EGC442 HW #4 Dr. Izadi

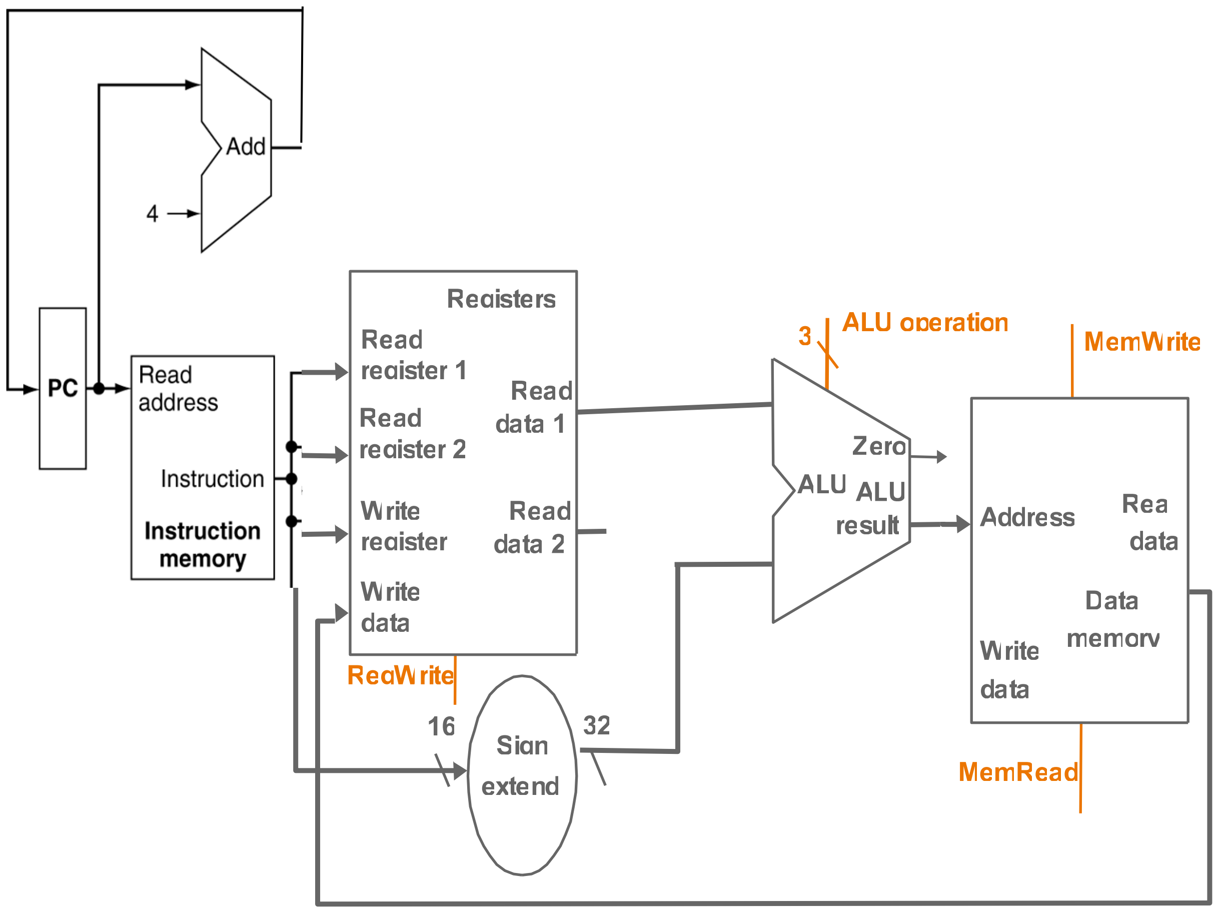
Problem 1 (30 Points)

