

Project 2 Discussion

CS 3339

Lee Hinkle

Texas State University, San Marcos

With deep acknowledgement to Dr. Martin Burtscher
and Ms. Molly O'Neil



When things just don't make sense...

- Often the compiler will tell you which line is incorrect, however sometimes it will actually be the line before the error that causes the problem.
- File systems on Linux and within Integrated Development Environments (IDEs) can be confusing – make sure you are actually referring to or editing the correct file.
- Break up the work into chunks of time so you can step away and come back after thinking it over.
- Practice good code and file “hygiene” – stay organized
Make frequent backups or use `git.txstate.edu`



P2 all new but many similarities to P1

If you did not get P1 working 100% it still makes sense to finish it. Carry forward code will be tested and you can use the disassembler you wrote to look at the new binaries.





Project 2

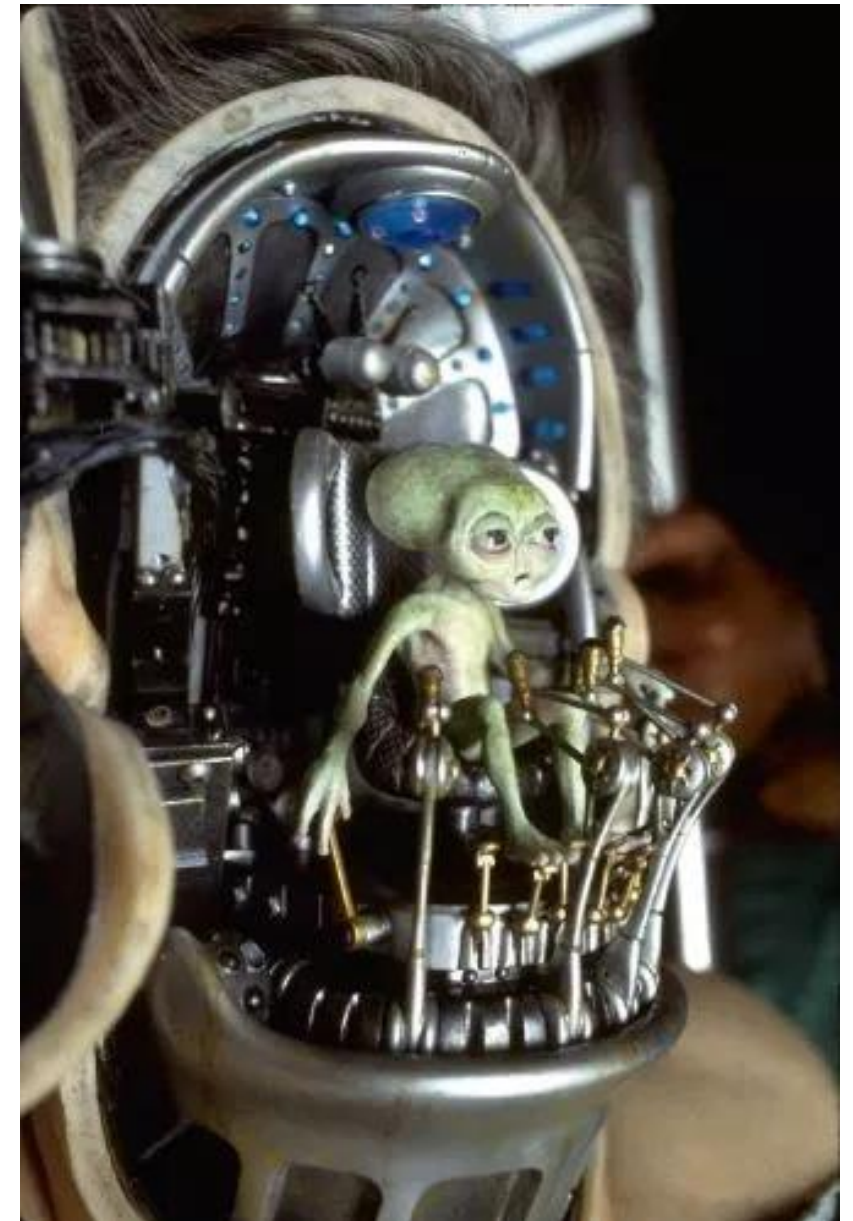
Object Oriented C++ code – there are several classes (ALU, Memory, CPU) that will be instantiated into objects.

