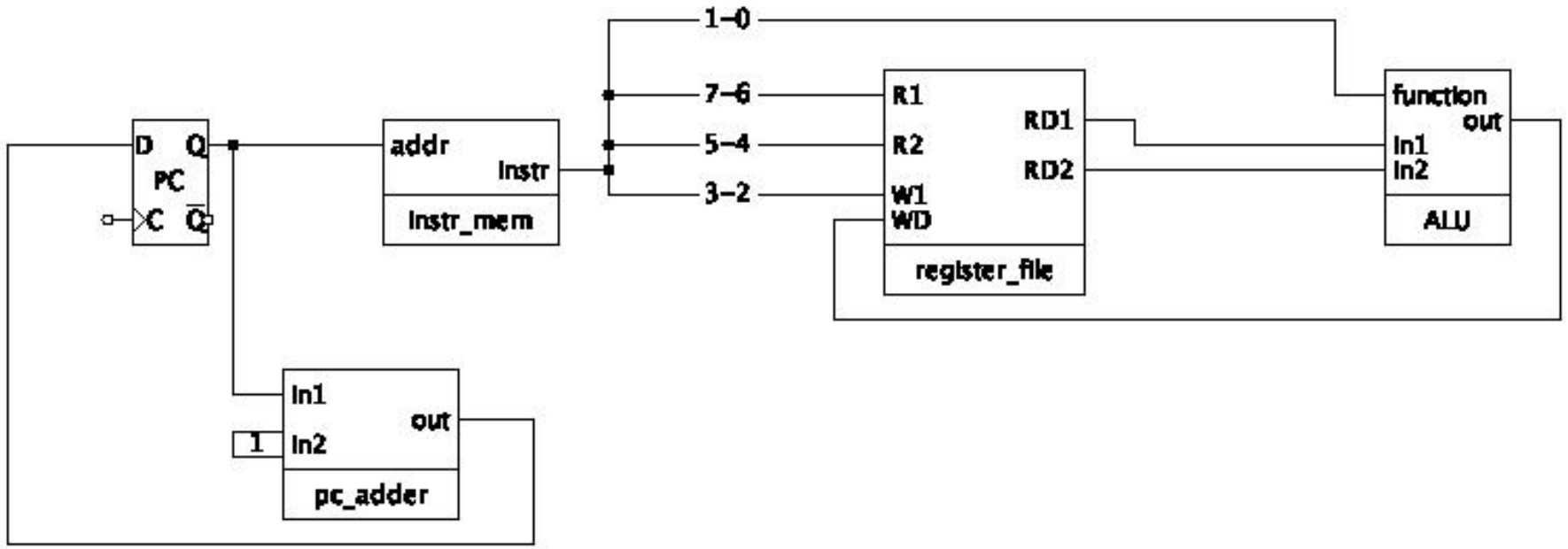
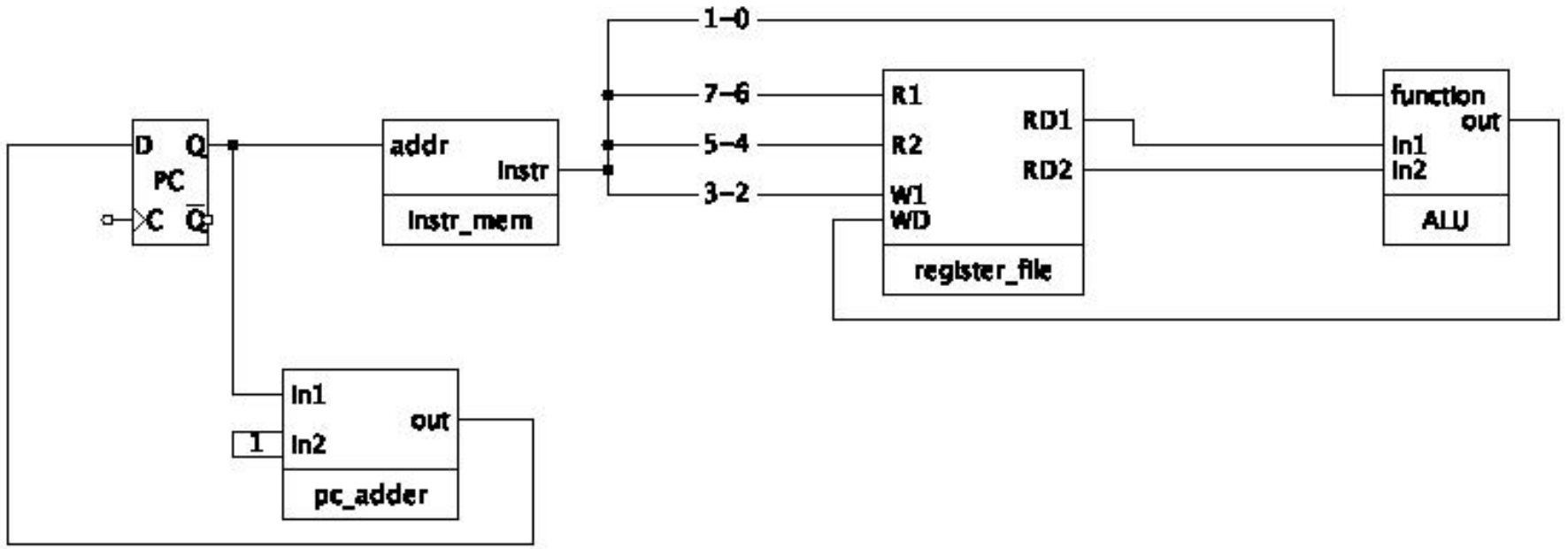
Simple MIPS Computer

