

## Chapter 4 – Design Process

CS 333  
Fall 2006

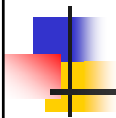
C  
S  
D  
A  
2/e

## Topics

- Review from last class
- Control unit design
- 2-bus and 3-bus designs

## Design Process Overview

- **Informal** description
- **Formal** RTN description (abstract RTN)
- **Block diagram** of architecture
- **Concrete** RTN steps
  - Different implementations possible
- **Hardware design** of blocks
- **Control sequences**
  - Control signals that allow data path to function properly
- **Control unit and timing**



## Previously in Class...

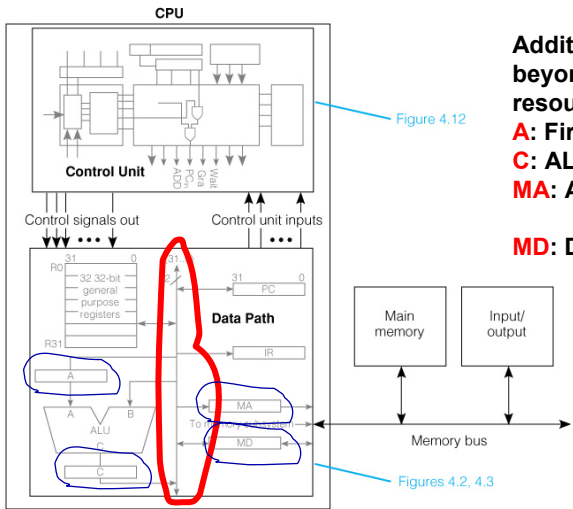
## Abstract and Concrete Register Transfer Descriptions

- The abstract RTN for SRC in Chapter 2 defines **“what,”** not **“how”**
- A concrete RTN uses a specific set of **real registers** and **buses** to accomplish the effect of an abstract RTN statement
- **Several** concrete RTNs could implement the same ISA
  - 1-bus
  - 2-bus
  - 3-bus

## Design Process

- Starting with formal description (**abstract RTN**)
- So far, for 1-bus architecture:
  - Write **concrete RTN steps** consistent with the architecture
  - **Keep track of demands** made by concrete RTN on the hardware
- Design **data path hardware** and identify needed **control signals**
- Today, design a **control unit** to generate control signals
- Control unit and timing
  - Minimum **clock period**
  - Minimum **strobe signal**
  - Minimum **gate signal**

Fig. 4.1 Block Diagram of 1-bus SRC



Additional resources needed beyond programmer-visible resources:

- A:** First operand to ALU
- C:** ALU result
- MA:** Address to request from main memory bus
- MD:** Data to/from memory

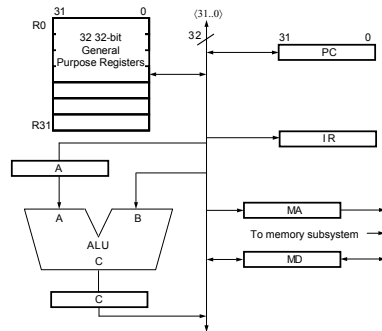
Figure 4.12

Figures 4.2, 4.3

Copyright © 2004 Pearson Prentice Hall, Inc.

Constraints Imposed by the Microarchitecture

- **One bus** connecting most registers allows many different RTs, but only **one at a time**
- Memory address must be copied into **MA** by CPU
- Memory data written from or read into **MD**
- First ALU operand always in **A**, result goes to **C**
- **Second ALU operand** always comes from bus
- Information only goes **into IR and MA** from bus (**unidirectional**)
  - A decoder (not shown) interprets contents of IR
  - MA supplies address to memory, not to CPU bus