Look at the .h code to better understand interface

Like project 1 you will decode the instruction fields

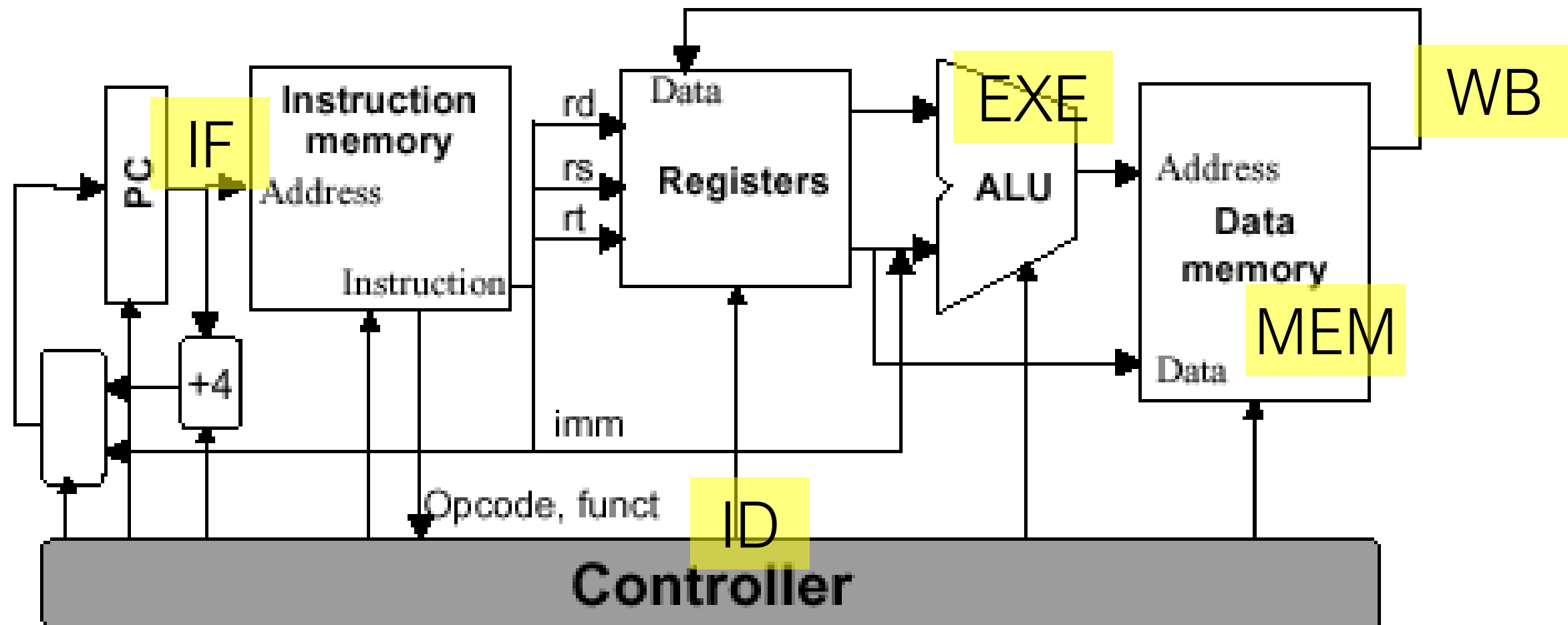
You “live” in CPU.cpp decode,
you are the controller, BE THE
CONTROLLER!!!

which really means you have
to setup the control signals
based on the instruction so
the rest of the processor
(emulator) can do it's thing.



Processor – Simplified View

```
fetch();  
decode();  
execute();  
mem();  
writeback();
```

