x0000000000000014

PC + 4 adder inputs: PC and 4

Branch adder inputs: PC and 0x0000000000000050

**4.5.6** Read register 1: 3

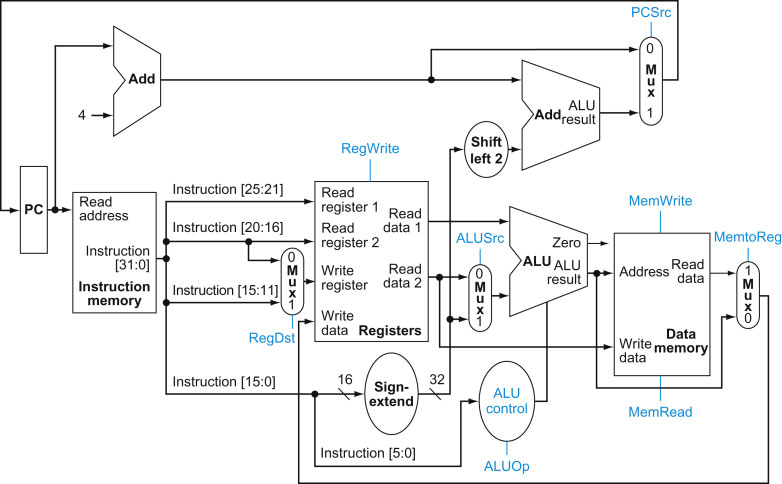
Read register 2: 2

Write register: 2

Write data: undefined. Either Reg[X3]+0x14 , or the undefined output of

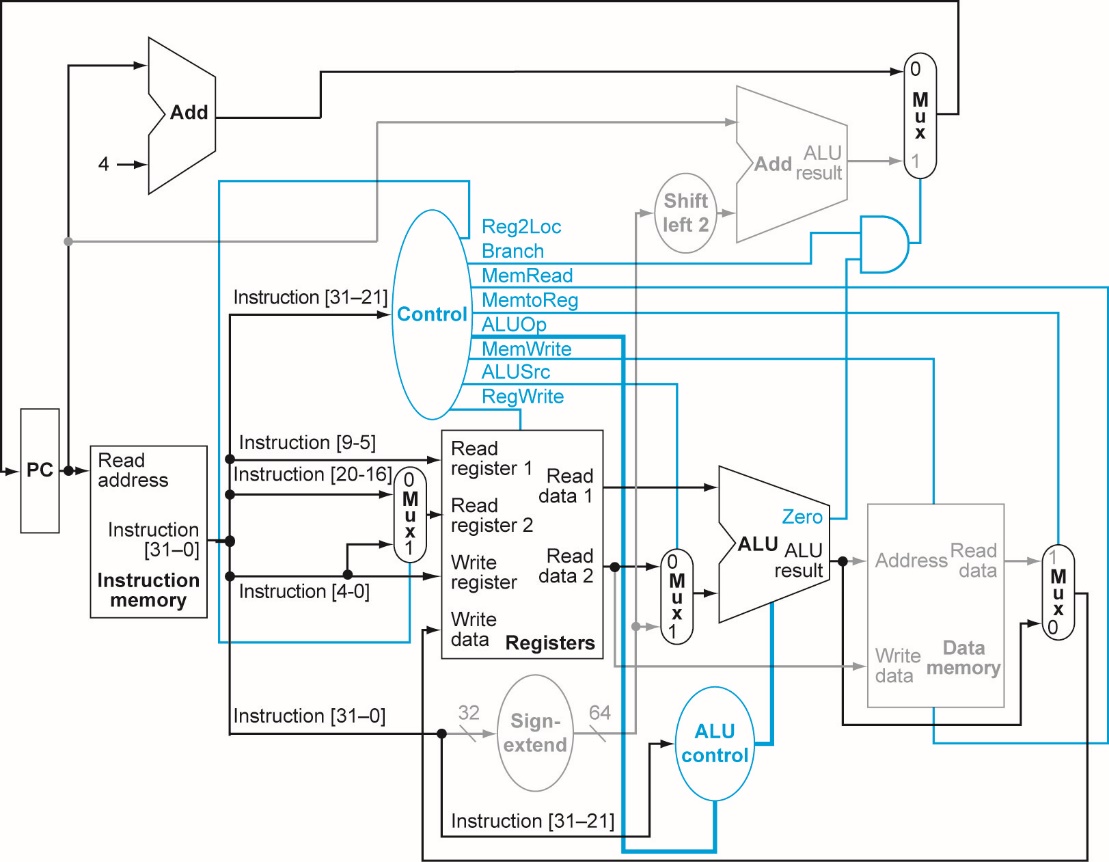
data memory, depending on the value of the MemToReg control wire.

