

Overview

- CPU Design
- Pipelining
- Cache









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. O_T this signal with PCSr 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 n forward the neg second	op ins ing or ative stage	structions to make this code safe for a pipeline without hazard detection. Assume the register file is written on edge of the CLK and branches are resolved in the h. The following code is designed for the single cycle
	MIPS p	roces	sor:
	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,	St0, St1
	add	\$t3,	\$t2, \$t1
	SW	\$t0,	4(\$t0)
	lw	\$t2,	4(\$t0)
	beq	\$t2,	\$0, L2

			Pipe	line		
•	Insert ne forward the nega second MIPS pe	op ins ing or ative stage	tructions to make thi hazard detection. As edge of the CLK and the following code sor:	s code s ssume th branche is desig	afe fo ne reg es are ned fo	or a pipeline without gister file is written on e resolved in the for the single cycle
	lw lw	St 2	4(\$0)	non		
	1.W	ψυ2,	4(00)	nop		
	nop			bea	ŝt0,	Stl. Ll
	addi	St 4	\$t2 1	nop		
	addi	¢+2	¢+2 1	add	\$t0,	\$t0, \$t1
	addi	¢=0	¢0, 20	add	\$t3,	\$t2, \$t1
	auui	φu0,	ŞU, 20	nop		
	пор			SW	\$t0,	4(\$t0)
	nop			lw	\$t2,	4(\$t0)
	addi	ŞtO,	\$t0, 4	nop		
	nop			nop		
	nop			beq	\$t2,	\$0, L2
	lw	\$t1,	0(\$t0)	nop		

	Р	ipeline
 Now as any nop 	ssuming there is forw p instructions you ca	varding but no hazard detection, remove in:
lw nop addi addi addi nop nop addi nop	\$t2, 4(\$0) \$t4, \$t2, 1 \$t3, \$t3, 1 \$t0, \$0, 20 \$t0, \$t0, 4	nop hop add \$t0, \$t1, L1 nop add \$t0, \$t0, \$t1 add \$t2, \$t1 nop sw \$t0, 4(\$t0) lw \$t2, 4(\$t0) nop
lw	\$t1, 0(\$t0)	beq \$t2, \$0, L2 nop

	Pip	beline	;	
 Insert r forward the neg second MIPS p 	nop instructions to mak ding or hazard detection gative edge of the CLK I stage. The following o processor:	e this code n. Assume and branch code is desi	safe for the regines are gned fo	r a pipeline without ster file is written on resolved in the r the single cycle
lw	\$t2, 4(\$0)			
nop addi addi addi lw nop nop beq nop add	<pre>\$t4, \$t2, 1 \$t3, \$t3, 1 \$t0, \$0, 20 \$t0, \$t0, 4 \$t1, 0(\$t0) \$t0, \$t1, L1 \$t0, \$t1, L1 \$t0, \$t1, \$t1 \$t2, \$t2, \$t1</pre>	sw lw nop nop beg nop	\$t0, \$t2, \$t2,	4(\$LU) 4(\$LO) \$0, L2



Now re	P		the existing nop	
addi addi lw lw addi beq addi add	<pre>tions \$10, \$0, 20 \$10, \$10, 4 \$11, 0(\$10) \$12, 4(\$0) \$13, \$13, 1 \$10, \$11, L1 \$14, \$12, 1 \$10, \$11, \$11</pre>	sw lw add nop beq nop	\$t0, 4(\$t0) \$t2, 4(\$t0) \$t3, \$t2, \$t1 \$t2, \$0, L2	

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)

