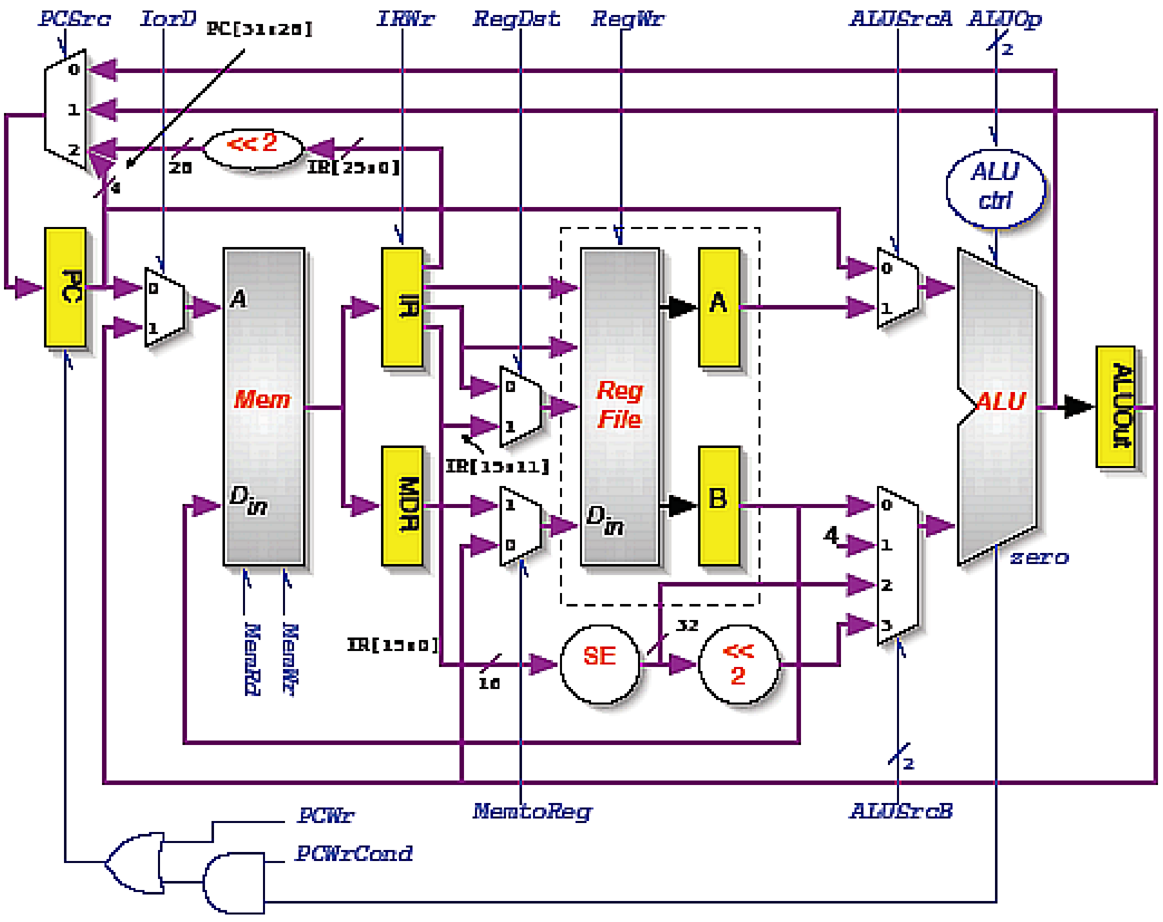
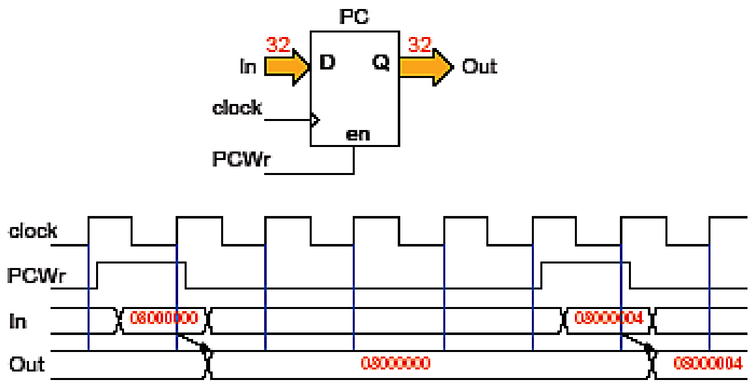
ECE 30
Introduction to Computer Engineering

Study Problems, Set #8
Spring 2014

1.



Shown above is a *multi-cycle* CPU. There are six registers in this datapath: `PC`, `IR`, `MDR`, `A`, `B`, and `ALUOut`. Of these, `PC` and `IR` are *enabled to change* when `PCWr` and `IRWr` are high (logic“1”) respectively. These control signals `PCWr` and `IRWr` turn high shortly after the sampling (rising) edge of the clock, as the control-FSM changes its state. Therefore, the outputs of these registers do not change until the next sampling edge of the clock after the enabling control signals are asserted, as shown in the following timing diagram. [Note: The "in" and "out" in the timing diagram correspond to the input to and the output from the PC.]