*lw rt, d16 (rs)*

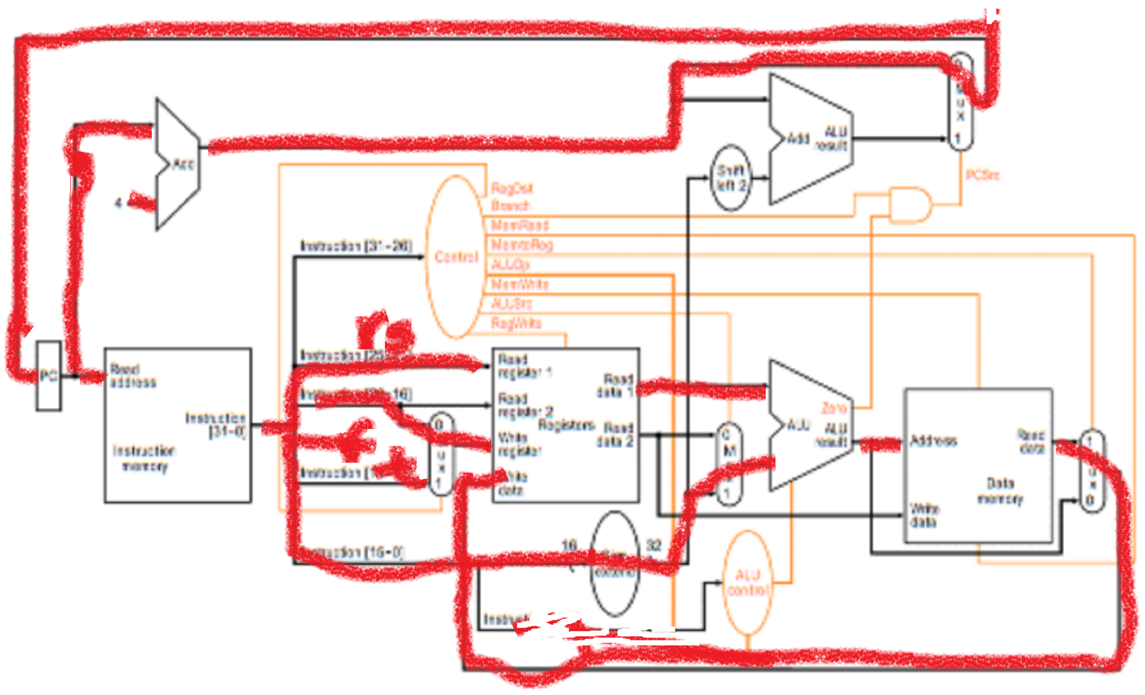
a. Draw only the data path for the instruction

b. Highlight the active components on a printed copy of http://www.engr.newpaltz.edu/~bai/EGC442/f519.pdf

a.



b.



a. The values of the signals are as follows:

| RegWrite | MemRead | ALUMux | MemWrite | ALUop | MemtoReg | Branch |
|---|---|---|---|---|---|---|
| 1 | x | d16 | 0 | add | Mem | 0 |

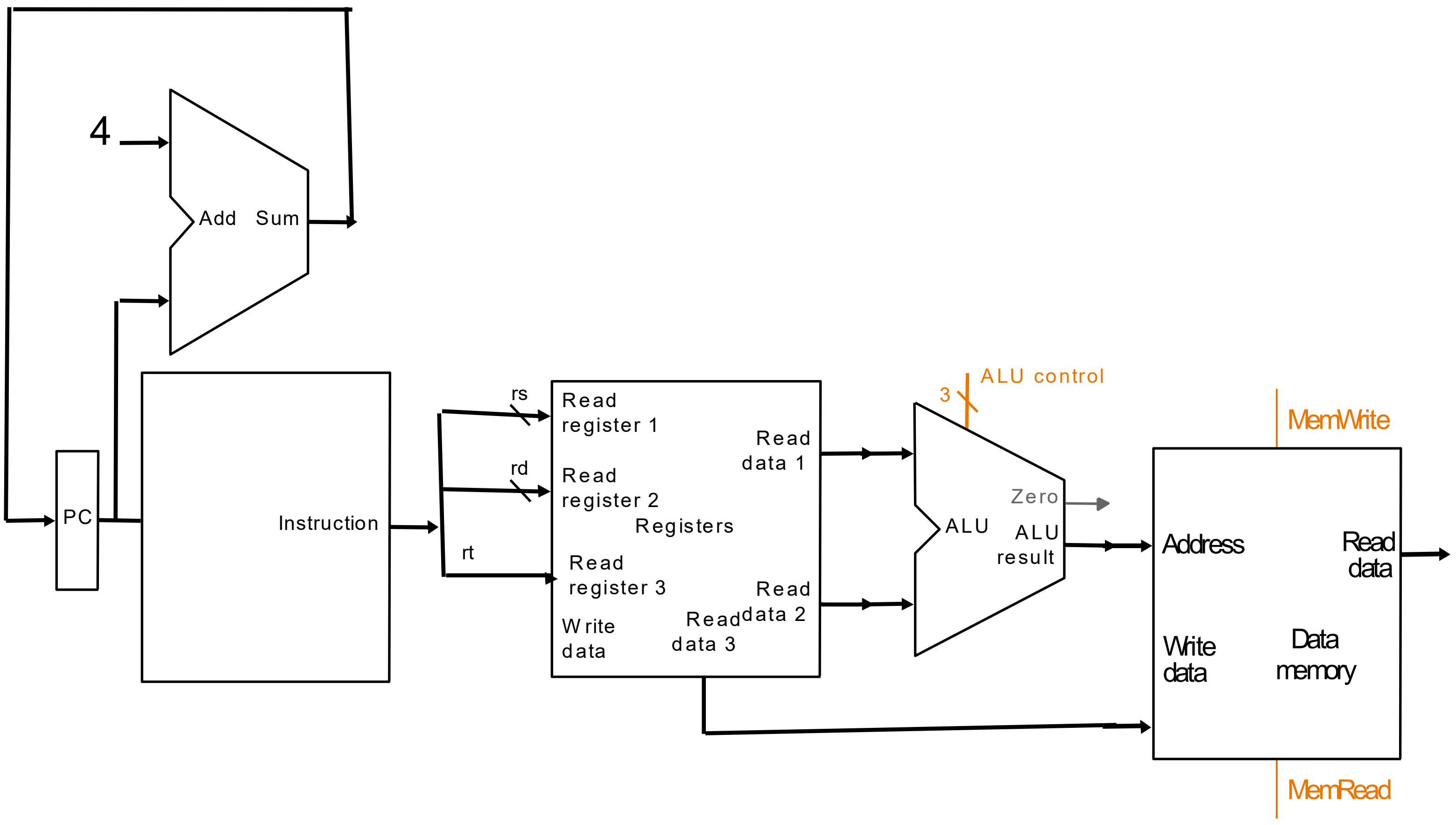
b. All except branch Add unit are active.

