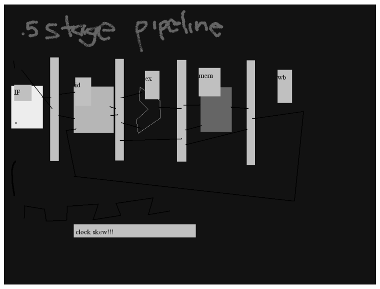
CS61C Midterm 3 Review

Summer 2004
Ben Huang
Pooya Pakzad
Navtej Sadhal

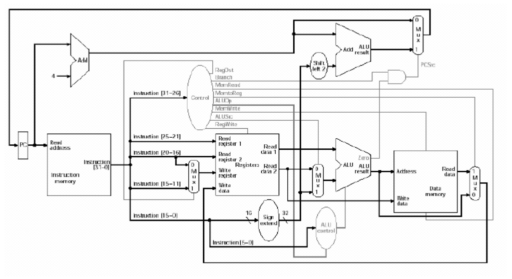
Overview

- CPU Design
- Pipelining
- Cache

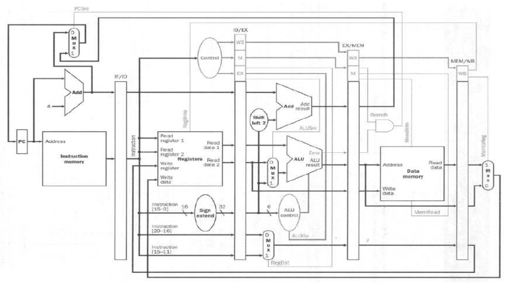
CPU Design



CPU Design



CPU Design



CPU Design

What changes need to be made to our CPU if we wanted to remove the ability to specify an offset for memory access instructions.

So the instruction:

```
    lw      $t0, 12($t1)
```

would become:

```
    addi    $at, $t1, 12
    lw      $t0, $at
```

What change could you make to the datapath to improve latency?

CPU Design

Combine ALU and memory stages in the datapath for a total of 4 stages because no instruction needs all 5 stages now.

CPU Design

What changes need to be made to our CPU if we wanted to add the instruction branch and increment (bincr):

bincr $t0, $t1, immed

that branches (unconditionally) to PC+4+(immed*4) and increments $t1 by $t0.

CPU Design

Assume a single cycle datapath:

In order to unconditionally branch, add a control signal (called uncondbranch) that is high on a bincr instruction. `Or` *this signal with PCSrc and send the output as the select signal for the mux that determines whether we choose PC+4 or the branch PC.*

In order to increment $t1 by $t0, let $t0 be rs and $t1 be rt (this is obviously an I-format instruction). Then choose: RegDst = 0, ALUSrc = 0, ALUOp = add, MemtoReg = 0, MemRead/Write = 0, and RegWrite = 1

CPU Design

What changes need to be made to our CPU if we wanted to add the instruction test and set (tas):

tas $t0, immed($t1)

that loads the value at address (immed+$t1) into $t0, and sets the same memory location to 1.

CPU Design

Assume a single cycle datapath:

This is just like a lw (all the signals are the same), except MemWrite is also high because we are writing to memory. We also need (1) a mux before the port writeData (in the memory module) to choose between the value 1 or readData2 and (2) a control signal to select which input to choose.

What is required of memory in order for this to work on the single cycle datapath? What would happen if we considered the pipelined datapath and we were required to read/write to memory in the MEM stage?

Pipeline

- IF | ID | EX | MEM | WB
- Pipelining allows us to decrease critical path decreasing clock cycle time
  - Increases throughput
  - But also increases latency

Pipeline

- Insert nop instructions to make this code safe for a pipeline without forwarding or hazard detection. Assume the register file is written on the negative edge of the CLK and branches are resolved in the second stage. The following code is designed for the single cycle MIPS processor:

```
lw      $t2, 4($0)
addi    $t4, $t2, 1
addi    $t3, $t3, 1
addi    $t0, $0, 20
addi    $t0, $t0, 4
lw      $t1, 0($t0)
beq     $t0, $t1, L1
add     $t0, $t0, $t1
add     $t3, $t2, $t1
sw      $t0, 4($t0)
lw      $t2, 4($t0)
beq     $t2, $0, L2
```

Pipeline

- Insert nop instructions to make this code safe for a pipeline without forwarding or hazard detection. Assume the register file is written on the negative edge of the CLK and branches are resolved in the second stage. The following code is designed for the single cycle MIPS processor:

```
lw      $t2, 4($0)
nop
nop
addi    $t4, $t2, 1
addi    $t3, $t3, 1
addi    $t0, $0, 20
nop
nop
addi    $t0, $t0, 4
nop
nop
lw      $t1, 0($t0)
nop
nop
beq     $t0, $t1, L1
nop
add     $t0, $t0, $t1
add     $t3, $t2, $t1
nop
sw      $t0, 4($t0)
lw      $t2, 4($t0)
nop
nop
beq     $t2, $0, L2
nop
```