: 2	14				-			1201	2000
		20	22	20	19	20			
: 2	23	24	18	19	22	23			
ik 2	23	24	21	18	21	19			
2k 2	22	21	22	20	21	21			
lk 2	28	35	52	81	82	81	22		
28k 2	29	34	53	82	79	80	20	21	
56k 2	27	36	52	80	82	78	81	22	20
2k 2	28	35	51	82	81	81	80	79	19
	ik 2 !k 2 !k 2 !k 2 !k 2 !k 2 !k 2 !k 2 !	ik 23 ik 22 ik 28 28k 29 56k 27 22k 28	ik 23 24 ik 22 21 ik 28 35 28k 29 34 36k 27 36 12k 28 35	ik 23 24 21 ik 22 21 22 ik 28 35 52 i8k 29 34 53 i6k 27 36 52 i2k 28 35 51	k 23 24 21 18 k 22 21 22 20 k 28 35 52 81 k 29 34 53 82 k 29 34 53 82 k 29 36 52 80 24 28 35 51 82	k 23 24 21 18 21 k 22 21 22 20 21 k 28 35 52 81 82 k8 29 34 53 82 79 i6k 27 36 52 80 82 i2k 28 35 51 82 81	ik 23 24 21 18 21 19 ik 22 21 22 20 21 21 21 ik 28 35 52 81 82 81 ik 29 344 53 82 79 80 i6k 27 36 52 80 82 78 i2k 28 35 51 82 81 81	k 23 24 21 18 21 19 k 22 21 22 20 21 21 21 k 22 21 22 20 21 21 21 k 28 35 52 81 82 81 22 8k 29 34 53 82 79 80 20 66k 27 36 52 80 82 78 81 2k 28 35 51 82 81 81 80	ik 23 24 21 18 21 19 Image: Constraint of the system ik 22 21 22 20 21 21 21 ik 28 35 52 81 82 81 22 ik 28 36 52 81 82 78 80 20 21 ik 28 36 52 80 82 78 81 22 ik 28 36 52 80 82 78 81 22 ik 28 35 51 82 81 81 80 79

Determining Cache Parameters

• Hit Time: ~20ns

Miss Time: ~80ns
 Stride cize (butee)

			Strid	e size (i	oytes)					
		4	8	16	32	64	2k	64k	128k	256k
_	4k	21	20	22	20	19	20			
tes	8k	23	24	18	19	22	23			
Q	16k	23	24	21	18	21	19			
ze	32k	22	21	22	20	21	21			
y S	64k	28	35	52	81	82	81	22		
vra	128k	29	34	53	82	79	80	20	21	
٩	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?

			Stri	de size	(bytes)					
		4	8	16	32	64	2k	64k	128k	256k
	4k	21	20	22	20	19	20			
ŝ	8k	23	24	18	19	22	23			
2	16k	23	24	21	18	21	19			
2	32k	22	21	22	20	21	21			
	64k	28	35	52	81	82	81	22		
5	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





ב נו	Det	err ^{Size: W}	nin Vhaťs ti	ing ne sma	Ca llest str	ach	e F	Para ways m	ame	eters
			Stri	de size	(bytes))				
		4	8	16	32	64	2k	64k	128k	256k
	4k	21	20	22	20	19	20			
tes)	8k	23	24	18	19	22	23			
ą	16k	23	24	21	18	21	19			
Ze	32k	22	21	22	20	21	21			
ŝ	64k	28	35	52	81	82	81	22		
rra	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
	Cad	he Siz	ze: 32kE	3	Bloc	k Size:	32B			<u> </u>

L	Jel	en		ing	Ua	aCH	ег	ala	anne	lei
	Associa	ativitiy	What	multiple	of stric	de size	is the a	rray size	e when v	we stop
	missing	g and s	start hiti Stri	ting aga de size	(hytes)					
		4	8	16	32	64	2k	64k	128k	256k
	4k	21	20	22	20	19	20			
es)	8k	23	24	18	19	22	23			
ā	16k	23	24	21	18	21	19			
ZG	32k	22	21	22	20	21	21			
Š	64k	28	35	52	81	82	81	22		
E.	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
	Cac	he Siz	re: 32kF	3	Bloc	k Size:	32B			

C	Dete	ern	nini	ing	Ca	ach	e F	Para	ame	eters
•	Associa size tha	ativitiy: at miss	What r	nultiple y acces	of stric	le size	is the a	rray size	e on the	last stride
		4	0		(bytes)	64	24	644	1294	2564
	41	4	0	10	32	10	20	04K	120K	2006
(s	4K	21	20	10	20	19	20			
oyte	0K 16k	23	24	21	19	22	10	-		
e (r	TOK	23	24	21	10	21	19	-		
N.	32K	22	21	22	20	21	21	-	-	
≥	64k	28	35	52	81	82	81	22		
vrra	128k	29	34	53	82	79	80	20	21	
4	256k	27	36	52	80	82	78	81	22	20
	512k	28	35	51	82	81	81	80	79	19
	Cac Ass	he Siz ociativ	e: 32kE ity: 2 w	ay	Bloc	k Size:	32B			



```
- 32kB size, 32B blocks, 2-way associative
```





Cache Behavior

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

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