COE 301 – Computer Organization

Assignment 6 Solution

Single-Cycle Processor Implementation

1. **(5 pts)** Describe the effect that a single stuck-at-0 fault (i.e., the signal is always 0 regardless of what it should be) would have for the signals shown below, in the single-cycle Datapath. Which instructions, if any, will not work correctly? Explain why.

Consider each of the following faults separately:

a) RegWrite = 0

This means that no register can be written in the register file. All R-type instructions with destination register Rd, such as ADD and SUB, and all I-type instructions with destination register Rt, such as ORI and LW, will not work because these instructions will not be able to write their results to the register file.

b) RegDst = 0

This means that RegDst is stuck at Rt and can never be Rd. All R-type instructions with destination register Rd, such as ADD and SUB, will not work because these instructions will not be able to write their results to Rd. Instead, they will write their results to the second source register Rt.

c) ALUScr = 0

This means that ALU is stuck to have its second operand coming from the register file, and can never be the immediate constant encoded inside the instruction. All I-type instructions with a 16-bit immediate constant, such as ORI, BEQ, LW, and SW, will not work because these instructions will not be able to use the 16-bit immediate constant encoded inside the instruction.

d) MemtoReg = 0

This means that the value written back to the register file is always the ALU result, and can never be the value read from data memory. The load instructions, such as LW, will not work because these instructions will not be able to write back the value read from data memory into the register file.

e) Branch = 0

This means that the Branch control signal will never indicate the presence of a branch instruction. The branch instructions, such as BEQ, will not work because these instructions will not be able to branch (branch is never taken) even when the branch condition is true.

2. **(5 pts)** Repeat question 1 but this time consider stuck-at-1 faults (the signal is always 1).

a) RegWrite = 1

This means that all instructions will write 'some value' in the register file. Store instructions, such as SW, branch instructions, such as BEQ, and jump instructions will not work properly. These instructions are not supposed to modify the register file. However, when RegWrite = 1 then Store, Branch, and Jump instructions will modify the register file.

b) RegDst = 1

This means that RegDst is stuck at Rd and can never be Rt. All I-type instructions with destination register Rt, such as ORI and LW, will not work because these instructions will not be able to write their results to Rt. Instead, they will write their result to some arbitrary register Rd whose number is encoded in the upper 5 bits of immediate constant.

c) ALUScr = 1

This means that the ALU is stuck to have its second operand coming from the immediate constant encoded inside the instruction. All R-type instructions with a second source register Rt, such as ADD and SUB, will not work because these instructions will not be able to use the second operand read from the register file. Instead the datapath will assume that the R-type instruction was an I-type instruction, and will assume the second ALU operand to be an immediate constant.

d) MemtoReg = 1

This means that the value written back to the register file is always the value read from data memory, and can never be the ALU result. All instructions that produce an ALU result, such as ADD, SUB, and ORI, will not work because these instructions will not be able to write back the ALU result into the register file.

