CS61c Spring 2014 Discussion 10 – Single Cycle Datapath & Control

1 Register Transfer Language (RTL)

- Use to describe flow of data: $dest \leftarrow src$
- Each line happens in parallel (at the same time): $b \leftarrow c$, $a \leftarrow b$
- In MIPS, use R[x] for register x, and Mem[y] for memory at y.

2 CPU Design

On the next page is the basic datapath as discussed in lecture, shown in simplified format, in which all wires are 32 bits wide, except wires for registers and an immediate. For convenience, the control block is visually split into input and output blocks.

3 Exercises

For the following exercises, assume that the ALU is able to output an "equals" signal, which is high/on/one when its two inputs are equal.

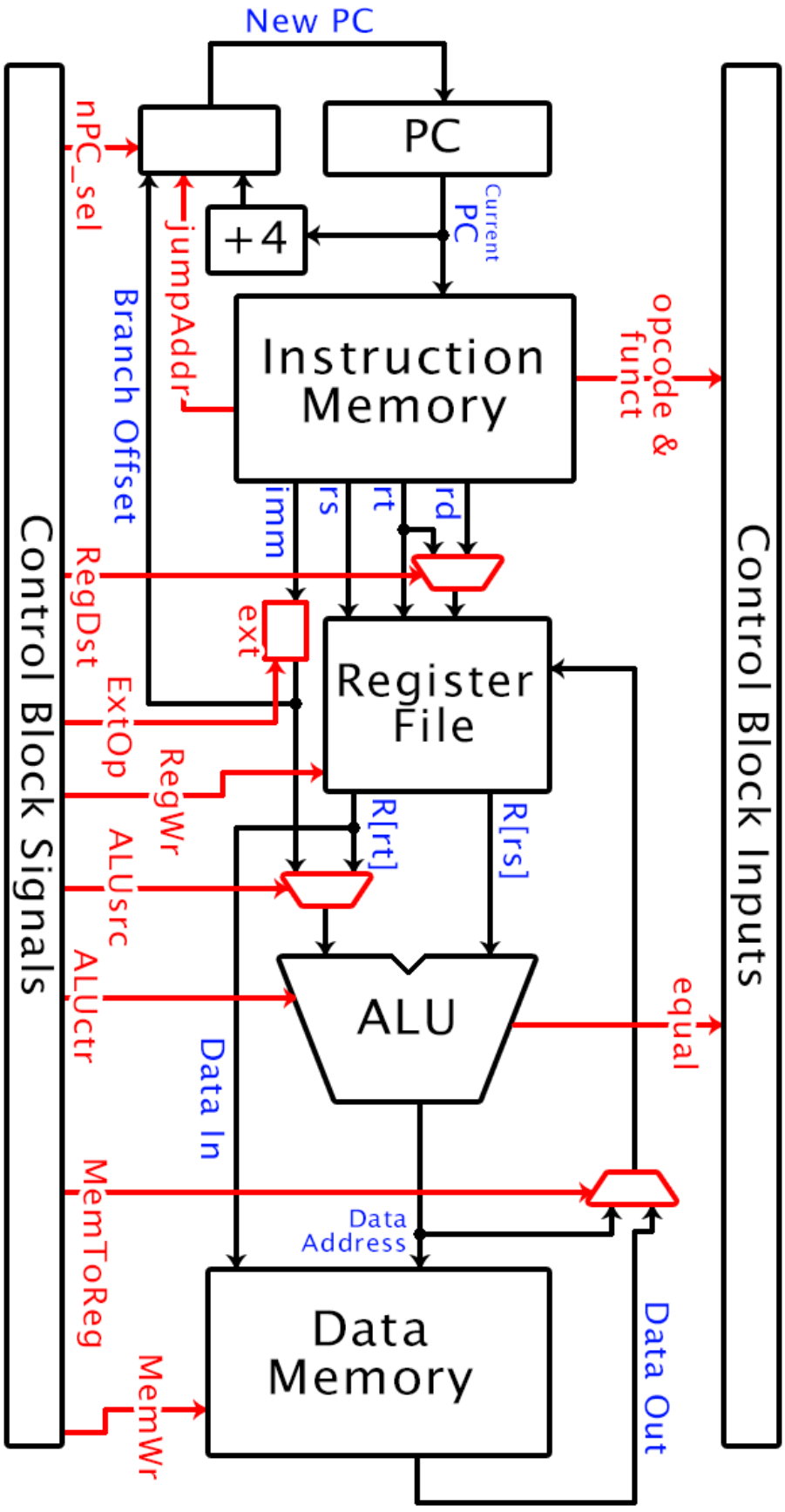
1. Label the wires in the datapath diagram, describing what data is on each line. For example, one of the outputs of the registers block might be R[rs]. (solution shown on the diagram in blue)
2. Add control signals and missing elements (such as multiplexers) to the diagram so that the datapath can execute the following instructions: `add`, `lui`, `sw`, `bne`, `j`. (solution shown on the diagram in red)
3. Fill out the values for the control signals from question 2 (Write your control signals' names along the top row):

Inst	nPCsel	RegDst	RegWr	ALUsrc	ALUctr	MemToReg	MemWr	ExtOp
`add`	PC+4	rd	YES	rt	add	NO	NO	✗
`lui`	PC+4	rt	YES	imm	shiftx16	NO	NO	zero*
`sw`	PC+4	✗	NO	imm	add	✗	YES	sign
`bne`	branch	✗	NO	rt	✗	✗	NO	sign**
`j`	jump	✗	NO	✗	✗	✗	NO	✗

* Doesn't really matter since ALUctr is shiftx16, ** Branch also multiplies immediate by 4 at some point

4. Suppose you wanted to add a new instruction, `beqr`, which will be used like this: `beqr $x, $y, $z` will branch to the address in `$z` if `$x` and `$y` are equal, otherwise continue to the next instruction. Show any changes that would need to be made to the datapath to make this instruction work.

   The diagram is already crowded, so I will explain the changes. Since we're reading three registers at once here, we need to add a third read port to the register file, and correspondingly, a third read address port. We can reuse `beq`'s datapath to compare two registers, `rs` and `rt`, so rd would contain the address to jump to. We would then connect `R[rd]` to the address logic module (which is essentially a mux).


New PC
PC
+4
Current PC
nPC_sel
jumpAddr
Branch Offset
Instruction Memory
opcode & funct
imm
rs
rt
rd
RegDst
ext
ExtOp
Register File
RegWr
R[rt]
R[rs]
ALUsrc
ALUctr
ALU
equal
Data In
MemToReg
Data Address
Data Memory
MemWr
Data Out
Control Block Signals
Control Block Inputs