Review: (Write on the board before class)
1) DLX, similar to many machines at end of 1980s
2) Opposite to ideas of the time: Always 32-bits, load-store, simple addressing modes and operations
3) Quick review of Microprogramming/Control to set the stage for Pipelining
4) Determines 2 o3 three parameters of performance
Time = IC * CPI * CCT
5) 5 basic steps of all instructions

Ch 5, 6, and 7 all based on DLX

Ch 5 Impementation

Interrupts

Invented for errors and I/O

No consistent naming accross machines

5 axes to evaluate the event:

1. **Synchronous versus asynchronous.** If the event occurs at the same place every time the program is executed with the same data and memory allocation, the event is synchronous. With the exception of hardware malfunctions, asynchronous events are caused by devices external to the processor and memory.
2. **User request versus coerced.** If the user task directly asks for it, it is a user-request event.
3. **User maskable versus user nonmaskable.** If it can be masked or disabled by a user task, the event is user maskable.
4. **Within versus between instructions.** This classification depends on whether the event prevents instruction completion by occurring in the middle of execution—no matter how short—or whether it is recognized between instructions.
5. **Resume versus terminate.** If the program's execution stops after the interrupt, it is a terminating event.

Look at categories (Figure 5.11 on page 217)

What is hard about interrupts?

Save state, proper PC, and indication of event at any clock cycle for within interrupts => restratable

e.g., VAX : 50 bytes of instructions and then a page fault
autoincrement means results changed before enter last phase
(history queue with registers values changed to undo inc/dec)

long running instructions takes (10 ms)

IBM solution MOVCL in 370: operand addresses specified in registers; make progress update registers pointint to source, desitnation, count; restrart with the new values (a complete copy)

VAX: orthogocal so operands addresses can be in memory;
operands loaded into register R21,R2,R3 and updated in place;
if interrupted set FPD bit in PSW to say that use values in registers when restart; otherwise, fetch addresses from memory again;

Calculate the speed of a few DLX instructions

Go through state diagram to calculate time
Decide on speed of memory (wait states or memory stall cycles)
Beginning of clock cycle sending PC to memory
May not be able to access memory and return data in one instruction

```
ADD R1,R2,R3
```

5 cycles plus memory reference time

```
LW R4,R1,#8
```

6 cycles plus 2 memory reference times

```
BNEZ R4,label
```

3 cycles untaken/4 cycles taken plus memory reference time

Total 14 cycles plus 4 memory references: if in cache perhaps 4*0 or 4*1 = 14/3 = 4.7 or 18/3 = CPI of 6.0
How calculate on average? Get mix!

BREAK
missing partners?
Next assignment? Ch 6 : 6.2, 6.3, 6.4
In the news:

Bring WCS advertisement!
IBM 16Mb DRAM at ISSCC
Yesterday IBM accounced RISC System/6000.
27 to 41 “MIPS” and 7.4 to 13 MFLOPS $13K to $93K
34 SPECmarks at high end
33% of market by 1993; 15% to 20% in 1992 or 1993 (2% now)
Last year DEC announced DECstation 3100: runs programs in less time than anycomputer they sell at any pricel bet in many programs faster than IBM 3090.

Microcode Format, program to interpret control
Do the Add instruction from the microcode