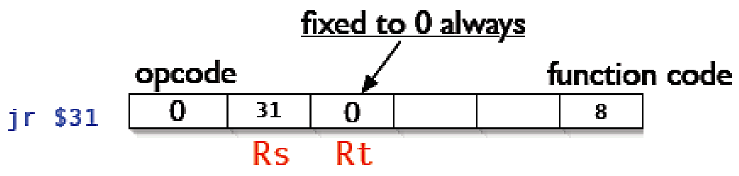


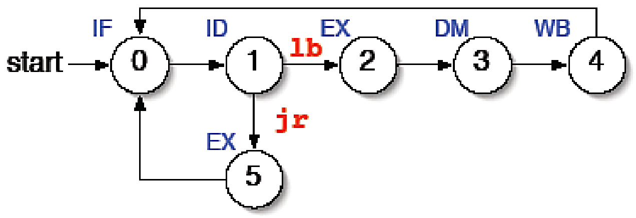
Assume that:

- `ALUOp` = 00 signals the ALU to *add* `A` and `B`.
  `ALUOp` = 01 signals the ALU to *subtract* `B` from `A`.
  `ALUOp` = 10 signals the ALU to do the operation specified by the R-format function code.

- `PC+4` and the *branch target* (`PC+4 + 4*offset`) are computed in the IF and ID/RF stages respectively by the ALU. Note that these are computed regardless of the instruction type.

Complete the following state table for the `lb` and `jr` instructions (state diagram shown below; states 0-5 only). Provide dashes ("-") for unspecified or don't care values in the state table. *Note that* `jr` *is a R-format instruction (see below).*





	0	1	2	3	4	5
IorD						
MemRd						
MemWr						
IRWr						
ALUSrcA						
ALUSrcB						
ALUOp						
PCSrc						
PCWr						
PCWrCond						
RegWr						
RegDst						
MemtoReg						
	IF	ID/RF	EX	DM	WB	EX

Solution:

The signals generated during IF and ID/RF are the same for both instructions, `jr` and `lb`. Columns 0-4 in the table shown below give the control signals generated for the `lb` instruction. Column 5 gives the signals generated for the `jr` instruction during the EX stage of that instruction.

	0	1	2	3	4	5
IorD	0	-	-	1	-	-
MemRd	1	-	-	1	-	-
MemWr	0	0	0	0	0	0
IRWr	1	0	0	0	0	0
ALUSrcA	0	0	1	-	-	1
ALUSrcB	01	11	10	-	-	00
ALUOp	00	00	00	-	-	00
PCSrc	00	-	-	-	-	00
PCWr	1	0	0	0	0	1
PCWrCond	-	0	0	0	0	-
RegWr	0	0	0	0	1	0
RegDst	-	-	-	-	0	-
MemtoReg	-	-	-	-	1	-
	IF	ID/RF	EX	DM	WB	EX

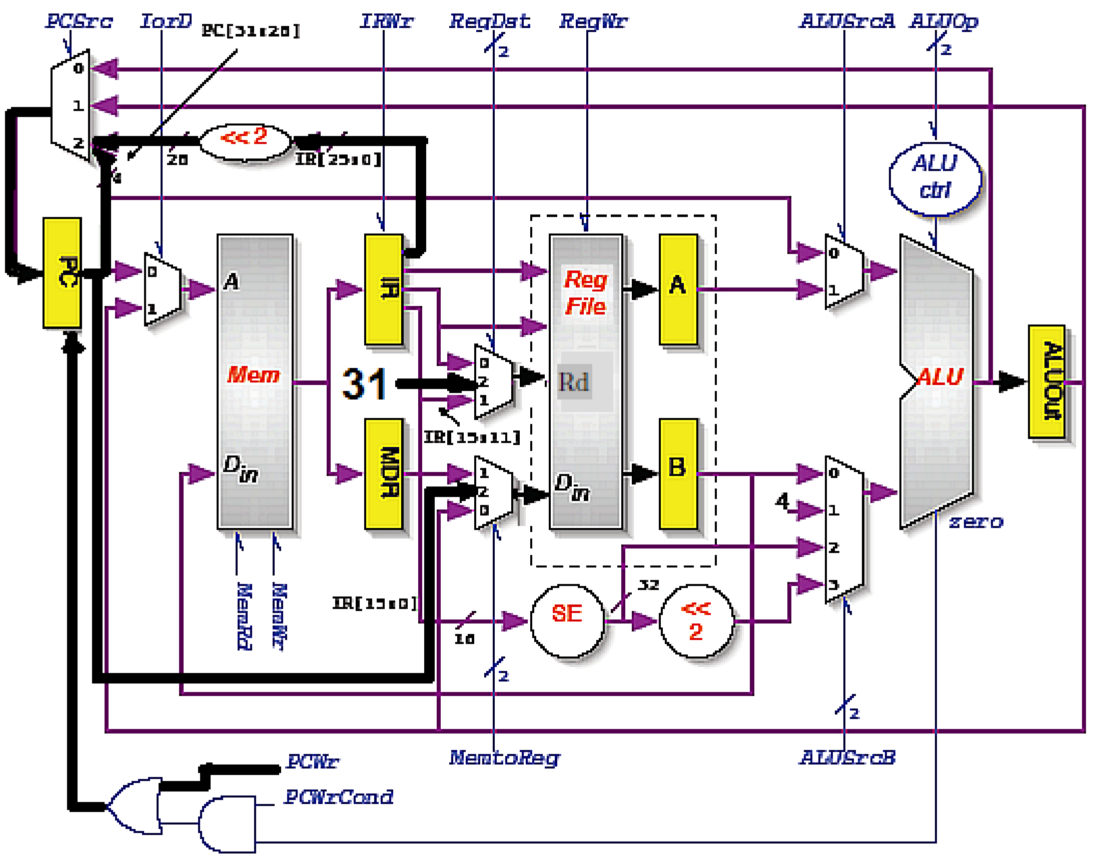
2. The `jal` (jump and link) instruction requires that `PC+4` is stored in `$RA` before PC is set to the jump address. The multi-cycle CPU shown in problem 1 requires a modification to support this instruction.

