Michael Palumbo
CS-281-B
HW-07

4.1

4.1.1

Instruction	RegDst	ALUSrc	Memto Reg	Reg-Write	Mem-Read	Mem Write	Branch	Alu OP1	ALU OP0
lw	0	1	1	1	1	0	0	0	0

4.1.2

PC, Instruction Memory, Control, Registers, ALU, Data-Memory

4.1.3
ALUSrc,MemtoReg, RegWrite, MemRead

4.7
101011 00011 00010 0000000000010100

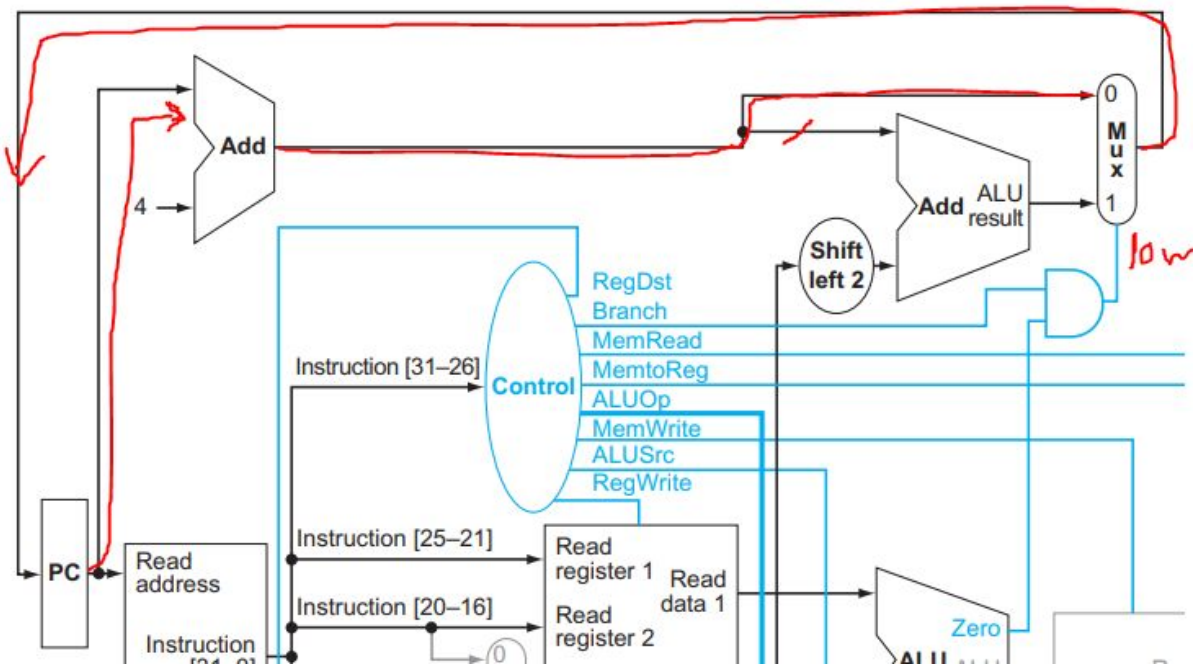
sw $r2 0x0014($r3)
note: 0x0014 is 20 dec

4.7.1
The jump shift left 2 is not really looked at since the Control block tells the Mux that we aren't jumping.
The Sign-extend looks at the imm. which is 0x0014 which just makes it 0x000000014 (so we can add it in the ALU later)

4.7.2
The ALU looks at 2 values. One is the location of rs which is $r3 in this example. That location is then added with the second value the ALU looks at which is the IMM. which is 0x000000014