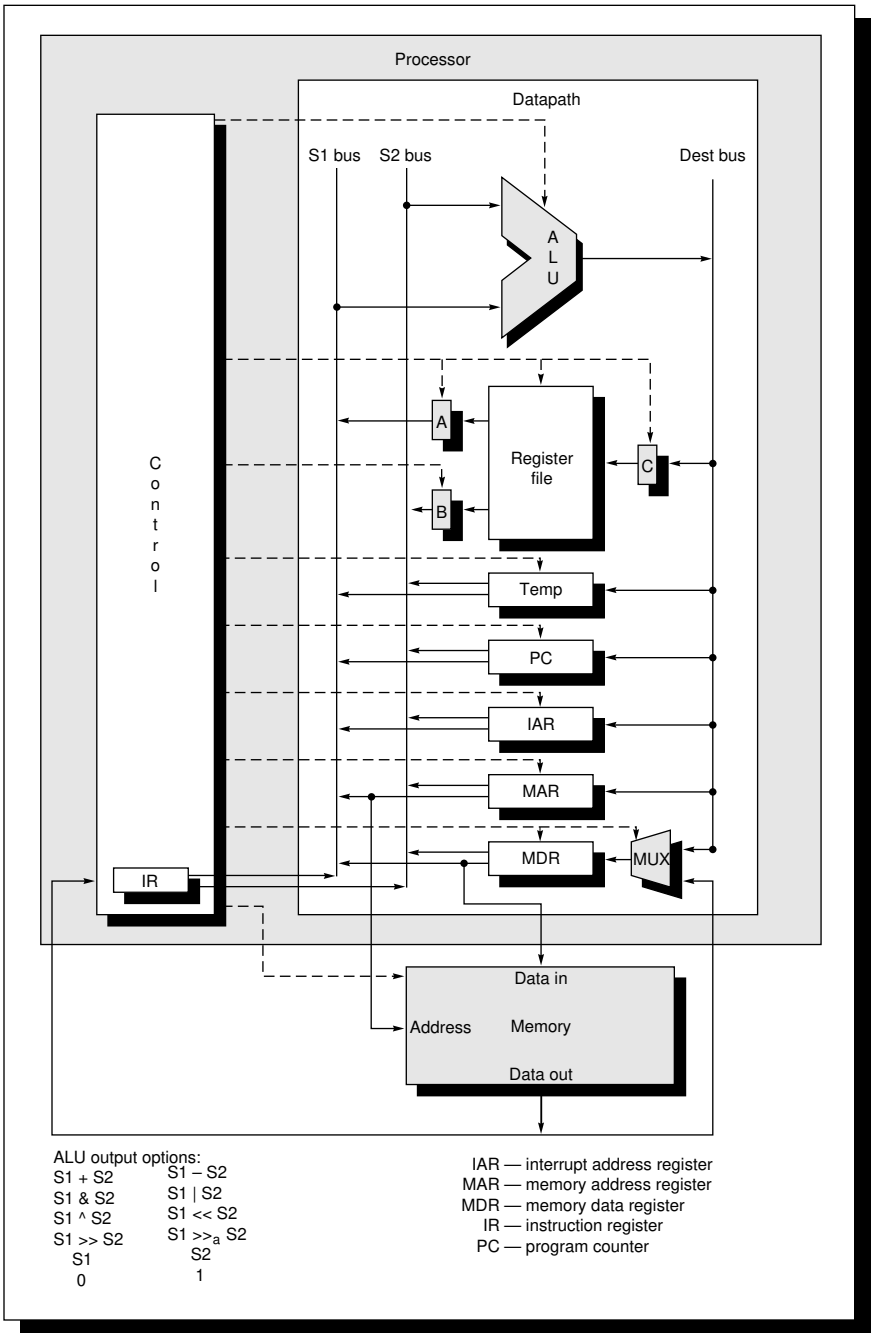
Talk about discussion questions:
5.20 [Discussion] <4,5.5,5.10> Hypothesis: If the first implementation of an architecture uses microprogramming, it affects the instruction set architecture. Why might this be true? Looking at examples in Chapter 4 or elsewhere, give supporting or contradicting evidence from real machines. Which machines will always use microcode? Why? Which machines will never use microcode? Why? What control implementation do you think the architect had in mind when designing the instruction set architecture?

5.21 [Discussion] <5.5,5.10> Wilkes invented microprogramming in large to simplify construction of control. Since 1980 there has been an explosion of computer-aided design software whose goal is also to simplify construction of control. Hypothesis: The advances in computer-aided design software have rendered microprogramming unnecessary. Find evidence to support and refute the hypothesis.

5.22 [Discussion] <5.10> The DLX instructions and the DLX microinstructions have many similarities. What would make it difficult for a compiler to produce DLX microcode rather than macrocode? What changes to the microarchitecture would make the DLX microcode more useful for this application?

	Synchronous vs. asynchronous	User request vs. coerced	User maskable vs. nonmaskable	Within vs. between instructions	Resume vs. terminat
I/O device request	Asynchronous	Coerced	Non maskable	Between	Resume
Invoking operating system service	Synchronous	User request	Non maskable	Between	Resume
Tracing instruction execution	Synchronous	User request	User maskable	Between	Resume
Breakpoint	Synchronous	User request	User maskable	Between	Resume
Integer arithmetic overflow	Synchronous	Coerced	User maskable	Within	Terminat
Floating-point arithmetic overflow or underflow	Synchronous	Coerced	User maskable	Within	Resume
Page fault	Synchronous	Coerced	Non maskable	Within	Resume
Misaligned memory accesses	Synchronous	Coerced	User maskable	Within	Terminat
Memory-protection violations	Synchronous	Coerced	Non maskable	Within	Terminat
Using undefined instructions	Synchronous	Coerced	Non maskable	Within	Terminat
Hardware malfunctions	Asynchronous	Coerced	Non maskable	Within	Terminat
Power failure	Asynchronous	Coerced	Non maskable	Within	Terminat

FIGURE 5.11 The events of Figure 5.9 classified using five categories.