   (a) In your own words, describe the modifications required in the hardware to implement this instruction. Draw the augmented datapath and highlight all connections required to execute `jal` (assuming the instruction has been loaded into IR).

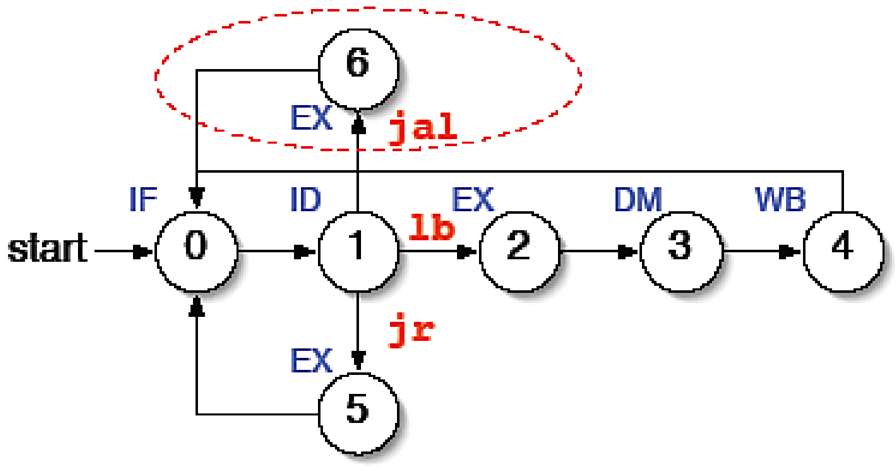
(b) Augment the state diagram and the state table in problem 1 for this instruction.

Solution:

(a) To enable loading `PC+4` into the register `$RA`, i.e., the 31st register in the register file, the 2-input multiplexer that determines the destination register must be upgraded to a 3-input multiplexer with the 3rd input hardwired to the value 31. Also, the 2-input multiplexer that determines the data input to the register file (Din) must be upgraded to a 3-input multiplexer, with the 3rd input connected to the register PC. The diagram below shows the augmented datapath. The connections that are required to execute the `jal` instruction are indicated in black bold lines.



(b) We can now augment the state diagram with a 6th state, as depicted below, to provide the control signals for the execution (EX phase) of the `jal` instruction. The signals generated during IF (phase 0) and ID/RF (phase 1) are the same for all three instructions (`jr`, `lb` and `jal`) and indicated in the state table below. In state 6 (EX phase for `jal`), PC will already contain `PC + 4`, which can be loaded into `$RA` by setting the multiplexers such that PC is applied to the register file's Din input and 31 to the register file's Rd input (to write into the 31st register, i.e., register `$RA`). Simultaneously, PCSrc = 10 ensures that PC gets updated with the jump address specified in the `jal` instruction, stored in register IR.



	0	1	2	3	4	5	6
IorD	0	-	-	1	-	-	-
MemRd	1	-	-	1	-	-	-
MemWr	0	0	0	0	0	0	0
IRWr	1	0	0	0	0	0	0
ALUSrcA	0	0	1	-	-	1	-
ALUSrcB	01	11	10	-	-	00	-
ALUOp	00	00	00	-	-	00	-
PCSrc	00	-	-	-	-	00	10
PCWr	1	0	0	0	0	1	1
PCWrCond	-	0	0	0	0	-	-
RegWr	0	0	0	0	1	0	1
RegDst	-	-	-	-	00	-	10
MemtoReg	-	-	-	-	01	-	10
	IF	ID/RF	EX	DM	WB	EX	EX