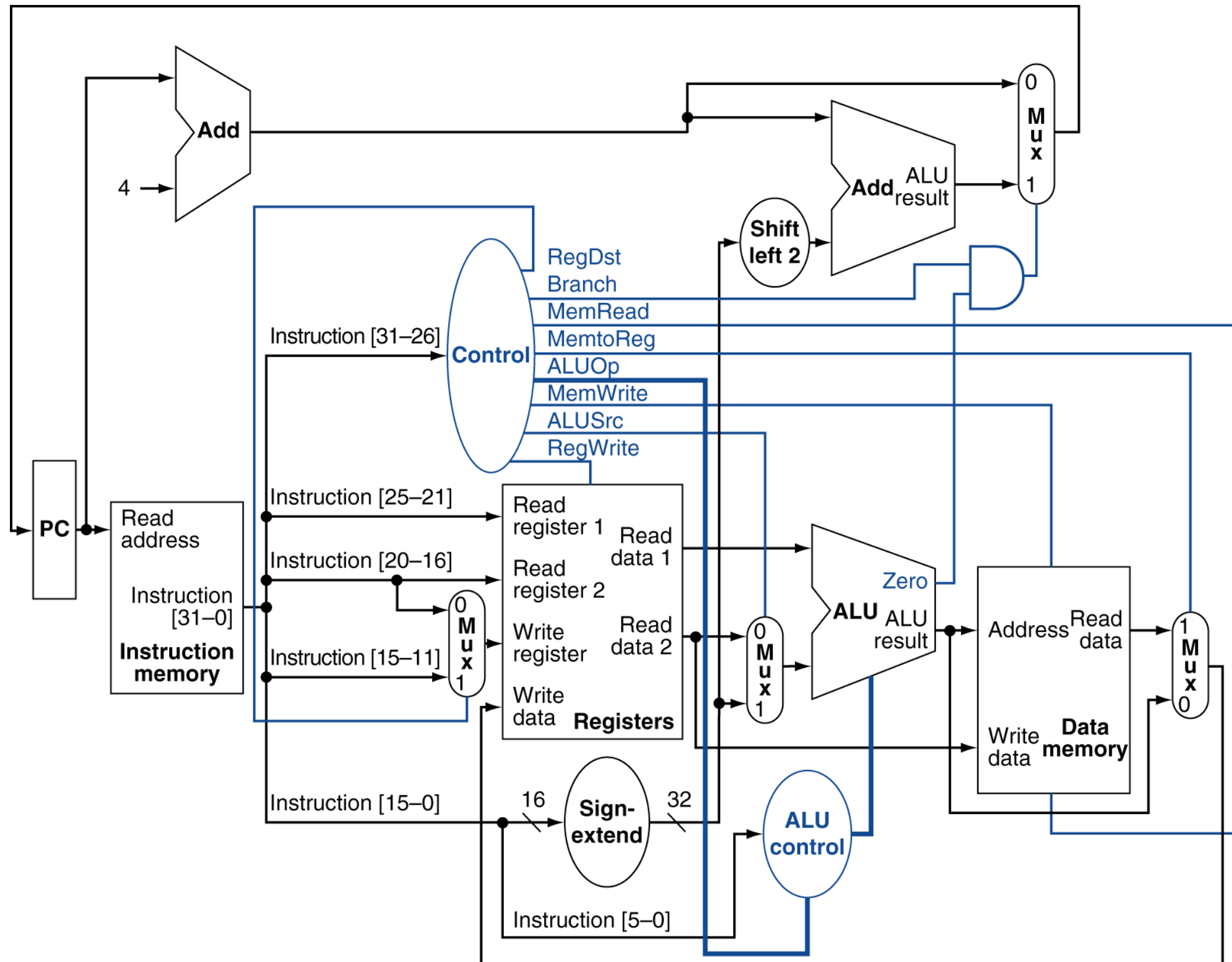


From <https://www.cise.ufl.edu/~mssz/CompOrg/CDA-proc.html>
accessed 2/12/2017



Datapath With Control

Note: Project control signal names differ – look at example instr and CPU.h



Some VI and VIM hints

In **command** mode - to get in command mode press <esc> and type :

yy – yanks a line to general buffer

5Y – yanks 5 lines, use whatever number you need

p – puts the line(s) after current line

r<char> - replace current character with typed one

dd – delete line

x – delete character

nnn enter – go to line nnn

q – quit if you made not changes

wq or x – quit and save changes

q! – I made changes but want to quit without saving them you stupid editor!!!

The other mode in vi and vim is the **insert** mode where characters typed are inserted at cursor. Usually you get into insert mode by typing 'i' while in command mode.