Processor
Datapath
S1 bus
S2 bus
Dest bus
A
L
U
A
B
Register file
C
Control
Temp
PC
IAR
MAR
MDR
MUX
IR
Data in
Address
Memory
Data out
ALU output options:
S1 + S2
S1 – S2
S1 & S2
S1 | S2
S1 ^ S2
S1 << S2
S1 >> S2
S1 >>a S2
S1
S2
0
1
IAR — interrupt address register
MAR — memory address register
MDR — memory data register
IR — instruction register
PC — program counter

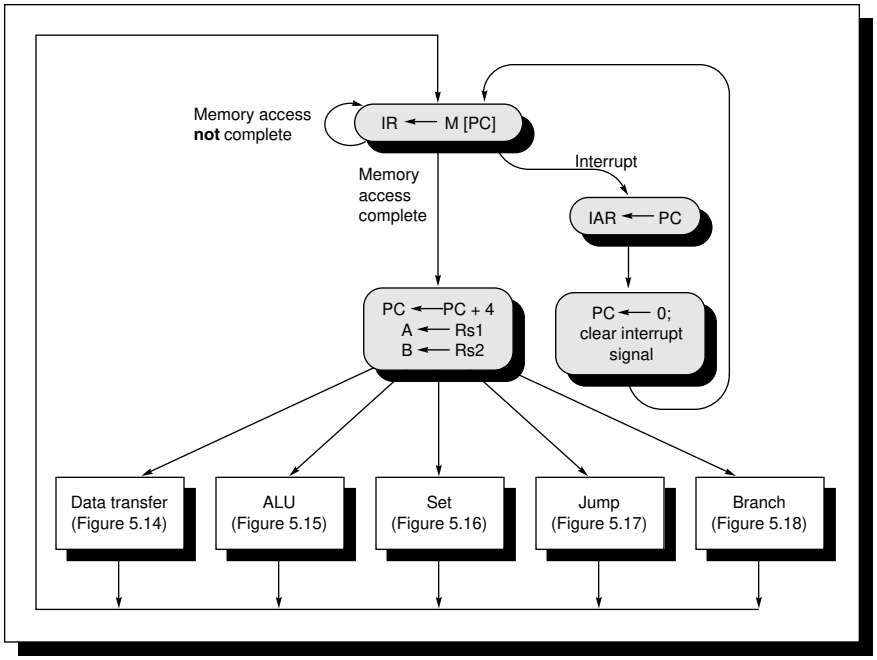


FIGURE 5.13 The top-level view of the DLX finite-state diagram for the non-floating-point instructions.