Nathan Bowman

Simple “Fake” Computer

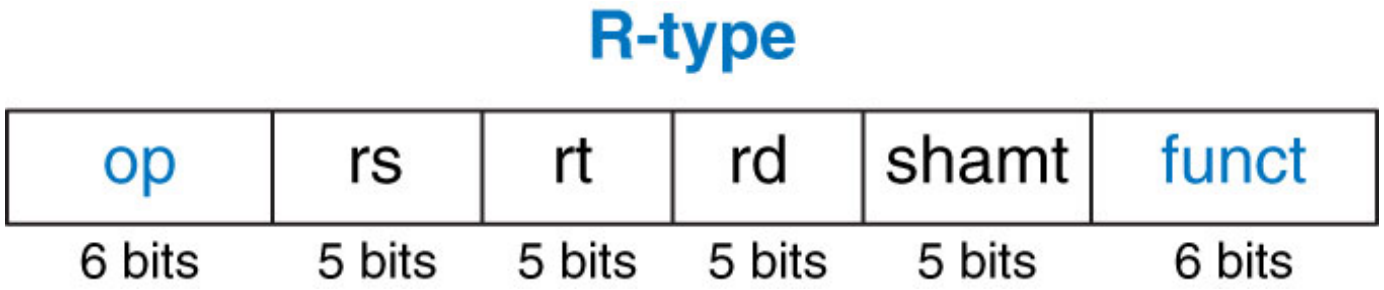


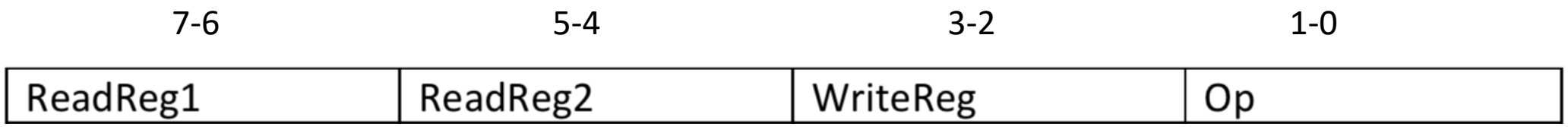
Simple “Fake” Computer



7-6	5-4	3-2	1-0
ReadReg1	ReadReg2	WriteReg	Op

MIPS R-type


R-type
op
rs
rt
rd
shamt
funct
6 bits
5 bits
5 bits
5 bits
5 bits
6 bits


7-6
5-4
3-2
1-0
ReadReg1
ReadReg2
WriteReg
Op

MIPS

MIPS is 32-bit architecture. This affects a lot of things, including:

Registers each store 32 bits

- If PC is 32 bits, memory is 2^{32} bytes

Instructions 32 bits long

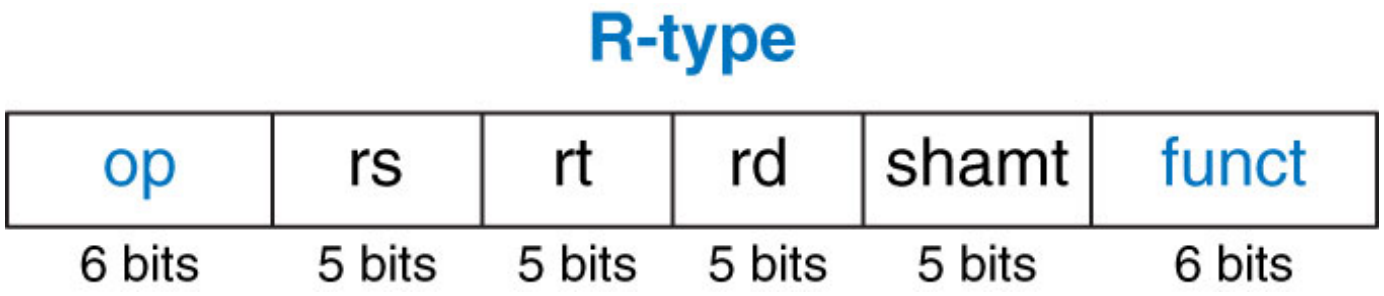
- 4 bytes – contrast with 1 byte for our previous architecture

MIPS

Coincidentally, MIPS has 32 registers

This is not because MIPS is 32-bit architecture

You can tell by looking at number of bits to specify register



MIPS instructions and memory

Memory grabs one byte, but MIPS instructions are 4 bytes

How do we grab entire instruction?

MIPS instructions and memory

~~Memory grabs one byte~~, but MIPS instructions are 4 bytes

Like many things in 32-bit architecture, memory will be moved around in chunks of 32 bits

32 bits referred to as one **word** of memory

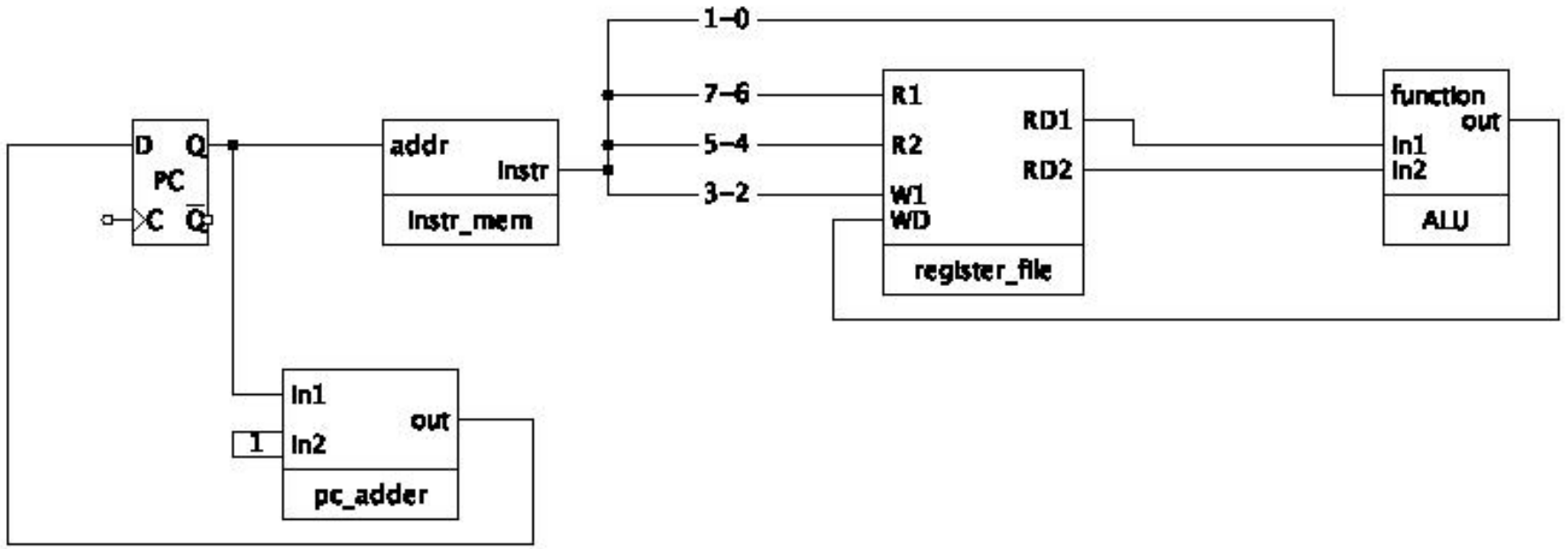
- Word is not standard unit like ”byte” – changes based on architecture

Memory still *byte-addressable*, but grabs 4 bytes at once

- We specify which byte to start with
- Slightly more complicated in practice, but we’ll discuss word boundaries later

