Review: (Write on the board before class)
1) Pipelining; individual instructoins may even be slower (latency) but average CPI close to one
2) Little change to processor (extra incrementor), big change to memory => 3 to 4 times speedup
3) Hazards that prevent speeup: Structural, Data, Control
4) Data further categorized: RAW, WAR, WAW
5) Showed how 2 extra buffers & Muxes at ALU resolve RAW Hazards in DLX

Today: Data, Control, Scheduling
Why no WAR & WAW in DLX?
All reads in ID stage (can't write first)
All writes in WB stage (can't write first)
Autoincrement would make it possible?
Yes for WAW; added writing at an earlier stage
No for WAR; reads still occur early in process, same time as following intnruction is fetched, so can't get a write completing before reads

Can't Solve loads with forwarding; see 6.10

Solution: HW interlock to garuentee execution, SW to scheudle to avoid interlocks
Always in SW called delayed load
6.13 shows how effective

Control: pipeline branches (PC changing)
Greater impact on performance
PC not changed until end of MEM => lots of stalls (see **6.16**)
3 clocks => 1/2 performance

2 part task:
1) determine branch taken or not as early as possible in pipeline
2) Computer PC for taken address as early as possible
Look at PC changing instruction in **Figure 6.4**
with enough hardware could move everything to ID phase **(6.17)**
Adder for address, conditional load of PC, register = 0,
decode in middle of ID
Improvement => 1 clock cycle stall

Brnach data: conditional about 50% taken, loops about 90% taken
conditional 11% to 17% of all instructions

Reducing penalty strategies
1) freeze pipeline: 1 clock cycle on any branch
2) Predict not taken: 0 penalty on untaken, 1 on taken; stop & restrt the pipeline
3) Delayed branch a la delayed load
    Smilar problem & solution:

How effective is compiler at scheduling? see p. 276, 6.22

Overall scheduling ub 6.24: 7% to 15% stalls (interger only)

What Makes Pipelines Hard?

Within Restartable interrupts (e.g., page fault):
several instructions in progress; must save propoer state & indicate cause of interrupt and WHICH INSTRUCTION caused it!

Dealyed branch => multiple PCs saved so can restart

Autoincrement addressing mode:
    WAW hazard; detection

Implicit condition codes; bottle neck since all can set; which is for a branch?

Multicycle operations: vary from 1 to 1000 clocks in an instruction
    Pipeline hte microcode! 6.9 shows another way
BREAK: Pickup Questionairre; Bring Book & Fix in class;
Known typos:
p. 60 Since α is usually 2 or larger, die costs are proportional to the third (or higher) power of the die area: Cost of die = f (Die area^3)
p. 147 Fig 4.5 CMPL length is 3 not 7
p. 156 JE IP-128<= name <= IP+128
p. 202 Rd => B;
p. 221 Fig 5.14; drop A<-Rd from Memory state;
    change Stores state from MDR <- A to MDR <- B
Learn 3 names
Next assignment? Exam Friday march 2; open book; no direct pointers; cross chapter questions ideal

Wednesday Feb 28 is review; getting award at class time; show tape on SPARC; Business aspects first then technical;

Stage	PC unit	Memory	Data path
IF	IMAR← PC; PC←PC+4;	IR←Mem[IMAR];	
ID			A ← Rs1; B ← Rs2;
EX			DMAR← A + $(IR_{16})^{16}$##$IR_{16..31}$; or ALU_{output} ← A *op* (B or $(IR_{16})^{16}$##$IR_{16..31}$); or ALU_{output}←PC + $(IR_{16})^{16}$##$IR_{16..31}$; cond← (Rs1 *op* 0); MDR←B;
	if (cond) PC←ALUoutput	MDR← Mem[DMAR] or Mem[DMAR]←MDR	
WB			Rd ← ALUoutput or MDR

FIGURE 6.3 The table shows the major functional units and what may happen in every pipe stage in each unit.