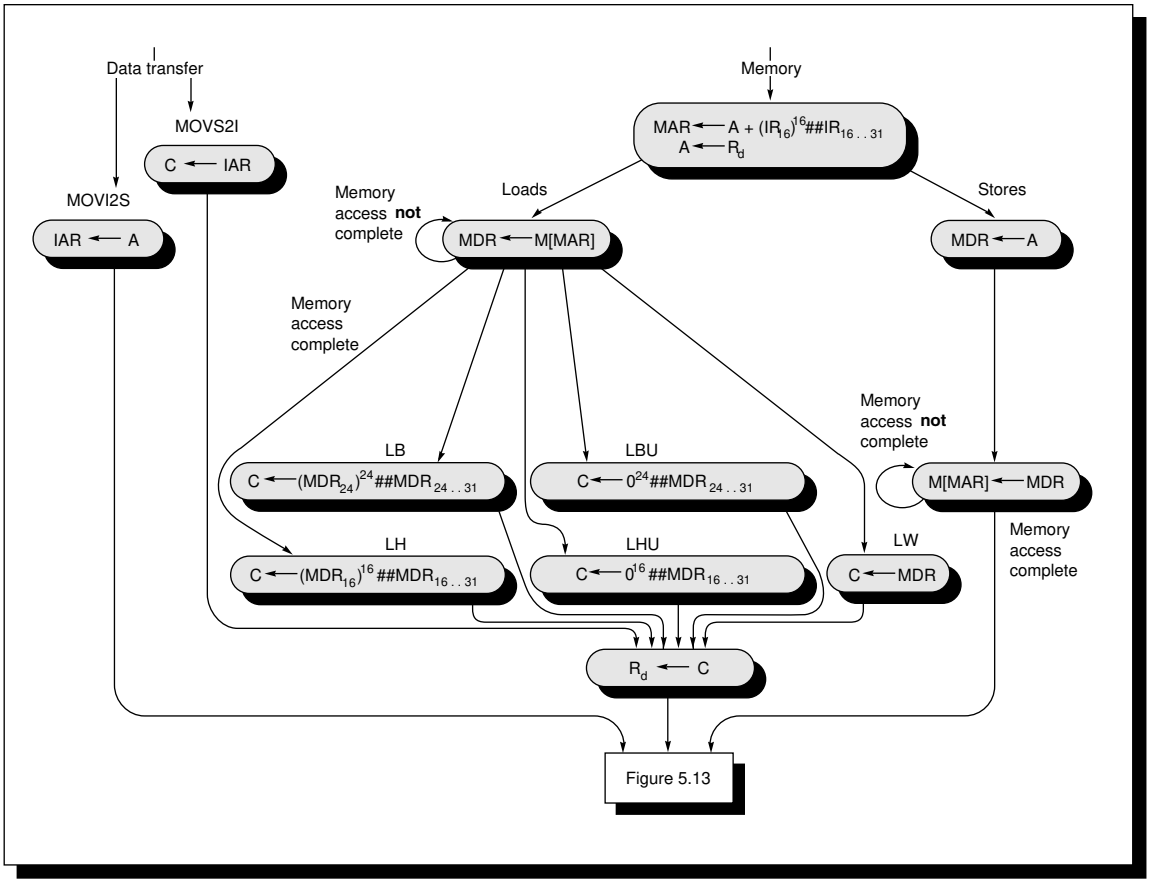


FIGURE 5.14

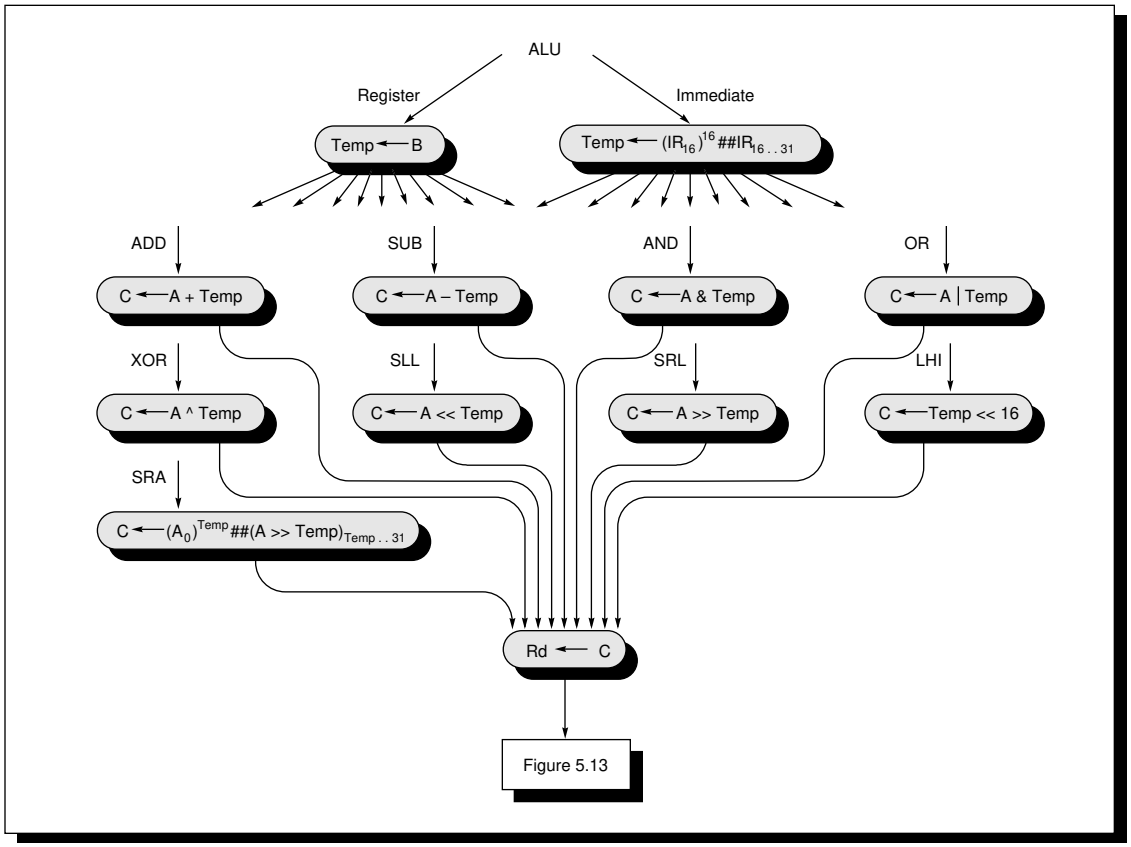


FIGURE 5.15 The execution and write-back states for the ALU instructions of DLX.