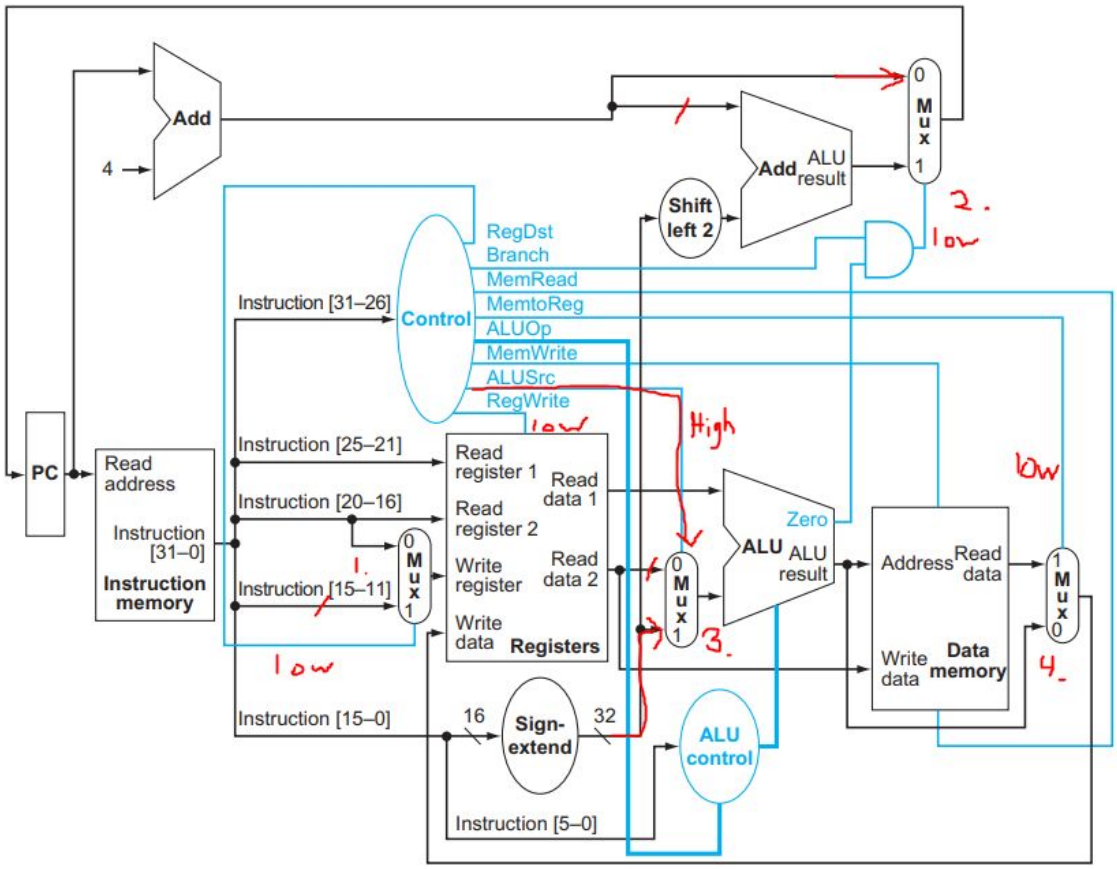
4.7.3

Nothing fancy here happens since we aren't jumping, so we simply add 4 to the memory address and carry on. Not moving to the second add command because Mux is 0



4.7.4

I labeled each Mux we'll look at



1: Mux that goes to the Write Register. Because RegDst is low, the Mux will take input from 20-16 which in our example is 00010 or r2. But since RegWrite is also low, we don't really use it for the write register at all.
2: Mux that is used primarily for the Jump command. We aren't jumping so we simply use the PC+4
3:Mux that goes into the ALU. ALUSrc is high, so we use the imm. value for the ALU
4: The mux command that returns to the Write data. We use the output of the ALU for this, which is $r3 + 0x00000014

4.7.5

As mentioned before the ALU takes $r3 and 0x00000014 which is the rs and immediate location and value
The first Add command, which comes from PC, is simply just moving the PC to the next command. It does this by adding 4. The second add command is not looked at, since we aren't jumping.