Stage	ALU instruction	Load or store instruction	Branch instruction
IF	IMAR←PC; IR←Mem[IMAR]; PC←PC+4;	IMAR←PC; IR←Mem[IMAR]; PC←PC+4;	IMAR←PC; IR←Mem[IMAR]; PC←PC+4;
ID	A←Rs1; B←Rs2;	A←Rs1; B←Rs2;	A←Rs1; B←Rs2;
EX	ALUoutput ← A *op* B; or ALUoutput←A *op* $((IR_{16})^{16}$##$IR_{16..31})$;	DMAR← A+ $((IR_{16})^{16}$##$IR_{16..31})$; MDR← B;	ALUoutput←PC + $((IR_{16})^{16}$##$IR_{16..31})$; cond←(Rs1 *op* 0);
MEM		MDR← Mem[DMAR]; or Mem[DMAR]←MDR;	if (cond) PC←ALUoutput;
WB	Rd ← ALUoutput;	Rd ← MDR;	

FIGURE 6.4 Events on every pipe stage of the DLX pipeline

`LW R1,32(R6)`	IF	ID	EX	**MEM**	WB		
`ADD R4,R1,R7`		IF	ID	**EX**	MEM		
`SUB R5,R1,R8`			IF	ID	**EX**		
`AND R6,R1,R7`				IF	**ID**		

FIGURE 6.10

Any instruction	IF	ID	EX	MEM	WB					
`LW R1,32(R6)`		IF	ID	EX	**MEM**	WB				
`ADD R4,R1,R7`			IF	ID	*stall*	**EX**	MEM	WB		
`SUB R5,R1,R8`				IF	*stall*	**ID**	EX	MEM	WB	
`AND R6,R1,R7`					*stall*	IF	ID	EX	MEM	WB

FIGURE 6.11

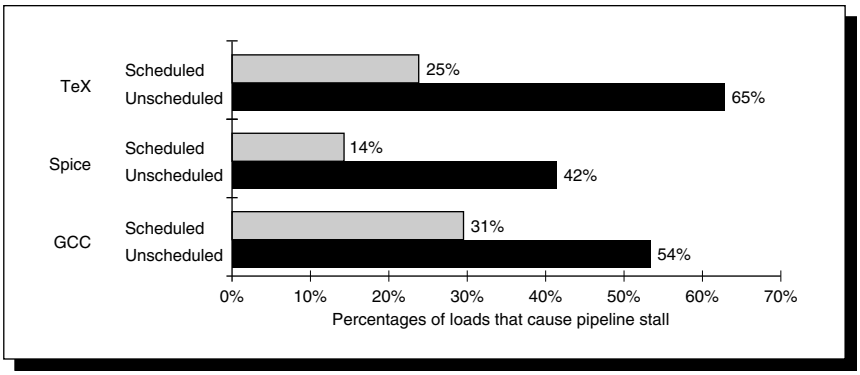

FIGURE 6.13 Percentage of the loads that result in a stall with the DLX pipeline.

Branch instruction	IF	ID	EX	MEM	WB					
Instruction *i*+1		IF	*stall*	*stall*	IF	ID	EX	MEM	WB	
Instruction *i*+2			*stall*	*stall*	*stall*	IF	ID	EX	MEM	WB
Instruction *i*+3				*stall*	*stall*	*stall*	IF	ID	EX	MEM
Instruction *i*+4					*stall*	*stall*	*stall*	IF	ID	EX
Instruction *i*+5						*stall*	*stall*	*stall*	IF	ID
Instruction *i*+6							*stall*	*stall*	*stall*	IF

FIGURE 6.16 Branch in DLX pipeline.

Pipe stage	Branch instruction
IF	IMAR← PC; IR←Mem[IMAR]; PC←PC+4;
ID	A← Rs1; B← Rs2; **BTA←PC+((IR$_{16}$)^{16}##** **IR$_{16..31}$)** **if (Rs1 *op* 0) PC←BTA**
EX	
MEM	
WB	

FIGURE 6.17 Revised pipeline structure (see Figure 6.4, page 256)

Branch instruction	IF	ID	EX	MEM	WB					
Instruction *i*+1		IF	IF	ID	EX	MEM	WB			
Instruction *i*+2			*stall*	IF	ID	EX	MEM	WB		
Instruction *i*+3				*stall*	IF	ID	EX	MEM	WB	
Instruction *i*+4					*stall*	IF	ID	EX	MEM	WB
Instruction *i*+5						*stall*	IF	ID	EX	MEM
Instruction *i*+6							*stall*	IF	ID	EX

Revised Branch in DLX pipeline.

