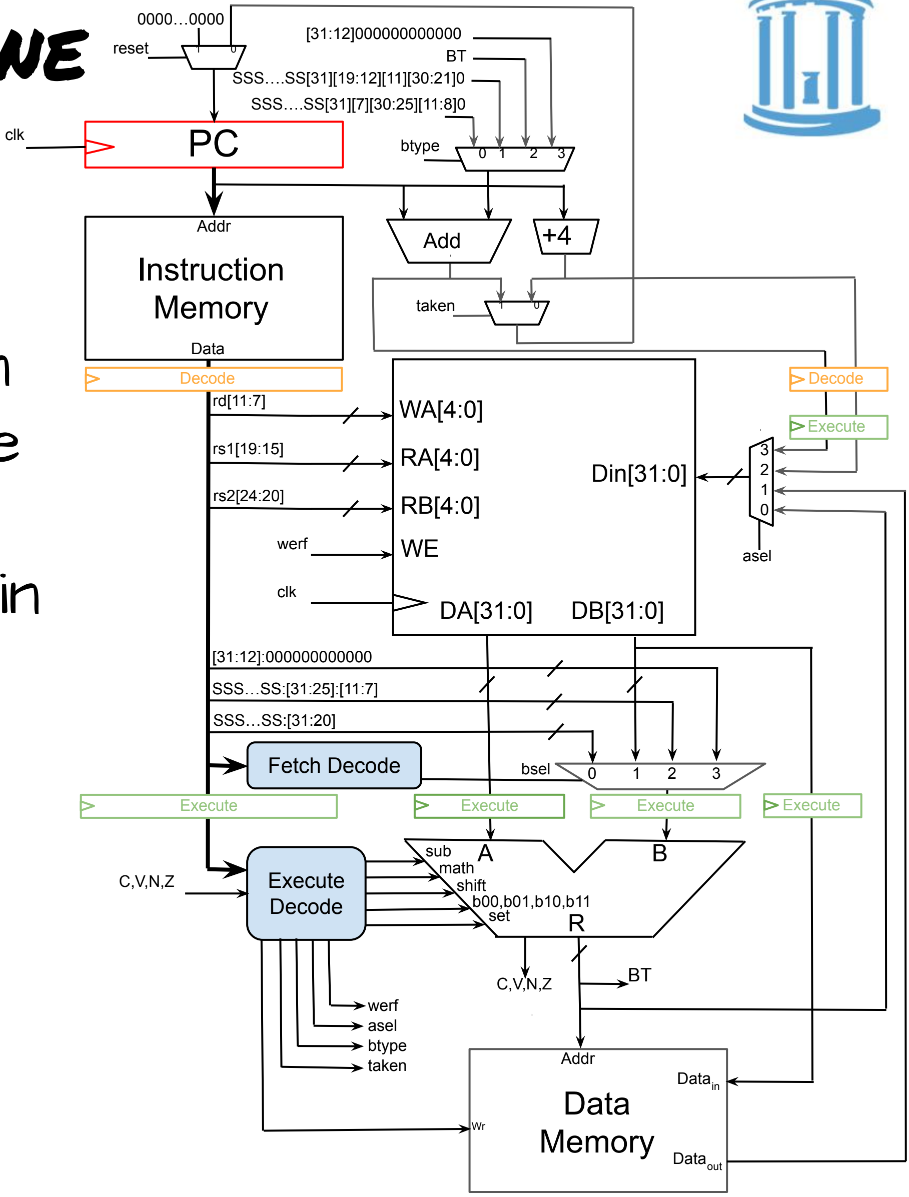
Pipeline Hazards

- Second Midterm on Tuesday
- DON'T ignore this lecture!

RISC-V 3-STAGE PIPELINE

- Fetch, Decode, and Execute Stages
- Instructions are decoded in both the Fetch and Decode stages
- Register ports are "Read" in the Decode stage and "Written" at in the end of the Execute stage
- PC+4, and PC relative calculations are "delayed" for use in later stages



Simple Instruction flow

Consider the following instruction sequence:

```
...
sub     t1,t1,t2
addi    t2,t2,2
andi    t0,t0,1
slt     t2,t2,t0
```

Instruction becomes available at the end of the Fetch stage

Operands at the end of Decode

Destination and PSR are updated at the end of Execute

Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0		
Decode		sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0	
Execute			sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0

t1 t2

t2 2

t0 1

t2 t0

t1 t2 t0 t2

Pipeline Control Hazards

Pipelining HAZARDS are situations where the next instruction cannot execute in the next clock cycle. There are two forms of hazards, CONTROL and STRUCTURAL.