## Abstract and Concrete RTN for SRC add Instruction

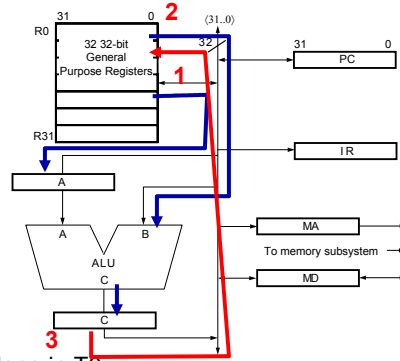
2/e **Abstract RTN:**  $(IR \leftarrow M[PC]; PC \leftarrow PC + 4; \text{instruction\_execution});$   
 $\text{instruction\_execution} := ( \dots$   
 $\text{add} (:= \text{op} = 12) \rightarrow R[\text{ra}] \leftarrow R[\text{rb}] + R[\text{rc}];$

Tbl 4.1 **Concrete RTN for add:**

Step	RTN
T0.	$MA \leftarrow PC; C \leftarrow PC + 4;$
T1.	$MD \leftarrow M[MA]; PC \leftarrow C;$
T2.	$IR \leftarrow MD;$
T3.	$A \leftarrow R[\text{rb}];$
T4.	$C \leftarrow A + R[\text{rc}];$
T5.	$R[\text{ra}] \leftarrow C;$

**6 clock cycles including fetch**

- Parts of 2 RTs ( $IR \leftarrow M[PC]; PC \leftarrow PC + 4;$ ) done in T0
- Single add RT takes 3 concrete RTs (T3, T4, T5)



## Concrete RTN for Arithmetic Instructions: addi

2/e

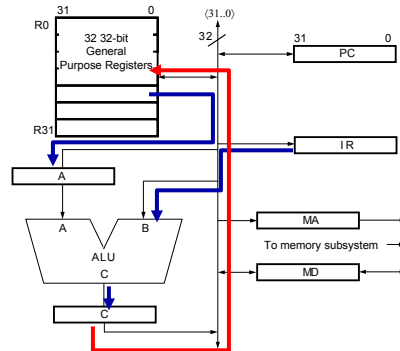
**Abstract RTN:**

$\text{addi} (:= \text{op} = 13) \rightarrow R[\text{ra}] \leftarrow R[\text{rb}] + c2(16..0) \{2's \text{ comp. sign extend}\};$

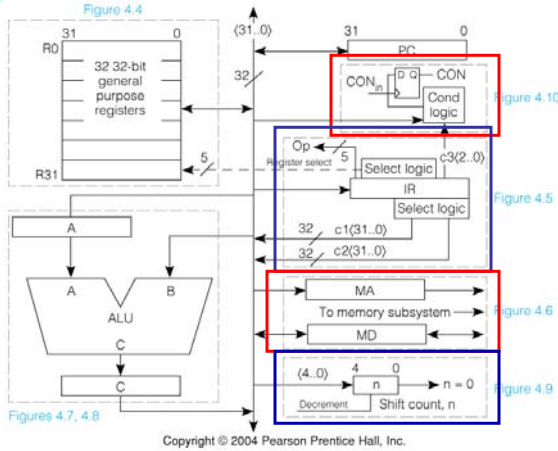
Tbl 4.2 **Concrete RTN for addi:**

Step	RTN
T0.	$MA \leftarrow PC; C \leftarrow PC + 4;$
T1.	$MD \leftarrow M[MA]; PC \leftarrow C;$
T2.	$IR \leftarrow MD;$
T3.	$A \leftarrow R[\text{rb}];$
T4.	$C \leftarrow A + c2(16..0) \{ \text{sign ext.} \};$
T5.	$R[\text{ra}] \leftarrow C;$

- Differs from add only in step T4
- Establishes **requirement for sign extend hardware**



## Adding Detail to the Data Path



• Concrete RTN lets us **add detail to the data path**

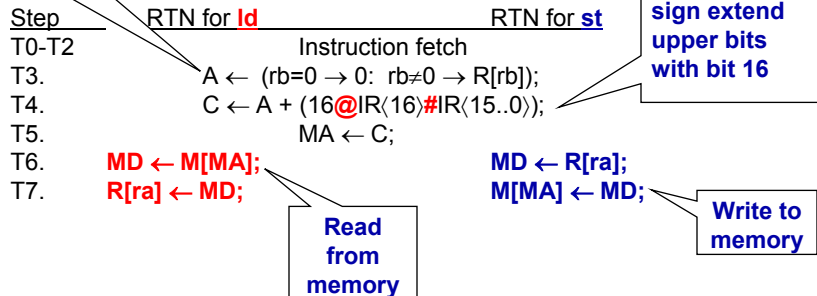
- Instruction register logic & new paths
- Condition bit flip-flop
- Shift count register