c. The resources (blocks) produce outputs, but their outputs are not used for this instruction are branch adder. No resources produce no outputs for this instruction (all units produce outputs)

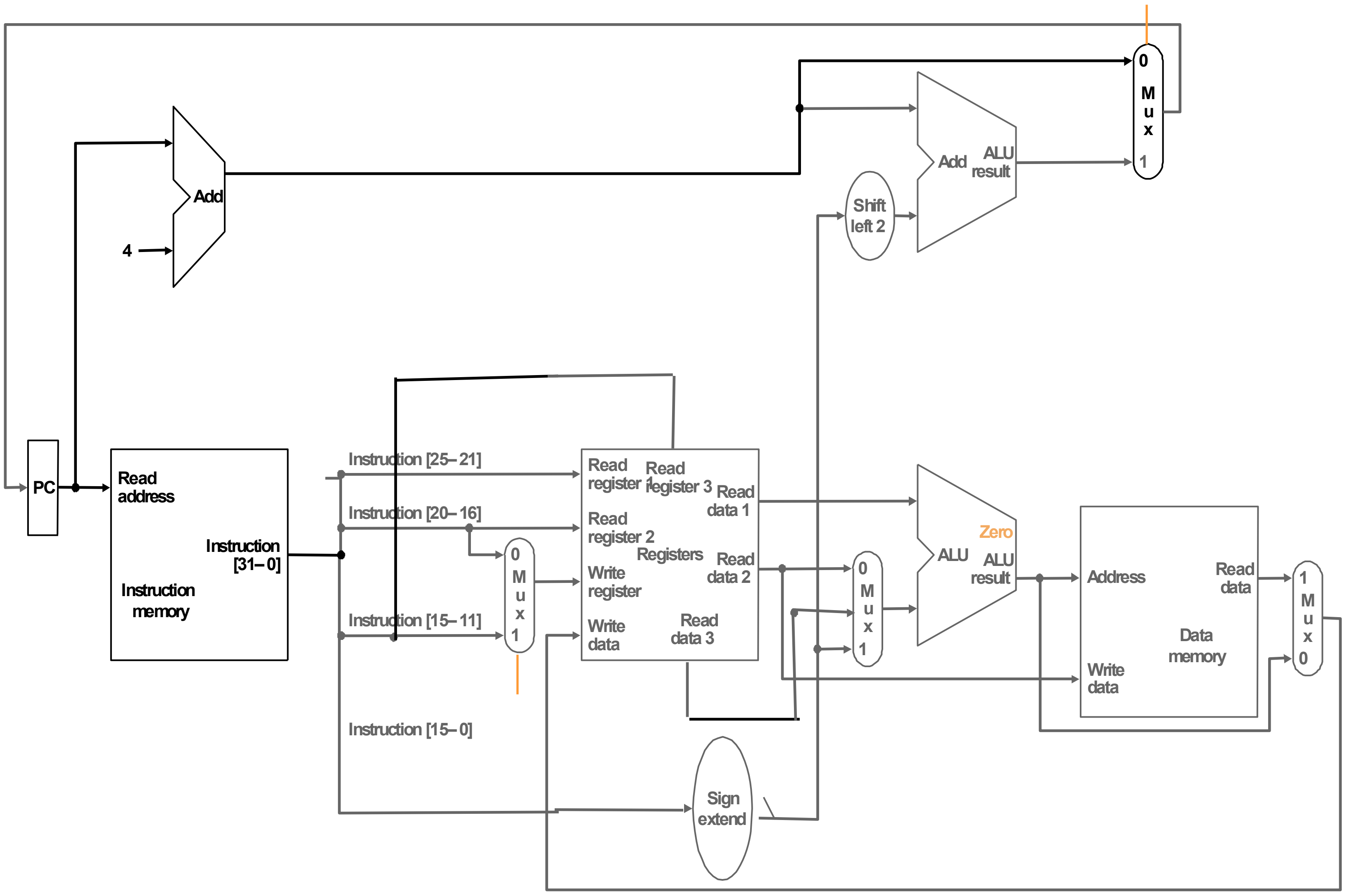
Problem 2 (15 Points)