Consider the instruction sequence shown:

```
  ...
loop:  add  t0,t0,t0
       addi t1,t1,-1
       blt  t1,x0,loop
       srai t0,t0,8
       sub   t1,t0,t1
```

Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	srai t0,t0,8	sub t1,t0,t1	add t0,t0,t0
Decode		add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	srai t0,t0,8	???
Execute			add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	???

When the branch instruction reaches the execute stage the next 2 instructions have already been fetched!

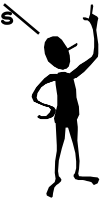
Branch Fixes

Problem: Two instructions following a branch are fetched before the branch decision is made (to take or not to take)

Solutions:

1. Program around it. Define the ISA such that the branch does not take effect until after instructions in the "DELAY SLOTS" complete. This is how MIPS pipelines work. It leads to ODD looking code in tight (short) loops. Of course you could always put NOPs in the delay slots.
2. Detect the branch decision as early as possible, and ANNUL instructions in the delay slots. This is what RISC-V does.

Early Detect and Annul



It helps that the ALU is not used by branch instructions

We can detect branch and jump instructions in the Decode stage. The decision to branch is decided no later than the current instruction in the Execute stage. Thus, we could make The branch decision in the Decode stage. We then annul the following instruction by disabling WERF and PSR updates! Making the next instruction a NOP!

```
...
loop:  add  t0,t0,t0
       addi t1,t1,-1
       blt  t1,x0,loop
       srai t0,t0,8
       sub   t1,t0,t1
```

Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	srai t0,t0,8	sub t1,t0,t1	add t0,t0,t0
Decode		add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	srai t0,t0,8	
Execute			add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	**NOP**

If we detect the branch in the decode stage then the PSR state of the instruction in the Execute stage can be combined to change the next PC.

The cost of taken branches

When an RISC-V branch is ***taken*** the branch instructions are effectively ***2 cycles rather than 1*** when they aren't. In a MIPS-like instruction set, one can often fill the delay slots with useful instructions, but they are executed whether or not the branch is taken.

The RISC-V approach is easier to understand, and since it does not "EXPOSE" the pipeline, it also allows for an alternative number of pipeline stages to be implemented in future designs, while conserving code compatibility.

Lastly, using RISC-V, many ***conditional branches can be eliminated using the condition execution***, which pipelines beautifully!

Structural Pipeline Hazards

There's another problem with our code fragment!

The destination register of instructions are written at the end of the Execute stage. However the following instruction might use this result as a source operand.

```
...
loop:  add   t0,t0,t0
       addi  t1,t1,-1
       blt   t1,x0,loop
       srai  t0,t0,8
       sub    t1,t0,t1
```

Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	srai t0,t0,8	sub t1,t0,t1	add t0,t0,t0
Decode		add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	srai t0,t0,8	
Execute			add t0,t0,t0	addi t1,t1,-1	blt t1,x0,loop	

The "BLT" instruction needs to access the contents of t1 before it is actually written are the end of i+3

Data Hazards

Problem: When a register source is needed from a later stage of the pipeline before it is written.

