CSEE W3827

Fundamentals of Computer Systems

Homework Assignment 5

Prof. Stephen A. Edwards

Columbia University

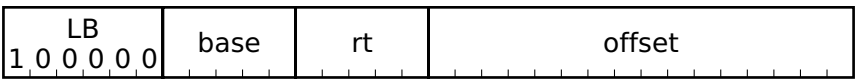
Due November 29th, 2011 at 10:35 AM

Show your work for each problem; we are more interested in how you get your answer than whether you get the right answer.

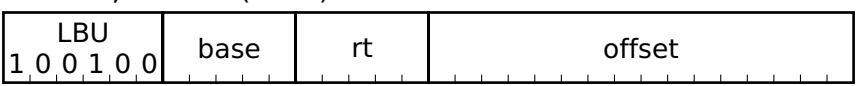
1. (10 pts.) How many seconds would it take to execute a program with 4.5×10^9 instructions on a 3 GHz processor able to run at 0.9 CPI?

2. (30 pts.) Modify the single-cycle MIPS processor shown on the next page to implement the `lb` ("Load byte") and `lbu` ("Load Byte Unsigned") instructions. They are encoded as shown below

`lb rt, offset(base)`



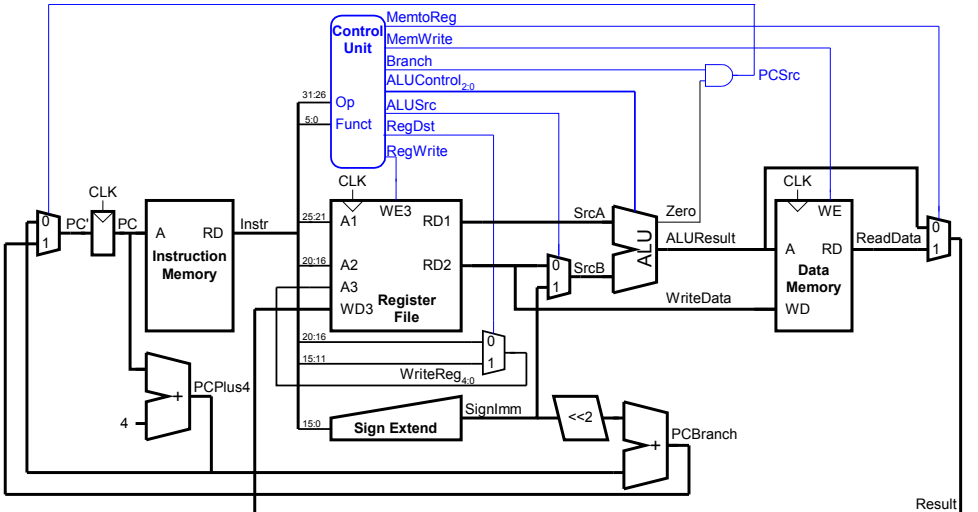
`lbu rt, offset(base)`



These instructions add the contents of the *base* register to a sign-extended *offset* immediate to form an address, fetch a byte from that address, and finally write the byte into the *rt* register.

The two instructions differ in how they handle the top 24 bits. `lb` treats the byte as a signed value and sign-extends the most significant bit of the byte across the top 24 bits; `lbu` treats the byte as unsigned and simply fills the top 24 bits of *rt* with 0's.

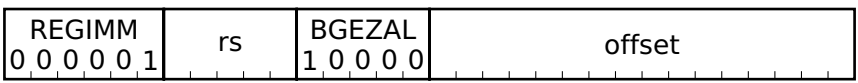
Start from the base single-cycle processor shown on the next page and add components to the datapath, name any new control signals, and show how to add to or modify the main decoder to accommodate these new instructions.



Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00
sw	101011	0	-	1	0	1	-	00
beq	000100	0	-	0	1	0	-	01

3. (35 pts.) Modify the single-cycle MIPS processor shown on the next page to implement the `bgezal` ("Branch on Greater than or Equal to Zero and Link") instruction. This is encoded as shown below

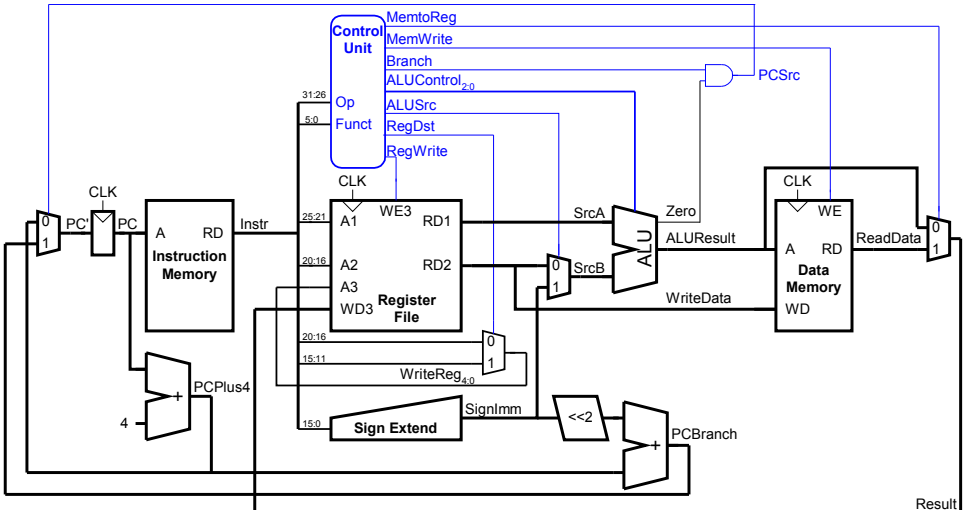
`bgezal rs, offset`



This instruction reads the register *rs* and, if it is greater than or equal to zero, passes control to the instruction *offset* words away from the PC (i.e., as the bne and other branch instructions do), otherwise it passes control to the next instruction in series.

Regardless of the contents of *rs*, this instruction also writes the value of PC + 4 into register 31 (`$ra`), providing the "and link" function.

As in the previous problem, start from the base single-cycle processor shown on the next page and add components and rules.



Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp
R-type	000000	1	1	0	0	0	0	1-
lw	100011	1	0	1	0	0	1	00
sw	101011	0	-	1	0	1	-	00
beq	000100	0	-	0	1	0	-	01

4. (25 pts.) Instead of implementing it directly in hardware, the `bgezal` instruction could be implemented as a pseudoinstruction:

```
                              bltz rs, L1
bgezal rs, offset      →      jal offset
                         L1:
```

Suppose running this software implementation always took two cycles while the hardware implementation took a single cycle, yet increased the clock period by 10%. As a fraction of the number of other instructions, how many *bgezal* instructions would a program have to execute to make the hardware implementation faster?

E.g., if *bgezal* were never executed, the processor using a hardware implementation would be 10% slower; if a program was exclusively *bgezal* instructions (unlikely), a hardware solution would be nearly twice as fast.