4.7.6

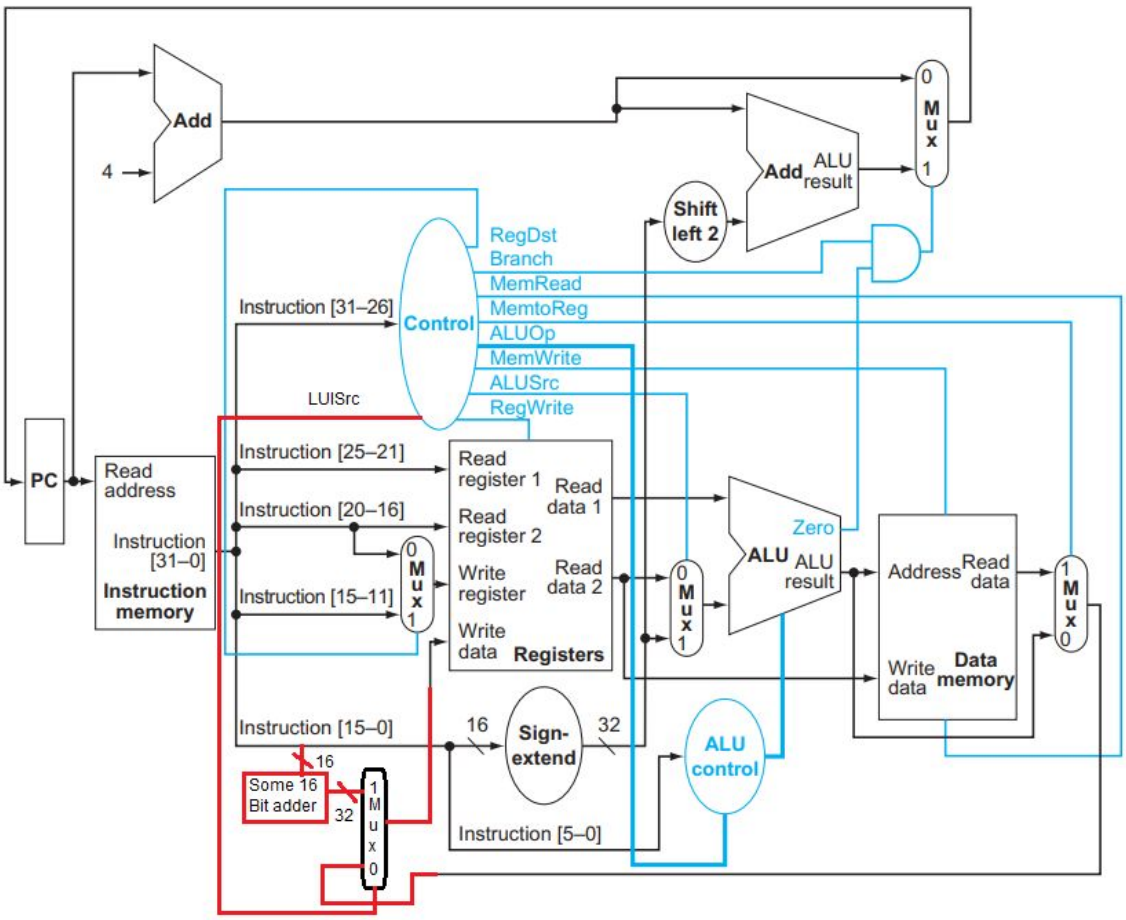
For the registers unit? I assume this means looking at "read register 1" and "read register 2" If so then we have register 1 being rs and register 2 being rt, which is r3 and r2 respectively.

3)

Concept: *li* can also function as LUI if the number is too high. So we need to account for that (in the next question). If the number is small enough, we can ignore rt since we will just give the rs value of immediate. If the number is too big, we probably have to do some ALU work, figure out if we need to use rt in conjunction immediately, and use LUI and ORI. I'm not just how that would be implemented, but that's the concept.