4.7



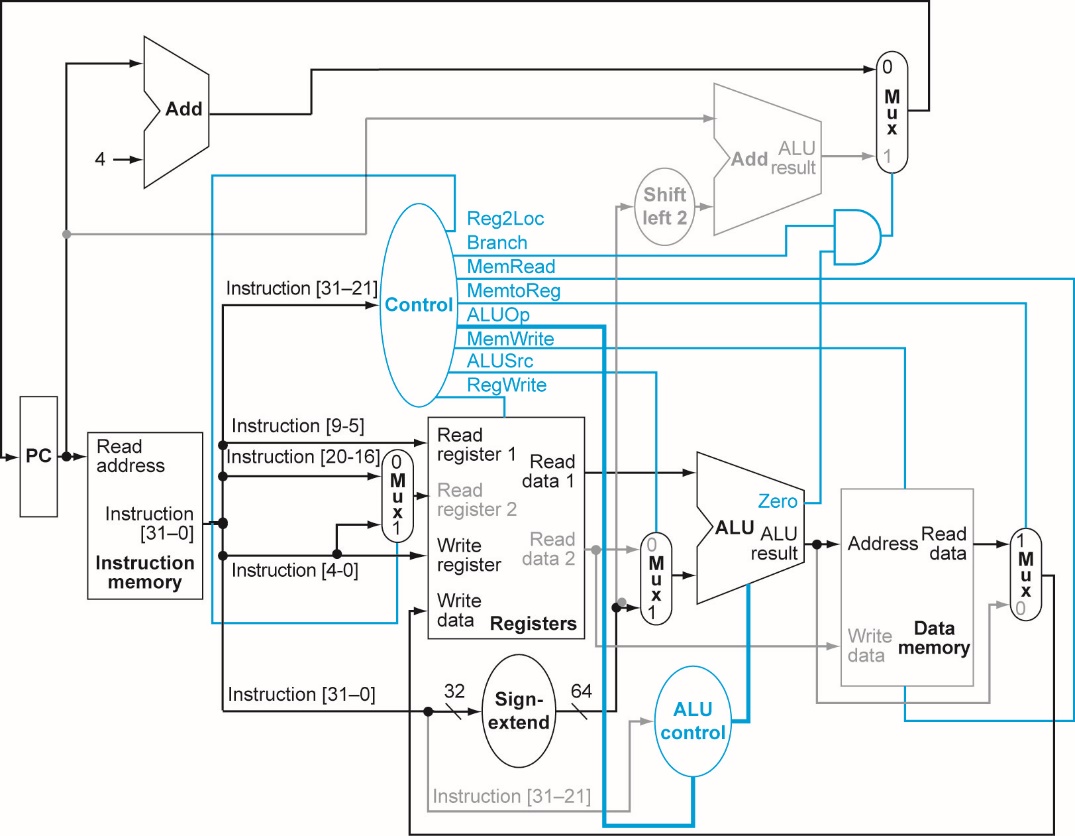
4.7.1 Figure 2.20 shows that instruction bit 28 contains the correct value for the Reg2Loc control line. In general, the value of the Reg2Loc wire is 0 for R-Type instructions, 1 for D-Type and CS-Type instructions, and “don’t care” for everything else. Reg2Loc is also “don’t care” for LSL and LSR because, even though they use an R-Type format, the do not use a second operand.

4.7.2



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i-mem/  d-mem | Register  File | Mux | ALU | Adder | Single Gate | Register Read | Register Setup | Sign Extend | Control |
| 250 ps | 150 ps | 25 ps | 200 ps | 150 ps | 5 ps | 30 ps | 20 ps | 20 ps | 50 ps |

4.7.2 RType: Read PC + IM + MUX+ Reg File + MUX + ALU + MUX + Reg Setup  
30 + 250 + 25 + 150 + 25 + 200 + 20 = 725ps



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i-mem/  d-mem | Register  File | Mux | ALU | Adder | Single Gate | Register Read | Register Setup | Sign Extend | Control |
| 250 ps | 150 ps | 25 ps | 200 ps | 150 ps | 5 ps | 30 ps | 20 ps | 20 ps | 50 ps |