diff

```
diff ../project2_debug_out/rand.debug_out ./rand.out | more
```

```
88c88
```

```
< 00400130: trap 3600001 b1
```

```
---
```

```
> 00400130: trap 1600001 b1
```

```
333c333
```

```
< $t8 : 00000000 $t9 : 00000000 $k0 : 41c64e6d $k1 : 7a1c395d
```

```
---
```

```
> $t8 : 00000000 $t9 : 00000000 $k0 : 41c64e6d $k1 : 000000b1
```

```
343c343
```

```
< $t8 : 00000000 $t9 : 00000000 $k0 : 41c64e6d $k1 : 7a1c6996
```

```
---
```

```
> $t8 : 00000000 $t9 : 00000000 $k0 : 41c64e6d $k1 : 000030ea
```

```
347c347
```

```
< MEM WR: addr = 0x100ffff0, data = 0x7a1c6996
```

```
---
```

```
> MEM WR: addr = 0x100ffff0, data = 0x30ea
```

```
354c354
```

```
< $t8 : 00000000 $t9 : 00000000 $k0 : 41c64e6d $k1 : 7a1c6996
```

```
---
```

```
> $t8 : 00000000 $t9 : 00000000 $k0 : 41c64e6d $k1 : 000030ea
```

```
358c358
```

```
< MEM RD: addr = 0x100ffff0, data = 0x7a1c6996
```

```
---
```

```
➤ MEM RD: addr = 0x100ffff0, data = 0x30ea
```

a -Added the text to file

c -Changes are made in the file

d -Deletion operation is performed

< Lines from the first file

> Lines from the second file


The Debug mode produces a ton of output! Cntrl-C to terminate quickly.



diff -y (side by side)

```
$ diff -y ../project2_debug_out/rand.debug_out ./rand.out | more
```

```
0040012c: lw $k1, -4($fp)                                0040012c: lw $k1, -4($fp)
MEM RD: addr = 0x100ffff4, data = 0xb1                MEM RD: addr = 0x100ffff4, data = 0xb1
$zero: 00000000    $at  : 00000000    $v0  : 00000000    $zero: 00000000    $at  : 00000000    $v0  : 00000000
$a0  : 00000000    $a1  : 00000000    $a2  : 00000000    $a0  : 00000000    $a1  : 00000000    $a2  : 00000000
$t0  : 00000000    $t1  : 00000000    $t2  : 00000000    $t0  : 00000000    $t1  : 00000000    $t2  : 00000000
$t4  : 00000000    $t5  : 00000000    $t6  : 00000000    $t4  : 00000000    $t5  : 00000000    $t6  : 00000000
$s0  : 00000000    $s1  : 00000000    $s2  : 00000000    $s0  : 00000000    $s1  : 00000000    $s2  : 00000000
$s4  : 00000000    $s5  : 00000000    $s6  : 00000000    $s4  : 00000000    $s5  : 00000000    $s6  : 00000000
$t8  : 00000000    $t9  : 00000000    $k0  : 00000000    $t8  : 00000000    $t9  : 00000000    $k0  : 00000000
$gp  : 10008000    $sp  : 100ffff4    $fp  : 100ffff8    $gp  : 10008000    $sp  : 100ffff4    $fp  : 100ffff8
hi   : 00000000    lo   : 00000000    hi   : 00000000    lo   : 00000000
00400130: trap 3600001 b1                                | 00400130: trap 1600001 b1
$zero: 00000000    $at  : 00000000    $v0  : 00000000    $zero: 00000000    $at  : 00000000    $v0  : 00000000
$a0  : 00000000    $a1  : 00000000    $a2  : 00000000    $a0  : 00000000    $a1  : 00000000    $a2  : 00000000
$t0  : 00000000    $t1  : 00000000    $t2  : 00000000    $t0  : 00000000    $t1  : 00000000    $t2  : 00000000
$t4  : 00000000    $t5  : 00000000    $t6  : 00000000    $t4  : 00000000    $t5  : 00000000    $t6  : 00000000
$s0  : 00000000    $s1  : 00000000    $s2  : 00000000    $s0  : 00000000    $s1  : 00000000    $s2  : 00000000
$s4  : 00000000    $s5  : 00000000    $s6  : 00000000    $s4  : 00000000    $s5  : 00000000    $s6  : 00000000
$t8  : 00000000    $t9  : 00000000    $k0  : 00000000    $t8  : 00000000    $t9  : 00000000    $k0  : 00000000
$gp  : 10008000    $sp  : 100ffff4    $fp  : 100ffff8    $gp  : 10008000    $sp  : 100ffff4    $fp  : 100ffff8
hi   : 00000000    lo   : 00000000    hi   : 00000000    lo   : 00000000
00400134: trap 0                                00400134: trap 0
```



Important: When you use the side by side option the right hand side of both files will likely be cut off.