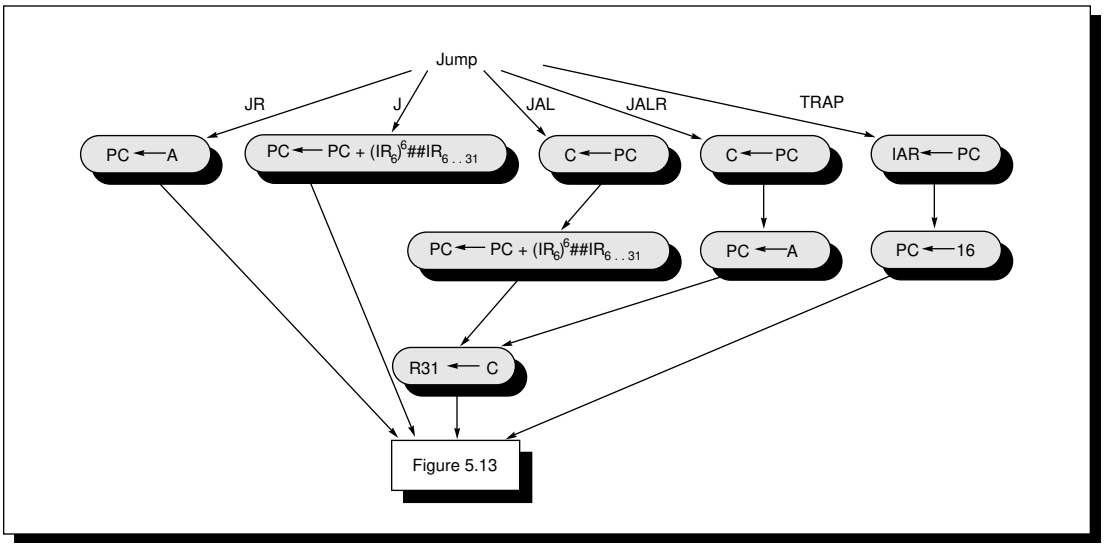


FIGURE 5.17

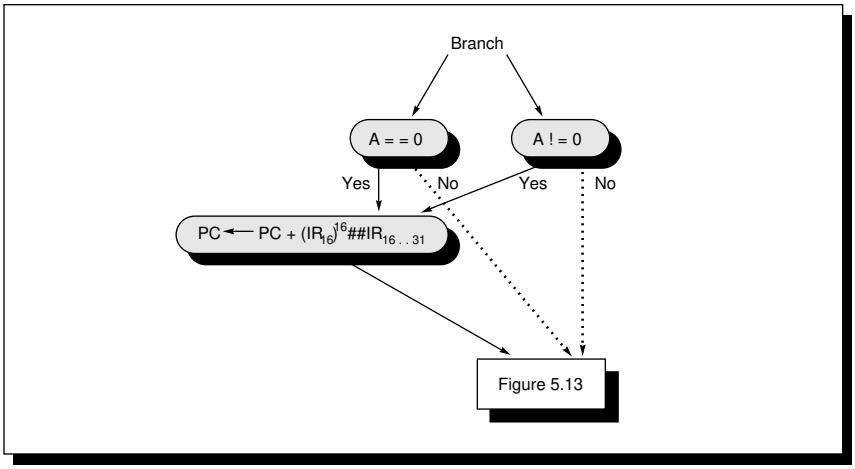


FIGURE 5.18 The execution states for the branch instructions of DLX.

Loads	8 * 21%	=	1.68
Stores	7 * 12%	=	0.84
ALU	6 * 37%	=	2.22
Set	7 * 6%	=	0.42
Jumps	4 * 2%	=	0.08
Jump and links	6 * 0%	=	0.00
Branch (taken)	5 * 12%	=	0.60
Branch (not taken)	4 * 11%	=	0.44
	Total CPI:		6.28

(page 225)

Loc	Label	Dest	ALU	S1	S2	C	Misc	Cond	Jump label	Commen
0	Ifetch:							Inter-rupt?	Intrpt	*Check interrupt*
1	Iloop:						Instr Read	Mem?	Iloop	*IR←M[PC wait for memory*
2		PC	ADD	PC	Const	4	AB←RF	Decode1		
…										
23	Reg:	Temp	Pass S2	B				Decode2		*source2 = reg*
…										
25	ADD/I:	C	ADD	A	Temp			Uncond	Write2	*ADD*
…										
34	Write2:						Rd←C	Uncond	Ifetch	*Write back & go fetch next instruction*

Microisntructions executed for ADD R1,R2,R3