4.7.3 LDUR: Read PC + IM + Reg File + ALU + Data MEM + MUX + Reg Setup = 925ps

4.7.4 STUR : Read PC + IM + Reg File + ALU + Data MEM = 880ps

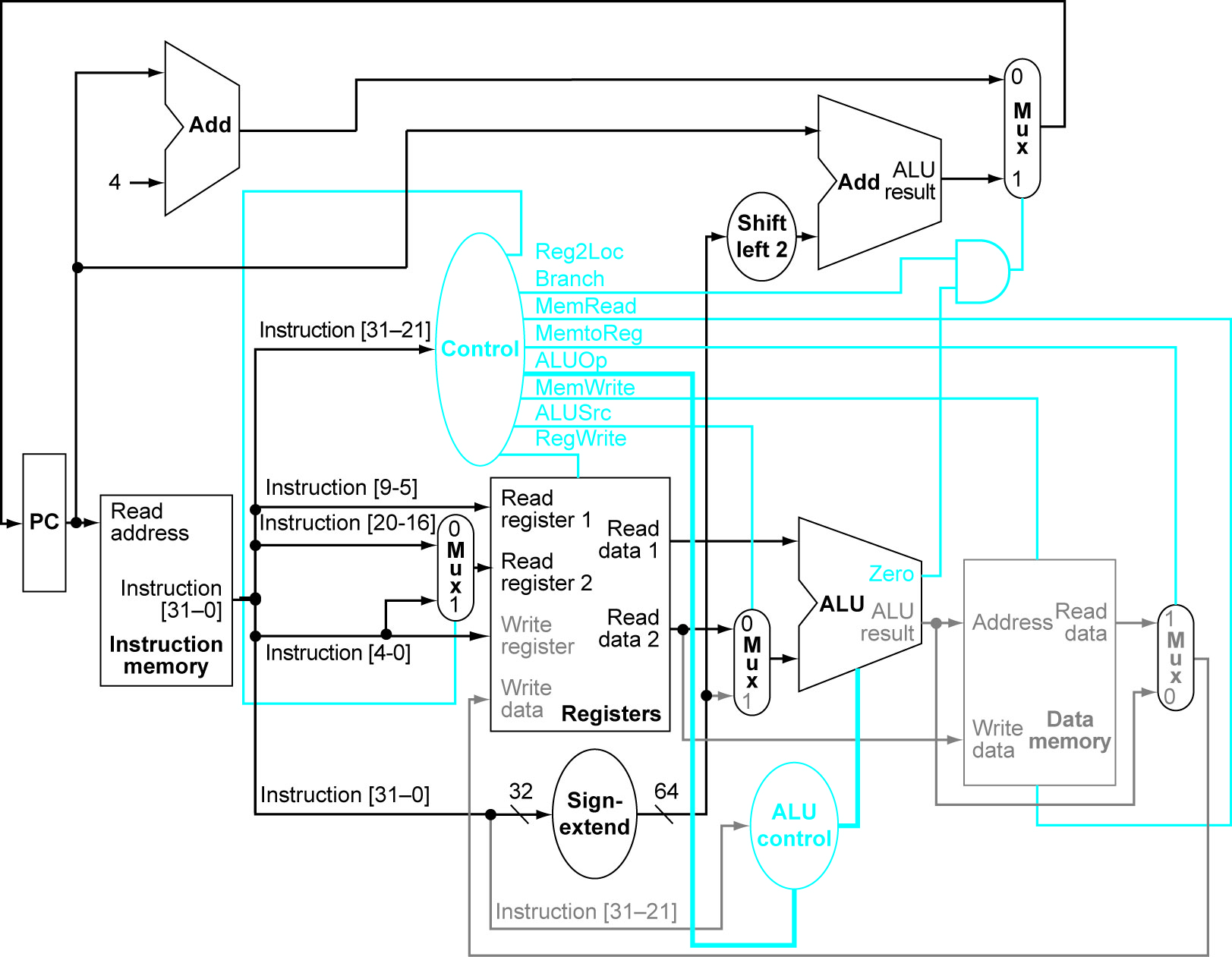


Figure 4.21 ^

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i-mem/  d-mem | Register  File | Mux | ALU | Adder | Single Gate | Register Read | Register Setup | Sign Extend | Control |
| 250 ps | 150 ps | 25 ps | 200 ps | 150 ps | 5 ps | 30 ps | 20 ps | 20 ps | 50 ps |

