CS 2200 A4 Recitation 2 (Sanjana & Jacob)

Turn in HW 1 at the front with your Buzzcard :)

Announcements

- Homework 1 is due RIGHT NOW if you haven’t given it to us already
- Homework 2 is released
    - Due Next Wednesday
    - Mostly coding in RAMA-2200 Assembly, with one short-answer question
- Reminder - Project 1 is due February 8
    - Get started soon if you haven’t already - office hours will be crowded on the due date, we want you to have time to get proper help if needed!

Revisiting Calling Conventions

The Problem With Procedure Calls

```
...
addi $s0, $zero, 1        !$s0 contains important value for later
addi $a0, $zero, 2
jalr $at, $ra             !jumps to double routine which doubles the argument
add  $s1, $v0, $s0        !returns the returned value plus $s0
...
```

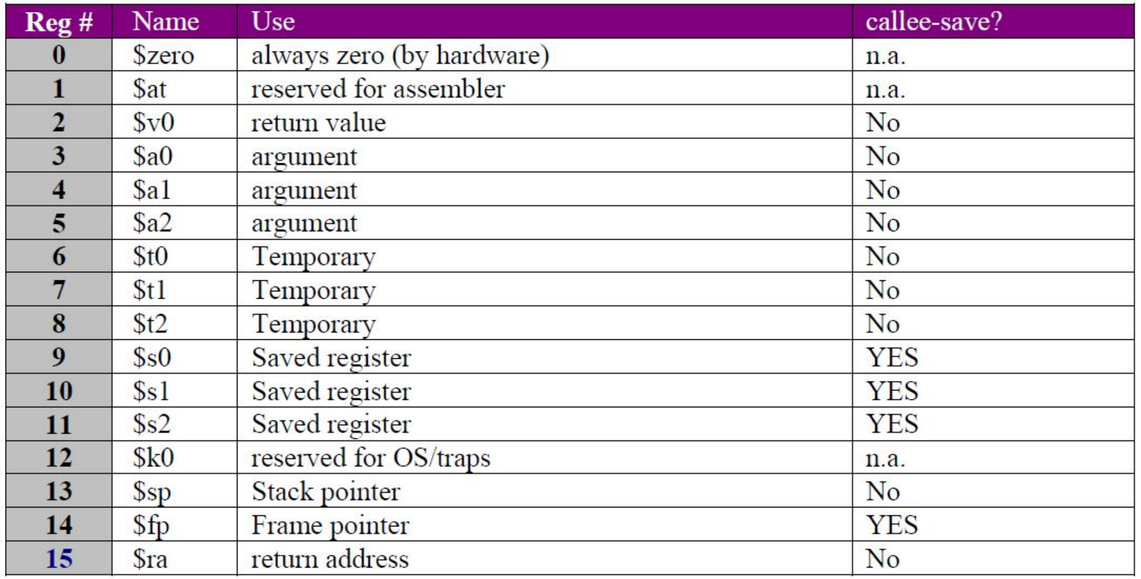
See the Issue?

- When we make procedure calls, we want to be able to resume where we left off before the procedure call
    - Need to track the state of the registers
- One problem - we don’t know what registers a procedure call will use!
- Enter the calling convention

Steps in Compiling Procedure Calls

- Preserve the state of the caller
- Pass the parameters to the callee
- Remember the return address
- Transfer control to callee
- Allocate space for callee’s local variables
- Receive return value(s) from callee and pass them back to the caller
- Return to the point of the procedure call

Peace Treaty



Reg #	Name	Use	callee-save?
0	$zero	always zero (by hardware)	n.a.
1	$at	reserved for assembler	n.a.
2	$v0	return value	No
3	$a0	argument	No
4	$a1	argument	No
5	$a2	argument	No
6	$t0	Temporary	No
7	$t1	Temporary	No
8	$t2	Temporary	No
9	$s0	Saved register	YES
10	$s1	Saved register	YES
11	$s2	Saved register	YES
12	$k0	reserved for OS/traps	n.a.
13	$sp	Stack pointer	No
14	$fp	Frame pointer	YES
15	$ra	return address	No

Table 2.2: Register convention

Finalized Convention

- Caller saves temp registers
- Caller saves ADDITIONAL parameters
- Caller allocates space for additional return values
- Caller saves previous return address (from current $ra)
- Caller executes JALR $ra, $at
- Callee saves previous $fp, copies contents of $sp into $fp
- Callee saves any s registers it plans to use
- Callee's local variables allocated on the stack
- Prior to return, callee restores any saved s0 to s3 registers
- Callee restores old $fp
- Return to caller JALR $zero $ra
- Upon return, caller restores previous $ra value
- Caller stores return values as desired
- Caller moves $sp to discard additional parameters
- Caller restores any saved temp (t0 - t3) registers from stack

Processor Implementation

Processor Implementation

Architecture != Implementation

- Can be many processor implementations for the same instruction set architecture
  - Cost/Performance trade-off => different implementations for different performance needs
- This is good: Parallel development of hardware and software
  - Software changes slower than hardware upgrades
  - Can maintain the same instruction set from one processor generation to the next

How is it implemented?