MIPS instructions and memory

Some changes needed

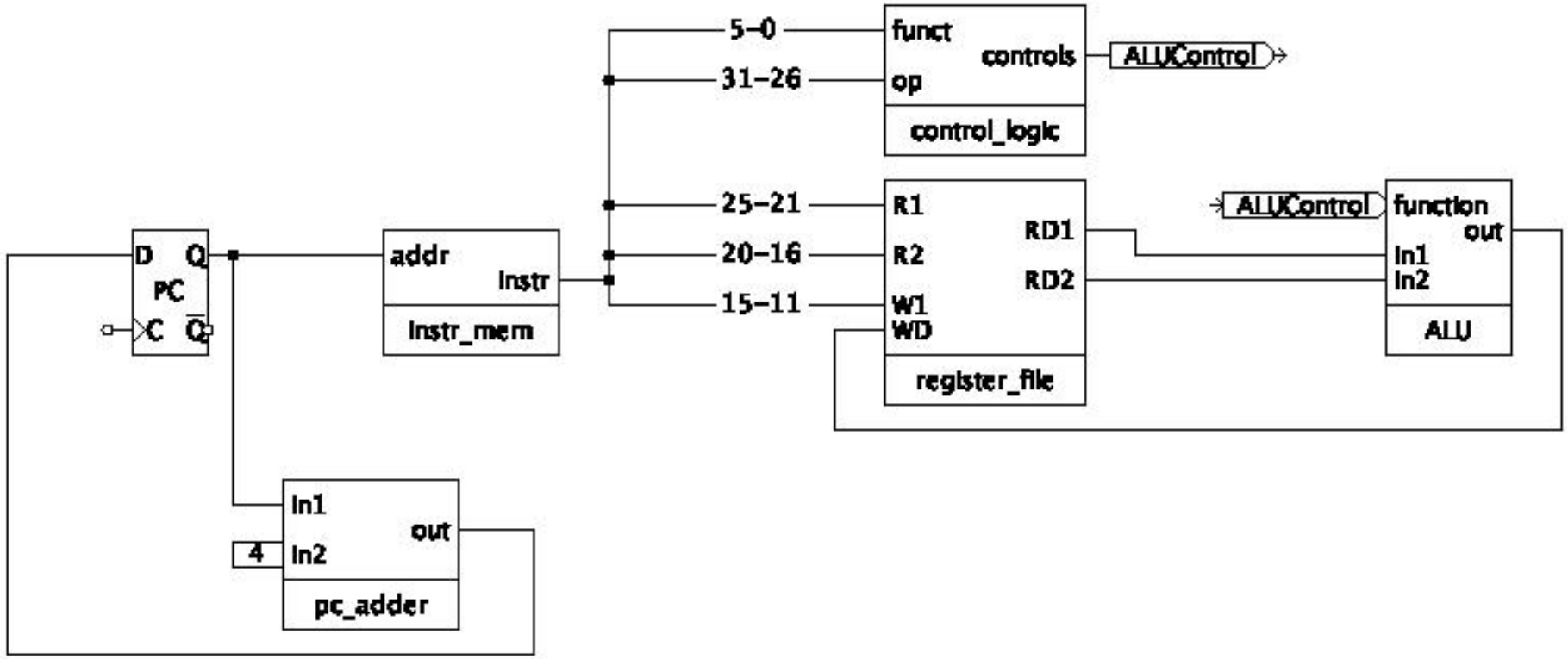


Changes

Fewer changes required than you might think

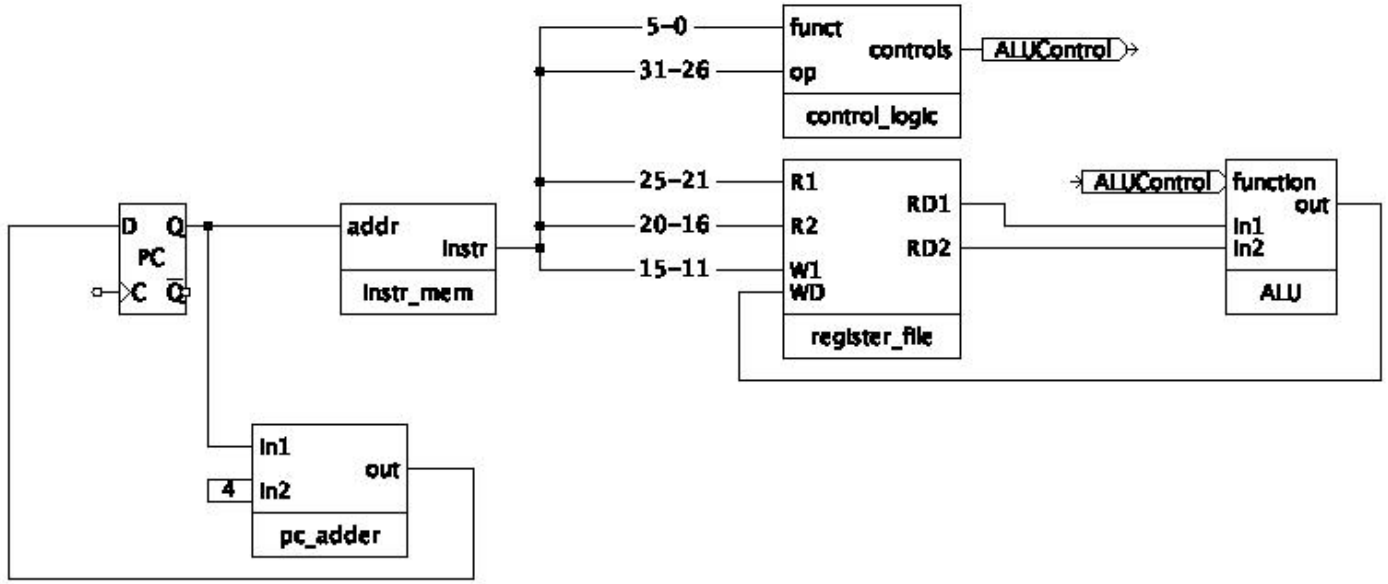
Memory now grabs 4 bytes at once, so adder must increment PC by 4

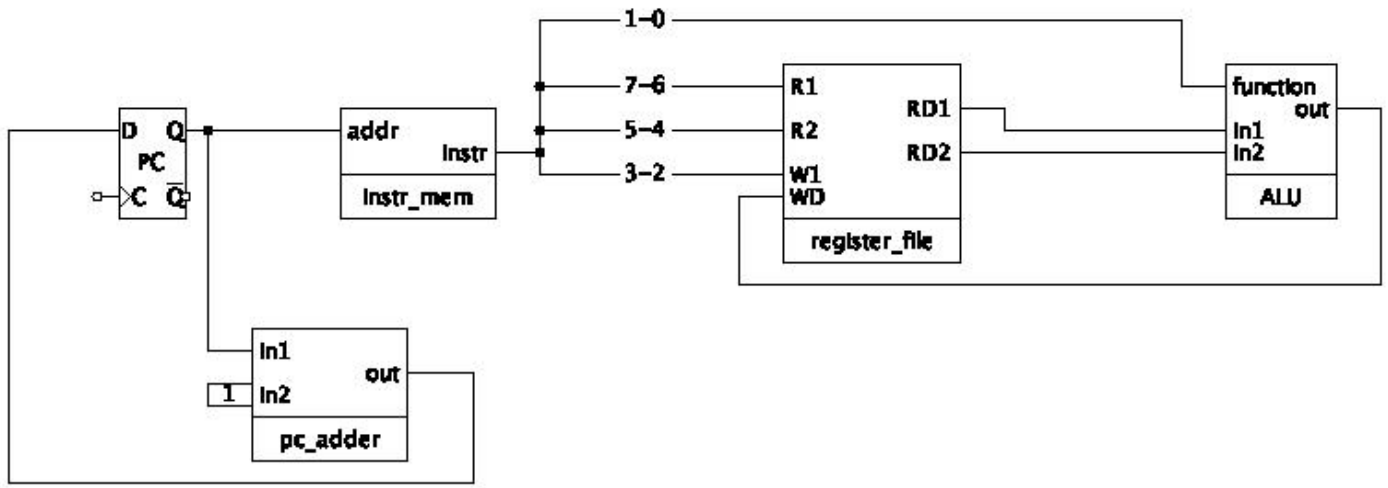
Unbundler needs to be adjusted to account for slightly different format



R-type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits


5–0
31–26
funct
op
controls
control_logic
ALUControl
25–21
20–16
15–11
R1
R2
W1
WD
RD1
RD2
register_file
ALUControl
function
In1
In2
out
ALU
D
Q
PC
C
Q
addr
Instr
Instr_mem
In1
4
In2
out
pc_adder


1–0
7–6
5–4
3–2
R1
R2
W1
WD
RD1
RD2
register_file
function
In1
In2
out
ALU
D
Q
PC
C
Q
addr
Instr
Instr_mem
In1
1
In2
out
pc_adder

R-type

That’s it for MIPS R-type datapath

MIPS has instructions other than R-type

Generally going to follow process of

1. Add instructions/types to assembly language as needed
2. Modify datapath to allow new instructions without messing up old ones