Pipeline

- Now assuming there is forwarding but no hazard detection, remove any nop instructions you can:

```
lw      $t2, 4($0)
nop
nop
addi    $t4, $t2, 1
addi    $t3, $t3, 1
addi    $t0, $0, 20
nop
nop
addi    $t0, $t0, 4
nop
nop
lw      $t1, 0($t0)
nop
nop
beq     $t0, $t1, L1
nop
add     $t0, $t0, $t1
add     $t3, $t2, $t1
nop
sw      $t0, 4($t0)
lw      $t2, 4($t0)
nop
nop
beq     $t2, $0, L2
nop
```

Pipeline

- Insert nop instructions to make this code safe for a pipeline without forwarding or hazard detection. Assume the register file is written on the negative edge of the CLK and branches are resolved in the second stage. The following code is designed for the single cycle MIPS processor:

```
lw      $t2, 4($0)
nop
addi    $t4, $t2, 1
addi    $t3, $t3, 1
addi    $t0, $0, 20
addi    $t0, $t0, 4
lw      $t1, 0($t0)
nop
nop
beq     $t0, $t1, L1
nop
add     $t0, $t0, $t1
add     $t3, $t2, $t1
sw      $t0, 4($t0)
lw      $t2, 4($t0)
nop
nop
beq     $t2, $0, L2
nop
```

Pipeline

- Now reorder the instructions to eliminate the existing nop instructions

```
lw      $t2, 4($0)
nop
addi    $t4, $t2, 1
addi    $t3, $t3, 1
addi    $t0, $0, 20
addi    $t0, $t0, 4
lw      $t1, 0($t0)
nop
nop
beq     $t0, $t1, L1
nop
add     $t0, $t0, $t1
add     $t3, $t2, $t1
sw      $t0, 4($t0)
lw      $t2, 4($t0)
nop
nop
beq     $t2, $0, L2
nop
```

Pipeline

- Now reorder the instructions to eliminate the existing nop instructions

```
addi    $t0, $0, 20
addi    $t0, $t0, 4
lw      $t1, 0($t0)
lw      $t2, 4($0)
addi    $t3, $t3, 1
beq     $t0, $t1, L1
addi    $t4, $t2, 1
add     $t0, $t0, $t1
sw      $t0, 4($t0)
lw      $t2, 4($t0)
add     $t3, $t2, $t1
nop
beq     $t2, $0, L2
nop
```

Caches

- Memory Hierarchy
- Spatial Locality, Temporal Locality
- Cache Parameters
  - Associativity
  - Block Size
  - Number of lines

Determining Cache Parameters

- Determine the geometry of this cache given the following access times (in ns) for iterations through an array with the specified size and stride:

Stride size (bytes)

Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Determining Cache Parameters

- Hit Time: ~20ns
- Miss Time: ~80ns

Stride size (bytes)

Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Determining Cache Parameters

- Cache Size: What's the biggest array that always hits?

Stride size (bytes)

Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Determining Cache Parameters

- Cache Size: What's the biggest array that always hits?

Stride size (bytes)

Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Cache Size: 32kB

Determining Cache Parameters

- Block Size: What's the smallest stride size that always misses?

Stride size (bytes)

Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Cache Size: 32kB

Determining Cache Parameters

- Block Size: What's the smallest stride size that always misses?

Stride size (bytes)

Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Cache Size: 32kB Block Size: 32B

Determining Cache Parameters

- Associativitiy: What multiple of stride size is the array size when we stop missing and start hitting again?

Stride size (bytes)

Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Cache Size: 32kB Block Size: 32B

Determining Cache Parameters

- Associativitiy: What multiple of stride size is the array size on the last stride size that misses every access?

Stride size (bytes)

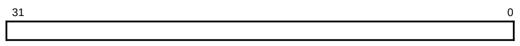
Array Size (bytes)

	4	8	16	32	64	2k	64k	128k	256k
4k	21	20	22	20	19	20			
8k	23	24	18	19	22	23			
16k	23	24	21	18	21	19			
32k	22	21	22	20	21	21			
64k	28	35	52	81	82	81	22		
128k	29	34	53	82	79	80	20	21	
256k	27	36	52	80	82	78	81	22	20
512k	28	35	51	82	81	81	80	79	19

Cache Size: 32kB Block Size: 32B
Associativity: 2 way

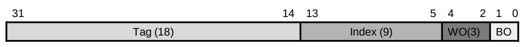
Cache Geometry and Addressing

- How do we divide up the address into index, tag, byte offset, and word offset for the preceding cache given a 32 bit word aligned address?
  - 32kB size, 32B blocks, 2-way associative



Cache Geometry and Addressing

- How do we divide up the address into index, tag, byte offset, and word offset for the preceding cache given a 32 bit word aligned address?
  - 32kB size, 32B blocks, 2-way associative



Cache Behavior

- Given the preceding cache (32kB size, 2-way associative, 32B blocks, LRU replacement), indicate whether the following accesses will hit or miss and the type of miss:

Address hex, byte:	00	08	40	80	44	4000	8000	8008	4010
H/M:									
Miss type:									

Cache Behavior

- Given the preceding cache (32kB size, 2-way associative, 32B blocks, LRU replacement), indicate whether the following accesses will hit or miss and the type of miss:

Address hex, byte:	0	8	40	80	44	4000	8000	8008	4	4010
H/M:	M	H	M	M	H	M	M	H	M	M
Miss type:	Com		Com	Com		Com	Com		Conf	Conf