4.7.5 CBZ: read PC+IM+MUX+Reg File + MUX +ALU + Single Gate + Single Gate (Figure 4.23) + MUX + Reg Setup = 735ps

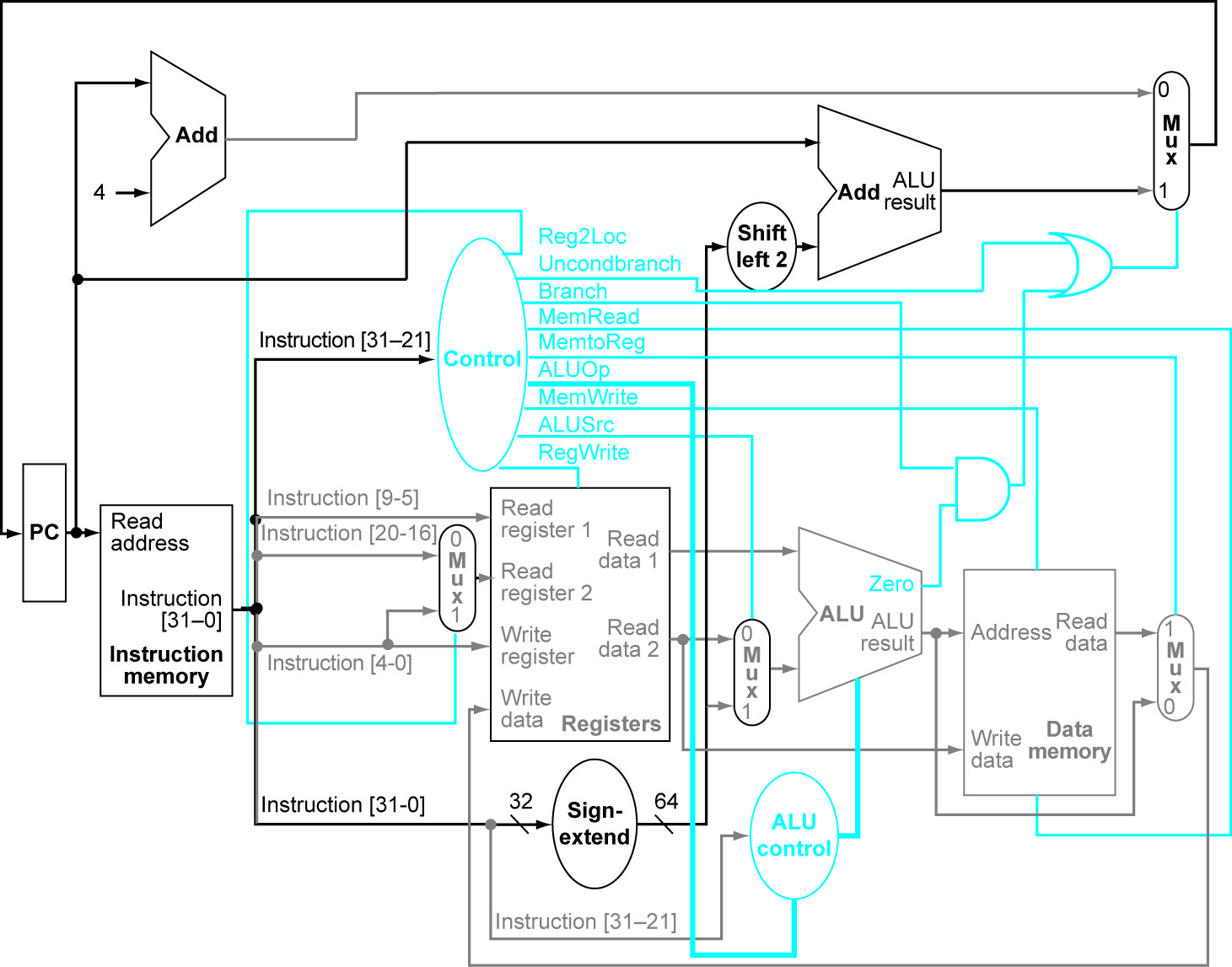


Figure 4.23 ^

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i-mem/  d-mem | Register  File | Mux | ALU | Adder | Single Gate | Register Read | Register Setup | Sign Extend | Control |
| 250 ps | 150 ps | 25 ps | 200 ps | 150 ps | 5 ps | 30 ps | 20 ps | 20 ps | 50 ps |

4.76 B: Read PC + IM + Sign Extend + Adder + MUX + Reg Setup = 525ps

4.7.7 I-Type: Read PC + IM + Reg File + ALU + MUX + Reg Setup = 525 ps (First two muxes not added since sign + mux < Reg File)

4.7.8 925ps

4.16.1 – Non-pipelined is the sum of all stages which is 1250ps

Pipelined is the max of all stages, or in this case 350ps

4.16.2 – Non-Pipelined is 1250ps, pipelined is 350\*5 = 1750ps

Author’s answers

A screenshot of a computer

Description automatically generated



ADDI X1, X2, #5 //X1=11, X2=22  
ADD X3, X1, X2 //X3 = 11 + 22 since X1 is not yet written to register file  
ADDI X4, X1,#15 // X4 = 27 + 15 or is it X4 = 11 + 15?  
//If we assume data forwarding, X1 will have new value of 27 and the correct  
// answer is 27+15 = 42. Without data forwarding we have 11+15 = 26

A screenshot of a computer code

Description automatically generated