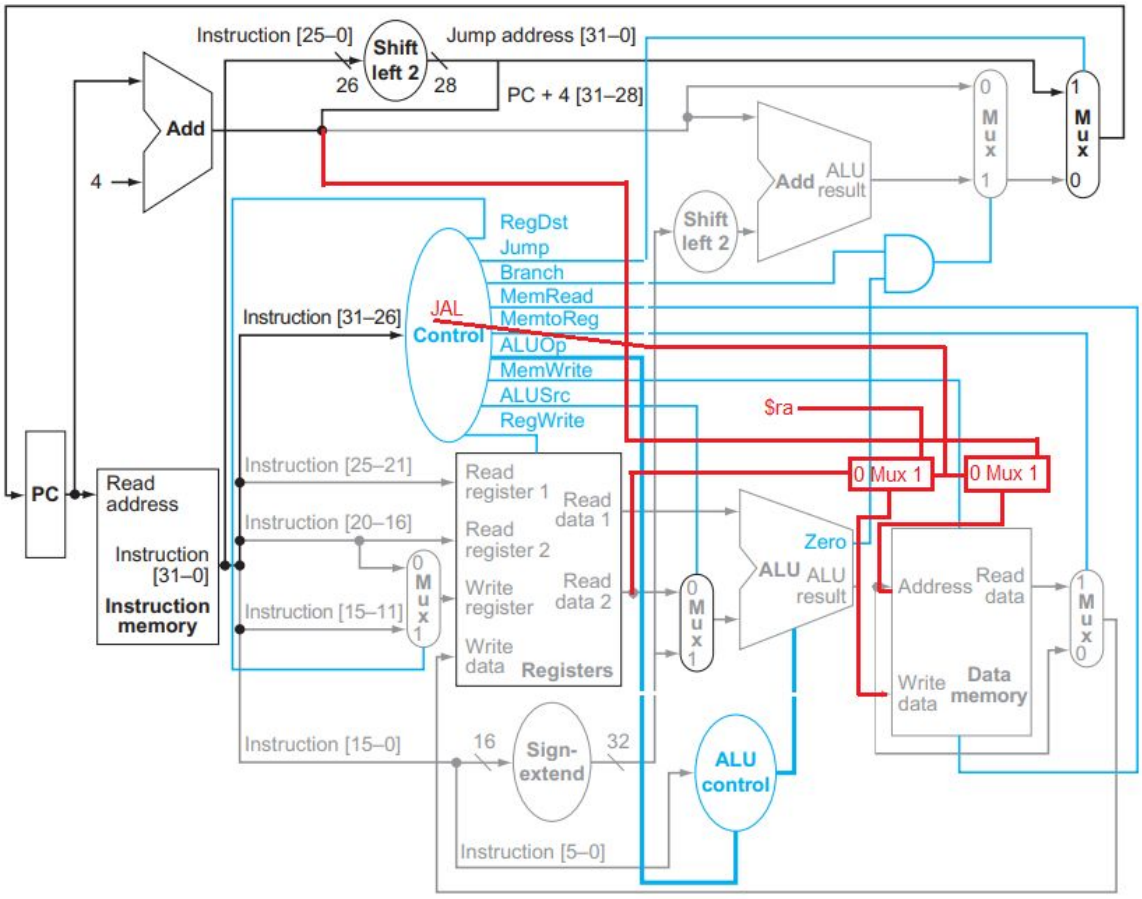
Instruction	Reg Dst	ALUSrc	Memto Reg	Reg-Write	Mem-Read	Mem Write	Branch	Alu OP 1	ALU OP0	JRReg
Li	0	x	x	1	x	0	0	x	x	1

4)



Changes are made in red. In Control, I added LUISrc, this toggles between usings the immediate value directing or using the end result that the pre-existing system would have produced. “Some 16 Bit adder” was my informal way of saying that we are adding 16 bits at the end of the immediate value.