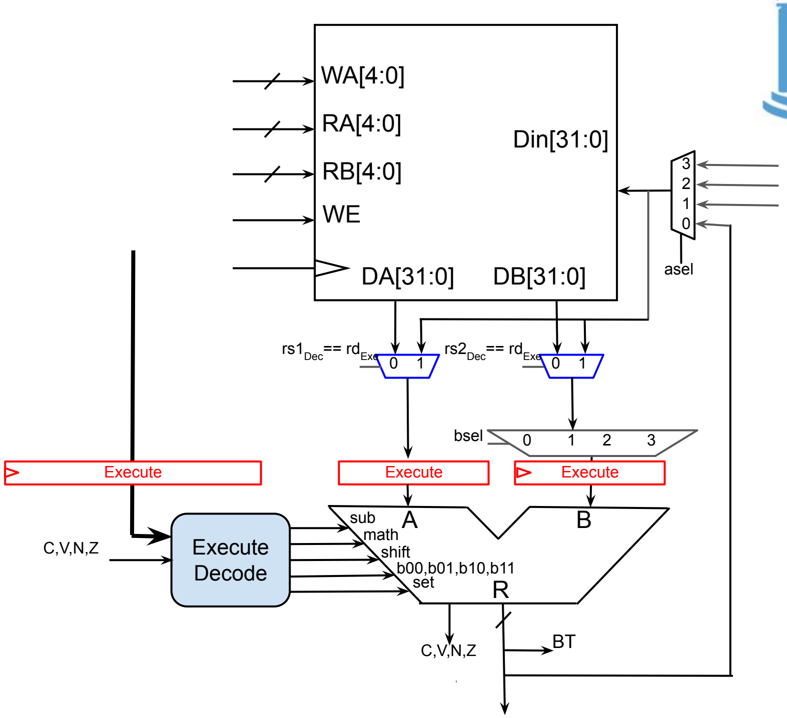
Solutions:

1. Program around it. One could document the weird semantics-- "You can't reference the destination register of an instruction in the immediately following instruction." Would make make assembly language even harder to understand. Would expose the pipeline, once again making future improvements difficult to implement while maintaining code compatibility.
2. Hardware bypass multiplexers.

Source Bypassing

The idea here is to also load the value about to be saved in the destination register into the pipeline registers that hold the ALU operands.

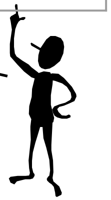
We will also need bypass MUXes on the StrReg and BXreg pipeline registers.



Pipeline

	i+2	i+3	i+4
Fetch	blt t1,x0,loop	srai t0,t0,8	sub t1,t0,t1
Decode	addi t1,t1,-1	blt t1,x0,loop	srai t0,t0,8
Execute		addi t1,t1,-1	blt t1,x0,loop

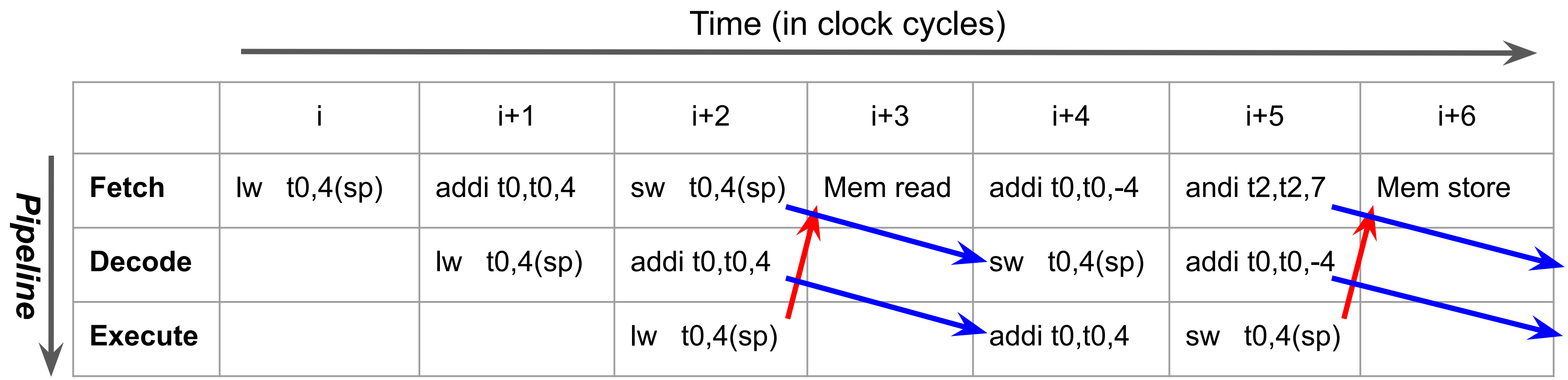
The new value for t1 will be computed just prior to the rising clock edge between i+3 and i+4, we can take the output of the ALU and provide it to the pipeline register and register file simultaneously