Reg[rt] → Mem [Reg [rd] + Reg [rs]]
Make your modification on printed copy of
http://www.engr.newpaltz.edu/~bai/EGC442/f519.pdf
And highlight the active components.

**swr Rt, Rd (Rs)**
**Rt -> M[Rs + Rd]**



4
Add
Sum
PC
Instruction
rs
rd
rt
Read register 1
Read register 2
Registers
Read register 3
Write data
Read data 1
Read data 2
Read data 3
3
ALU control
ALU
Zero
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead



0
M
u
x
1
Add
ALU result
Shift left 2
Add
4
PC
Read address
Instruction [31– 0]
Instruction memory
Instruction [25– 21]
Instruction [20– 16]
Instruction [15– 11]
Instruction [15– 0]
0
M
u
x
1
Read register 1
Read register 3
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Read data 3
0
M
u
x
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
M
u
x
0
Sign extend

**a. This instruction uses instruction memory, both register read ports, the ALU to add Rd and Rs together, uses the result as the address of memory where contents of rt is stored. Therefore the used blocks would be:**

- **Instruction memory**
- **Register**
- **ALU**
- **Data Memory**

**b. We need to ability to read rs, rd, and rt simultaneously. This requires modification to our register files resulting in three outputs out of our registers. In addition, we need the ability to add rs and rd. This can be accomplished by adding an input to ALUMux from rd.**

**c. We need to ability to control the ALUMux which chooses between rt and rd.**

Problem 3 (20 Points)

**a. Clock cycle time is determined by the critical path, which for the given latencies happens to be to get the data value for the load instruction: I-Mem (read instruction), Regs (takes longer than Control), Mux (select ALU input), ALU, Data Memory, and Mux (select value from memory to be written into Registers). Th e latency of this path is 400 ps + 200 ps + 30 ps + 120 ps + 350 ps + 30 ps = 1130 ps. 1430 ps (1130 ps + 300 ps, ALU is on the critical path).**

**b. The speedup comes from changes in clock cycle time and changes to the number of clock cycles we need for the program: We need 5% fewer cycles for a program, but cycle time is 1430 instead of 1130, so we have a speedup of (1/0.95)*(1130/1430) = 0.83, which means we actually have a slowdown.**

**c. Th e cost is always the total cost of all components (not just those on the critical path, so the original processor has a cost of I-Mem, Regs, Control, ALU, D-Mem, 2 Add units and 3 Mux units, for a total cost of 1000 + 200 + 500 + 100 + 2000 + 2*30 + 3*10 = 3890.
We will compute cost relative to this baseline. The performance relative to this baseline is the speedup we previously computed, and our cost/**

performance relative to the baseline is as follows:
New Cost: 3890 + 600 = 4490
Relative Cost: 4490/3890 = 1.15
Cost/Performance: 1.15/0.83 = 1.39. We are paying significantly more for significantly worse performance; the cost/performance is a lot worse than with the unmodified processor.

Problem 4 (20 Points)

a.