diff -y piped into grep

```
diff -y ../project2_debug_out/rand.debug_out ./simulator.out | grep '|' | more
```

```
00400130: trap 3600001 b1          -          -          | 00400130: trap 1600001 b1
  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d      |  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d
  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d      |  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d
MEM WR: addr = 0x100ffff0, data = 0x7a1c6996          |  MEM WR: addr = 0x100ffff0, data = 0x30ea
  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d      |  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d
MEM RD: addr = 0x100ffff0, data = 0x7a1c6996          |  MEM RD: addr = 0x100ffff0, data = 0x30ea
  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d      |  $t8 : 00000000    $t9 : 00000000    $k0 : 41c64e6d
  $t8 : 00000000    $t9 : 00000000    $k0 : 00000000      |  $t8 : 00000000    $t9 : 00000000    $k0 : 00000000
MEM RD: addr = 0x100ffff0, data = 0x7a1c6996          |  MEM RD: addr = 0x100ffff0, data = 0x30ea
  $t8 : 00000000    $t9 : 00000000    $k0 : 00000000      |  $t8 : 00000000    $t9 : 00000000    $k0 : 00000000
  $t8 : 00000000    $t9 : 00000000    $k0 : 00000063      |  $t8 : 00000000    $t9 : 00000000    $k0 : 00000063
  $t8 : 00000000    $t9 : 00000000    $k0 : 00000063      |  $t8 : 00000000    $t9 : 00000000    $k0 : 00000063
hi   : 00000033    lo   : 013bc301          |  hi   : 00000030    lo   : 0000007e
  $t8 : 00000000    $t9 : 00000000    $k0 : 00000063      |  $t8 : 00000000    $t9 : 00000000    $k0 : 00000063
hi   : 00000033    lo   : 013bc301          |  hi   : 00000030    lo   : 0000007e
  ^ - - - - -      ^ - - - - -      ^ - - - - -      |  ^ - - - - -      ^ - - - - -      ^ - - - - -
```

name comes from g/re/p – global search/regular expression/print output

Important: When you use the side by side option the right hand side of both files will likely be cut off.



Shell Script Basics

```
$ vim hello
insert mode, type
#!/bin/bash
echo Hello There
<esc> wq
$ hello
-bash: hello: command not found
$ ./hello
-bash: ./hello: Permission denied
$ chmod +755 hello
$ ./hello
Hello There
$
```



Shell Script Basics

```
#!/bin/bash
#Script to compile, run, and compare output
make
./simulator rand.mips > simulator.out
diff -y ./rand.out ./simulator.out
```

```
[lbh31@zeus project2]$ ./runall
make: `simulator' is up to date.
CS 3339 MIPS Simulator
Running: rand.mips
```

```
177
1181983268
```

```
Program finished at pc = 0x400164 (4817 instructions execute
```

```
CS 3339 MIPS Simulator
Running: rand.mips
```

```
177
1181983268
```

```
Program finished at pc = 0x400164 (4817 instructions execute
```



Some more project specific hints

- PC is already incremented in CPU::fetch
- Jump does a lot less than JAL.
- You probably will not need to change 'trap' code at all.
- You have direct access to hi and lo registers, but must use ALU to move from regFile to move to correct spot – can't assign to writeData directly (done in CPU::mem)!
Use add + zero register trick
- The debug outputs and completed instructions provide very good hints – review them carefully.



SLL and SRA convention

Follow the provided debug examples – the source register differs from greensheet

```
case 0x00: D(cout << "sll " << regNames[rd] << ", " << regNames[rs] << ", " << dec << shamt);  
    // Rd = Rs << shamt;
```

<snip>

```
case 0x03: D(cout << "sra " << regNames[rd] << ", " << regNames[rs] << ", " << dec << shamt);  
    // Rd = Rs >> shamt (with sign extension)
```



“Unlike the MIPS single-cycle CPU we will cover in class, you should resolve branch and decode instructions inside `decode()`. Do not use the ALU to compare the sources for conditional branch instructions. Rather, do the comparison and update the pc field accordingly inside `decode()`.”