Load/Store Stalls

Load and Store memory accesses are the actual **bottleneck** of the RISC-V pipeline. Also, recall that instructions and load/stores actually come from the same memory. Thus, we need to stall instruction fetching to allow for loads and stores.

```
...
loop:  lw   t0,4(sp)
       addi t0,t0,4
       sw   t0,4(sp)
       addi t0,t0,-4
       andi t2,t2,7
```



Time (in clock cycles)

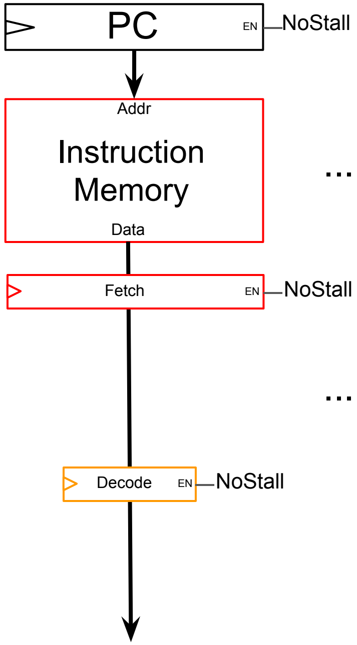
Pipeline

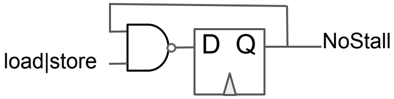
	i	i+1	i+2	i+3	i+4	i+5	i+6
Fetch	lw t0,4(sp)	addi t0,t0,4	sw t0,4(sp)	Mem read	addi t0,t0,-4	andi t2,t2,7	Mem store
Decode		lw t0,4(sp)	addi t0,t0,4		sw t0,4(sp)	addi t0,t0,-4	
Execute			lw t0,4(sp)		addi t0,t0,4	sw t0,4(sp)	

Load/Store stall implementation

Disable loading of pipeline registers for one clock when a load or store instruction reaches the execute stage.

1. Adding enable lines to the PC and pipeline registers on the control path
2. A simple 2-state state machine to stall the pipeline for 1 state to allow for the load/store memory cycle.





Where does this leave us

Overall we can now nearly triple the clock rate. Instructions have a throughput of one-per-clock with the following caveats:

1. Taken branches take 2 cycles.
2. Loads and store take 2 cycles.

3x speed up:
100 MHz clock
now 300 MHz

You can pipeline an RISC-V CPU even more. There exist implementations with 7, 8, and 9 pipeline stages. But the overhead of bypass paths and stall cases increase.

Reality vs Specmanship

Assuming approximately 10% of instructions executed are branches, and of those 80% of the time they are taken, and 15% of instruction executed are loads or stores, what sort of real speed up do we expect?

$\text{Perf}_{\text{before}} = (100) * 1 = 100 \text{ Clocks} * 10 * 10^{-9} \text{ sec/clock} = 1000 * 10^{-9} \text{ secs}$

$\text{Perf}_{\text{after}} = (10)((0.8) * 2 + (0.2) * 1) + 15 * 2 + 75 * 1 = 123 \text{ Clocks}$

$123 * 3.333 * 10^{-9} \text{ sec/clock} = 410 * 10^{-9} \text{ secs}$

$$\text{Speedup} = \frac{\text{Perf}_{\text{before}}}{\text{Perf}_{\text{after}}} = 1000/410 = 2.439 \text{ X}$$

Next time

It appears memory access time is our real bottleneck. What tricks can be applied to improving CPU performance in this case?

- Interleaving
- Block-transfers
- Caching

BEING FIVE A BOY AND HIS BLOG
BY GEORGE SFARNAS © 2009
MY COMPUTER'S BEEN RUNNING SLOW LATELY, MY DAD SAID IT NEEDS MORE RAM!
IT CREEPS ALONG, THEN FREEZES, THEN STUTTERS, THEN STOPS, THEN CREEPS ALONG AGAIN, THEN FREEZES...
HMMM, MY COMPUTER SOUNDS LIKE MY GRANDFATHER... MAYBE GRANDPA NEEDS MORE RAM
WWW.BEINGFIVE.COM
SFARNAS