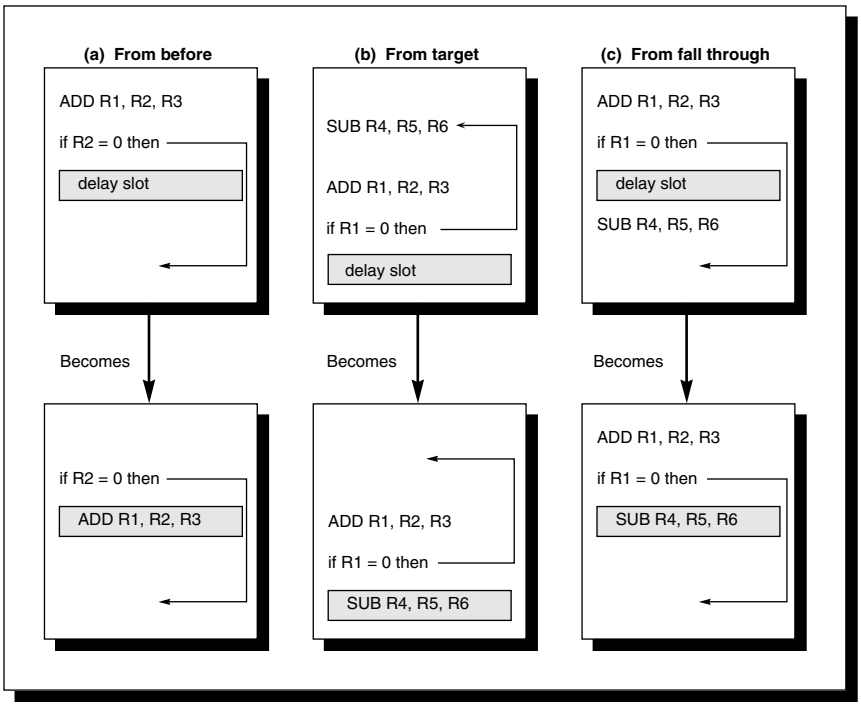

FIGURE 6.20 Scheduling the branch-delay slot.

Scheduling strategy	Requirements	Improves performance when?
(a) From before branch	Branch must not depend on the rescheduled instructions.	Always.
(b) From target	Must be OK to execute rescheduled instructions if branch is not taken. May need to duplicate instructions.	When branch is taken. May enlarge program if instructions are duplicated.
(c) From fall through	Must be OK to execute instructions if branch is taken.	When branch is not taken.

FIGURE 6.21 Delayed-branch–scheduling schemes and their requirements.

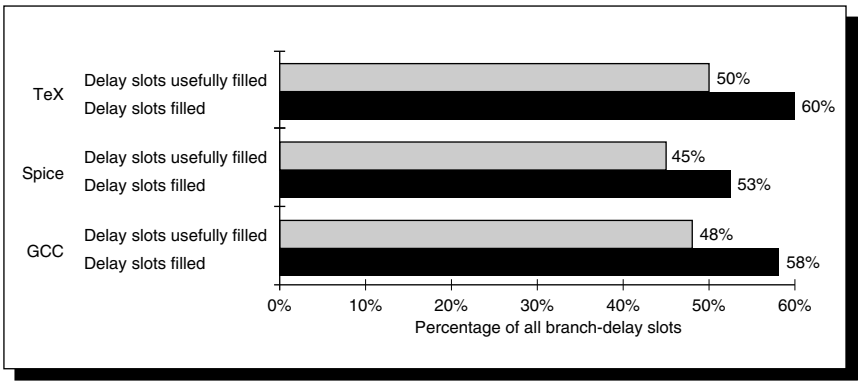

FIGURE 6.22 Frequency with which a single branch-delay slot is filled and how often the instruction is useful to the computation.

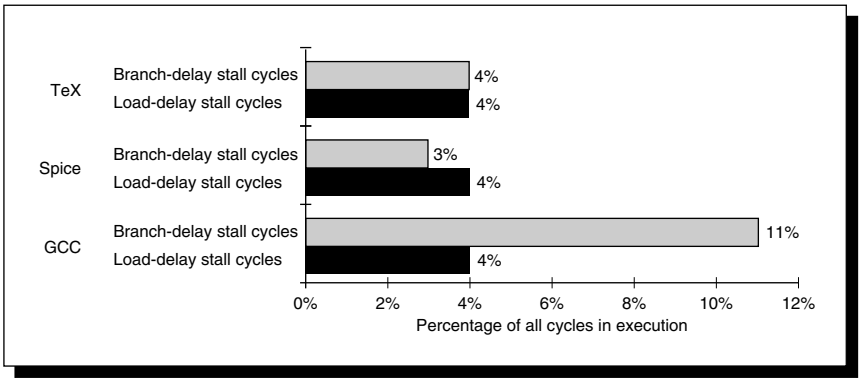

FIGURE 6.24 Percentage of the clock cycles spent on delays versus executing instructions.