- What we know: Instructions in our architecture
  - Memory Recall...
  - Add, Addi, And, Nand, Beq, Halt, etc.
- **Question:** What do we need in order for these instructions to work?
  - Example: registers (how many did we say we use?)
  - What else?

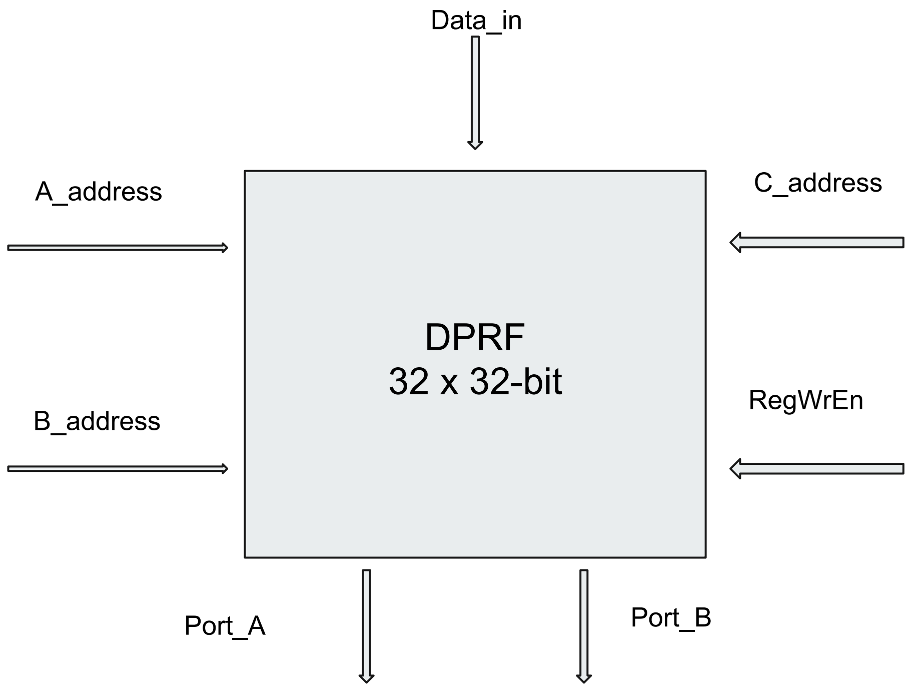
Elements

- PC: **ADDRESS** (I repeat, **ADDRESS**) of currently executing instruction (or next instruction, if incremented)
- IR: Current instruction
    - Think: why would we need to hold this value?
        - **Hint**: *How is an instruction formatted? Does it contain useful information?*
- Memory: holds all data AND instructions part of the program
- Register File
    - Single Ported vs Dual Ported: *how are they different?*
- ALU

Combinational vs Sequential Logic

- Combinational
  - Output is a boolean combination of the inputs
  - No concept of state
  - Examples?
- Sequential
  - Output is a boolean combination of the inputs *and* the current state
  - Examples?

How many wires for each of the in/outputs?



Edge-Triggered Logic

- How do we change the contents of a register?
    - Clock signals!
- Two options for *when* the actual contents change
    - Level-logic: the change happens as long as the clock signal is high
    - Edge-triggered: the change happens either the clock changes from low to high (rising-edge) or when the clock changes from high to low (falling-edge)
- You can assume all registers used for our LC-2200 datapath are rising-edge triggered
- You can also assume that in this class, memory is level logic
    - Contents available after read access time

Breaking Down An Instruction

- What will an ADD instruction look like in terms of the datapath elements we've discussed so far? How should we connect these elements?
- How many cycles will it take to complete?

Delays & Clock Width

- Remember: with positive edge triggered registers, writing to a register must always complete (end) a clock cycle
- Clock width > max sum of delays and access times in any set of steps that could occur in a clock cycle
- Example Question

1.	PC output stable	20 ps
2.	Wire delay from PC to Addr of Memory	250 ps
3.	Memory read time	1500 ps
4.	Wire delay from Dout of Memory to IR	250 ps
5.	Setup time for IR	20 ps
6.	Hold time for IR for input to stabilize	20 ps
7.	Wire delay from IR to Register file	250 ps
8.	Register file read	500 ps
9.	Wire delay from Register file to input of ALU	250 ps
10.	Time to perform ALU operation	100 ps
11.	Wire delay from ALU output to Register file	250 ps
12.	Time for writing into a Register file	500 ps

Bus Design

- In the previous example, we connected elements to each other as we needed them to implement the ADD instruction
  - Could we conceivably extend this to the entire ISA? In other words, can we just connect all the elements to each other as needed for each instruction? **Should** we do this?
- Better idea - have one or more common **buses** shared by all the elements
  - Why is this not decreasing performance?
  - This leads to certain considerations
    - WrEn lines
    - Drivers (tri-state buffers)

Questions?