e) Branch = 1

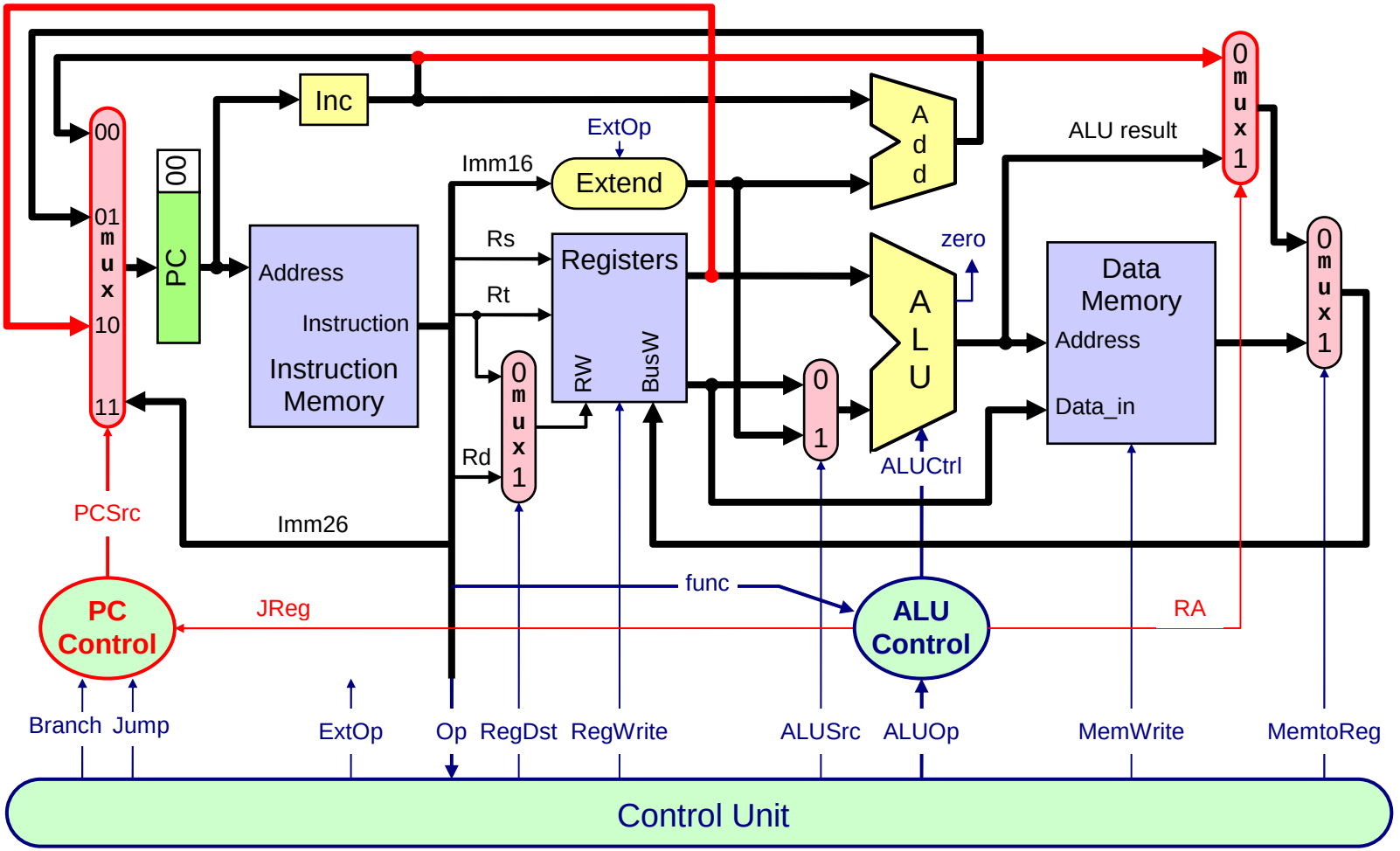
This means that the Branch control signal will always indicate the presence of a branch instruction. All non-branch instructions, such as ADD, SUB, ORI, LW, and SW, will not work because these instructions will be able to branch. However, these instructions were never supposed to branch.

3. **(6 pts)** We wish to add the instruction **`jalr`** (jump and link register) to the single-cycle datapath. Add any necessary datapath and control signals and draw the result datapath. Show the values of the control signals to control the execution of the **`jalr`** instruction.

The jump and link register instruction is described below:

```
jalr rd, rs  # rd = pc + 4 , pc = rs
```

op[6] = 0	**rs[5]**	**0**	**rd[5]**	**0**	**func[6] = 9**


00
01
mux
10
11
PCSrc
PC
00
Inc
Address
Instruction
Instruction Memory
Imm26
Imm16
ExtOp
Extend
Rs
Rt
Rd
Registers
RW
BusW
0
mux
1
Add
ALU
zero
ALUCtrl
ALU result
Data Memory
Address
Data_in
RA
JReg
func
PC Control
ALU Control
Branch
Jump
ExtOp
Op
RegDst
RegWrite
ALUSrc
ALUOp
MemWrite
MemtoReg
Control Unit

The necessary changes to the datapath and control are shown in red.