Op Code 1010 1100 0110 0010 0000 0000 0001 0100
Instruction is sw $2, ($3) 0x14
Sign extend: 0000 0000 0001 0100 =>
0000 0000 0000 0000 0000 0000 0001 0100

Jump address portion of op code (the part of data path which is not active)
1010 1100 0110 0010 0000 0000 0001 0100
So, shifting by 2: 00 0110 0010 0000 0000 0001 0100 00

| Sign-extend | Jump's shift-left-2 |
|---|---|
| 00000000000000000000000000010100 | 0001100010000000000001010000 |

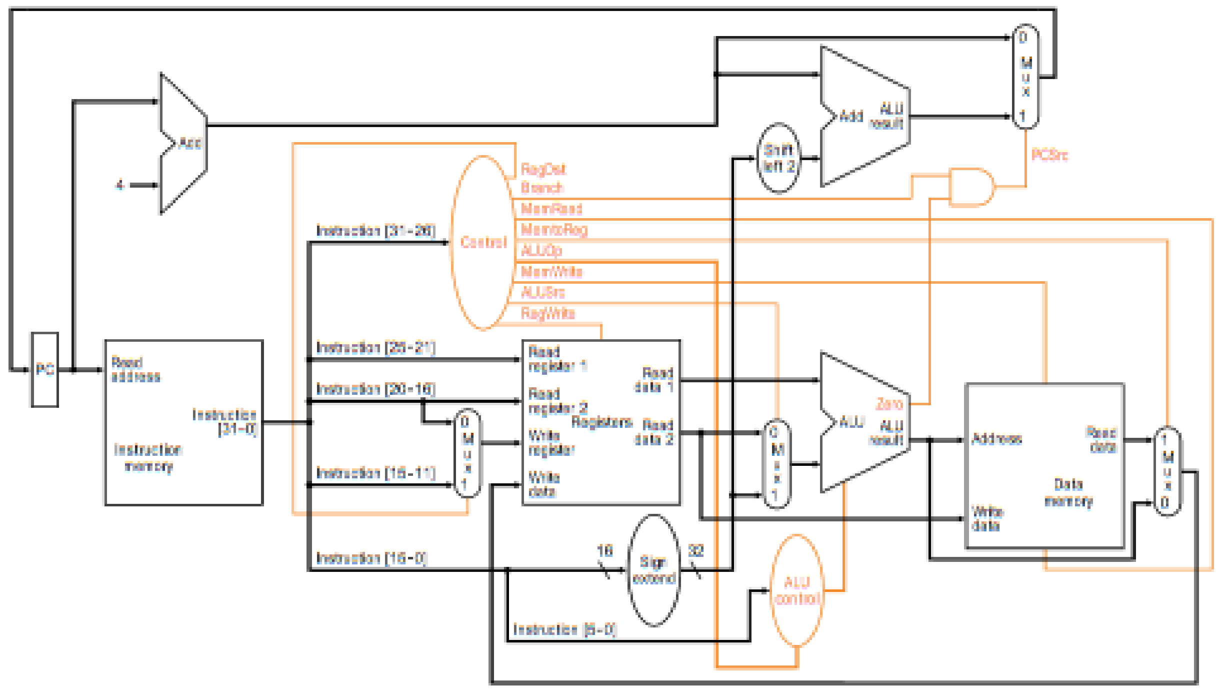
b.

1010 1100 0110 0010 0000 0000 0001 0100

Op code: 101011 = 0x2B ➔ sw
ALU Op for instruction type:
00 = lw, sw
01 = beq,
11 = arithmetic
So, ALUOp = 00

c.

| New PC | Path |
|---|---|
| PC+4 | PC to Add (PC+4) to branch Mux to jump Mux to PC |



d.

101011 00011 00010 0000 0000 0001 0100

Op code Rs = 3 Rt = 2 d16 = 20

| RegDst Mux | ALUSrc Mux | MemorytoReg | Mux PCSrc Mux | Not in the diag |
|---|---|---|---|---|

| WrReg Mux | ALU Mux | Mem/ALU Mux | Branch Mux | Jump Mux |
|---|---|---|---|---|
| 2 or 0 (RegDst is X) | 20 | X | PC+4 | PC+4 |

e.

For the ALU, one input would 20 and the other input would be content of register 3 which is -3.
For the PC add units, one input would be PC and the other would be 4.
Finally, the ALU Branch would shift 20, two times left and add it to PC + 4.

| ALU | Add (PC+4) | Add (Branch) |
|---|---|---|
| -3 and 20 | PC and 4 | PC+4 and 20*4 |

f.

| Read Register 1 | Read Register 2 | Write Register | Write Data | RegWrite |
|---|---|---|---|---|