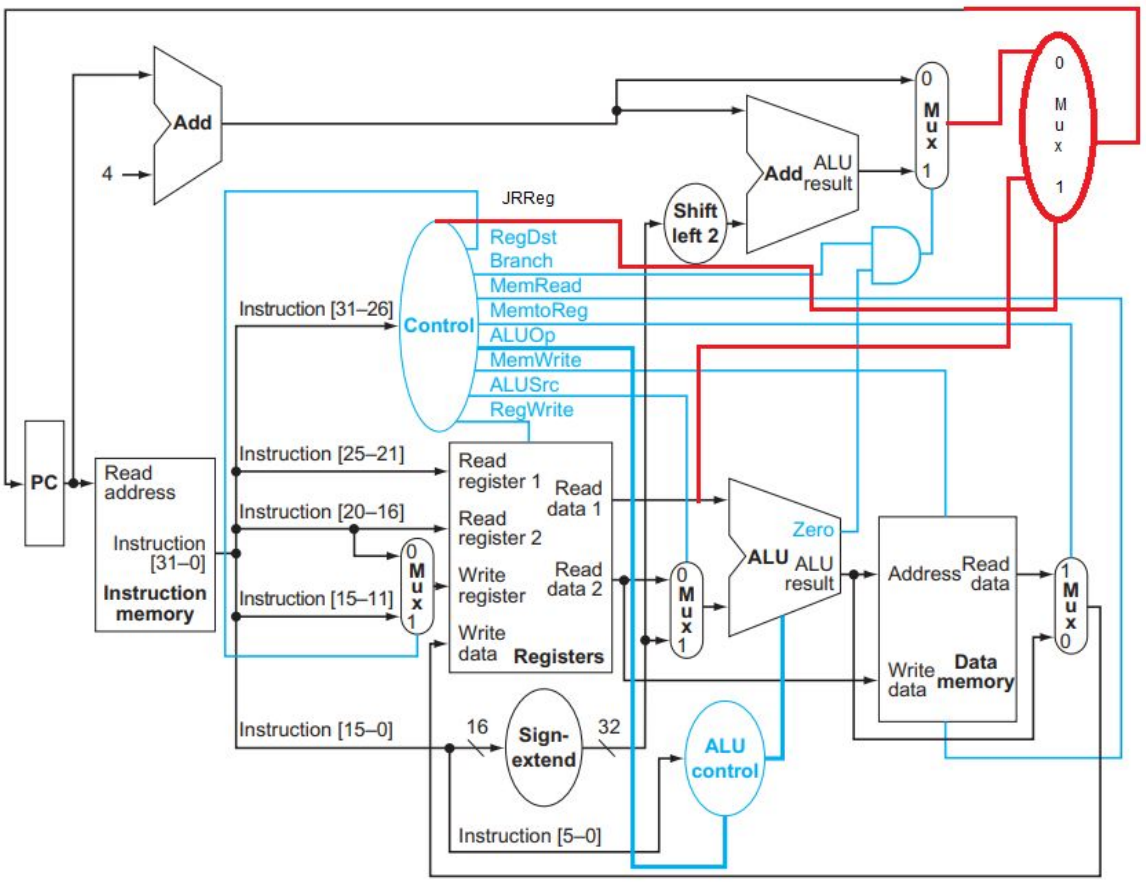
Instruction	Reg Dst	ALUSrc	Memto Reg	Reg-Write	Mem-Read	Mem Write	Branch	Alu OP 1	ALU OP0	JRReg
Lui	0	x	x	1	x	0	0	x	x	1



This is figure 2.24, I usually will use 2.17, since they aren't very different except for the added jump implementation (and I didn't want a greyed out version for the others). In the book they implemented J. They create a Jump switch that has a Mux that will use the immediate instead of the PC+4 address. In Jal, we need to record the memory address of where we just were to $ra. So I creduly added another switch called JAL (there was no space to the side, so I added in the middle). If the switch is high then we look at 2 MUXes, we replace the data with PC+4 and the address to $ra, if it's low, then we use the pre-existing system.

Instruction	Reg Dst	ALUSrc	Memto Reg	Reg-Write	Mem-Read	Mem Write	Branch	Alu OP 1	ALU OP0	JRReg
jal	0	0	0	0	1	1	0	0	0	1

6)



My changes are in red. In control we add a new value it can toggle, JReg. As you can see we use register one which is rs and send it to the mux in the top right. If JRReg is high, then we will use that value of rs for the next address, if not then we will use the systems pre existing methods of finding the new address.

Instruction	Reg Dst	ALUSrc	Memto Reg	Reg-Write	Mem-Read	Mem Write	Branch	Alu OP 1	ALU OP0	JRReg
jr	x	x	x	0	x	0	0	x	x	1