For address calculations use:

```
pc = (pc & 0xf0000000) | addr << 2;
```

adjust letters in blue as appropriate for the instruction



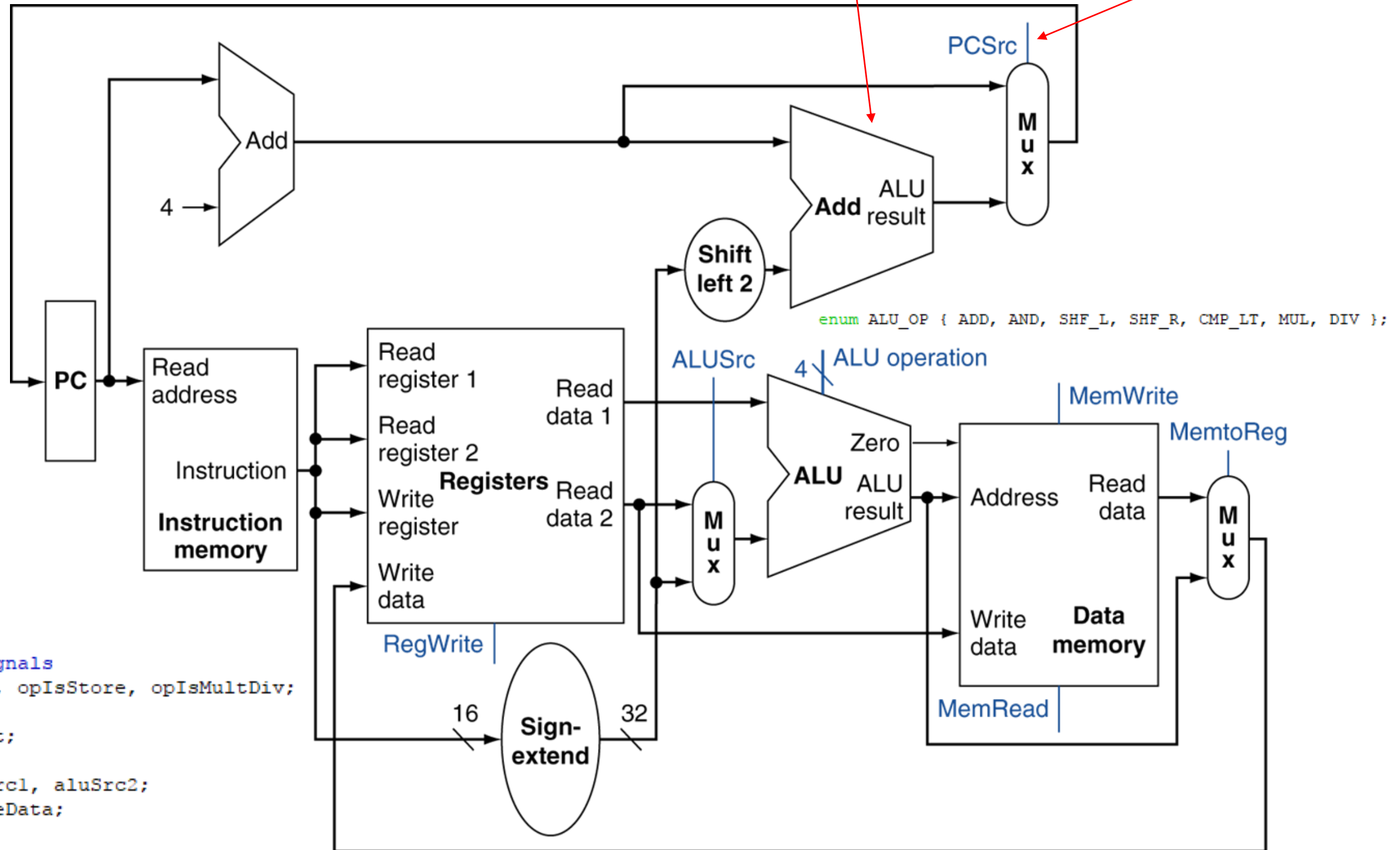
```

void CPU::fetch() {
    instr = iMem.loadWord(pc);
    pc = pc + 4;
}

```

You do not have
this ALU use '+'

Do not use ALU to
compare values in rs
and rt use '==' for beq,
bne instructions



```

// Control signals
bool opIsLoad, opIsStore, opIsMultDiv;
ALU_OP aluOp;
bool writeDest;
int destReg;
uint32_t aluSrc1, aluSrc2;
uint32_t storeData;

```



“Do not change any code outside of the CPU::decode() function. However, your decode() code **may not update the register file** (e.g., no statements like regFile[x] = y; may go in decode) and **may not interact with memory** (e.g., no calls to loadWord or storeWord in decode). Use decode() to correctly set up the control signals so that my execute(), mem(), and writeback() functions behave correctly.”

Submit only your final CPU .cpp file and the optional README .txt file to TRACS. **Put your netID at the front of each filename e.g. lbh31_CPU.cpp and lbh31_README.txt.** Upload each file separately and do not compress them. Do not submit any additional files (i.e., no other .cpp files, no .h files, no input or output files).