For the datapath, we need a bigger 4-input multiplexer at the input of the PC. The first input is used to increment the PC. The second input is used for taken branches, where the branch target is PC-relative. The third input is used to jump register, where the input to the PC comes from a general-purpose register, and the fourth input is used for jump instructions.

Our focus here is on the implementation of the JALR instruction. Part of this instruction is to jump to register 'Rs', so we must ensure that we add a path from the output of register Rs (first ALU input) back to the PC multiplexer input. This path is shown in red for clarity.

We need a 'JReg' (Jump Register) control signal to jump according to the value of register 'Rs'. This signal is best generated by the ALU control logic, since it depends on the function field. This control signal is shown in red and is used as an input to the 'PC Control' logic. The 'PC Control' logic generates the 'PCSrc' control signal, which is used to control the 4-input multiplexer at the input of the PC. When JReg is equal to '1', PCSrc will be '10' to select the value of register Rs as input to PC.

We also need to store PC+4 in register Rd. To accomplish this, we need another multiplexer (shown in red) to select between the incremented PC and the ALU result to be placed on BusW. Again here, multiple solutions exist.

We must add a path from the output of the incremented PC to the input of this new multiplexer. This path is shown in red in the above diagram. Another control signal called 'RA' (Return Address) selects between the incremented PC and the ALU result. The MemtoReg multiplexer selects between the output of the 'RA' multiplexer and the Data Memory output to place on BusW.

The main control signals for the JALR instruction are the same for other R-type instructions, such as ADD and SUB. These control signals are shown in the table below:

Instr.	RegDst	RegWrite	ALUSrc	ALUOp	MemWrite	MemtoReg	Branch	Jump
JALR	Rd = 1	1	Rt = 0	R-type	0	0	0	0

The ALU Control signals for the JALR instruction are shown below. JReg = 1 and RA = 1. ALUCtrl is a don't care.

ALUOp	func	JReg	RA	ALUCtrl
R-type	JALR	1	1	X

Note to my grader:

Please make sure you understand the student solution because there is no unique solution to this problem.

4. (4 pts) We want to compare the performance of a **single-cycle CPU design** with a **multi-cycle CPU**. Suppose we add the multiply and divide instructions. The operation times are as follows:

Instruction memory access time = 190 ps, Data memory access time = 190 ps
Register file read access time = 150 ps, Register file write access = 150 ps
ALU delay for basic instructions = 190 ps, ALU delay for multiply or divide = 550 ps

Ignore the other delays in the multiplexers, control unit, sign-extension, etc.

Assume the following instruction mix: 30% ALU, 15% multiply & divide, 30% load & store, 15% branch, and 10% jump.

a) What is the total delay for each instruction class and the clock cycle for the single-cycle CPU design.

Instruction Class	Instruction Memory	Register Read	ALU	Data Memory	Register Write	Total Delay
Basic ALU	190 ps	150 ps	190 ps		150 ps	680 ps
Mul & Div	190 ps	150 ps	550 ps		150 ps	1040 ps
Load	190 ps	150 ps	190 ps	190 ps	150 ps	870 ps
Store	190 ps	150 ps	190 ps	190 ps		720 ps
Branch	190 ps	150 ps	190 ps			530 ps
Jump	190 ps	150 ps				340 ps

Clock cycle = max delay = 1040 ps.

b) Assume we fix the clock cycle to 200 ps for a multi-cycle CPU, what is the CPI for each instruction class and the speedup over a fixed-length clock cycle?

CPI for Basic ALU = 4 cycles

CPI for Multiply & Divide = 6 cycles (ALU takes 3 cycles)

CPI for Load = 5 cycles

CPI for Store = 4 cycles

CPI for Branch = 3 cycles

CPI for Jump = 2 cycles

I am going to assume that 30% for load and store is divided equally as 15% and 15%, because the problem does not separate their percentages.

Average CPI = 0.3 * 4 + 0.15 * 6 + 0.15 * 5 + 0.15 * 4 + 0.15 * 3 + 0.1 * 2 = 4.1

Speedup of multi-cycle over single-cycle = (1040 * 1) / (200 * 4.1) = 1.27