## Abstract and Concrete RTN for Load and Store

ld (:= op= 1) → R[ra] ← M[disp] :  
 st (:= op= 3) → M[disp] ← R[ra] :  
 where  
 disp<31..0> := ((rb=0) → c2<16..0> {sign ext.} :  
 (rb≠0) → R[rb] + c2<16..0> {sign extend, 2's comp.}) :

Need a way to use 0 if rb=0

Tbl 4.3



## Concrete RTN for Conditional Branch

br (:= op= 8) → (cond → PC ← R[rb]);  
 cond := ( c3<2..0>=0 → 0:  
           c3<2..0>=1 → 1:  
           c3<2..0>=2 → R[rc]=0:  
           c3<2..0>=3 → R[rc]≠0:  
           c3<2..0>=4 → R[rc]<31>=0:  
           c3<2..0>=5 → R[rc]<31>=1 );

Lower order bits of IR

never  
 always  
 if register is zero  
 if register is nonzero  
 if positive or zero  
 if negative

Tbl 4.4

Step	Concrete RTN
T0-T2	Instruction fetch
T3.	CON ← cond(R[rc]);
T4.	CON → PC ← R[rb];

One bit CON register  
 Not programmer-visible  
 Holds result of condition

If CON true, then set PC to address in rb

## Abstract and Concrete RTN for SRC Shift Right

shr (:= op = 26) → R[ra]<31..0> ← (n @ 0) # R[rb]<31..n> :  
 n := ( (c3<4..0>=0) → R[rc]<4..0> : shift count in reg.  
       (c3<4..0>≠0) → c3<4..0> );                    or const. field

Tbl 4.5

Step	Concrete RTN
T0-T2	Instruction fetch
T3.	n ← IR<4..0>;
T4.	(n=0) → (n ← R[rc]<4..0>);
T5.	C ← R[rb];
T6.	Shr (:= (n≠0) → (C<31..0> ← 0#C<31..1>: n ← n-1; Shr) );
T7.	R[ra] ← C;

n is a physical register

holds shift count

Recursion indicates need for control signals

step T6 is repeated n times



## Designing the Data Path

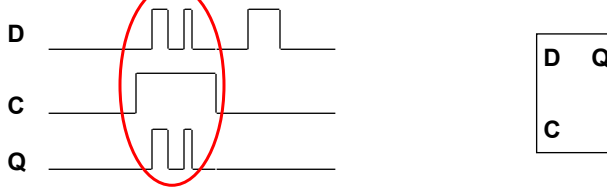
C  
S  
D  
A  
2/e

### Data Path/Control Unit Separation

- Interface between data path and control consists of **gate** and **strobe** signals
- A **gate selects** one of several values to apply to a common point, say a bus
- A **strobe changes the values** of the flip-flops in a register to match new inputs
- The **type of flip-flop** used in regs. has much influence on control and some on data path
  - **Latch**: simpler hardware, but more complex timing
  - **Edge triggering**: simpler timing, but about 2× hardware

## Latch and Edge-Triggered Operation

- Latch output follows input while strobe is **high**



- Edge triggering samples input at **edge time**

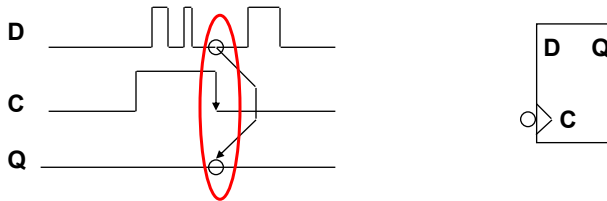
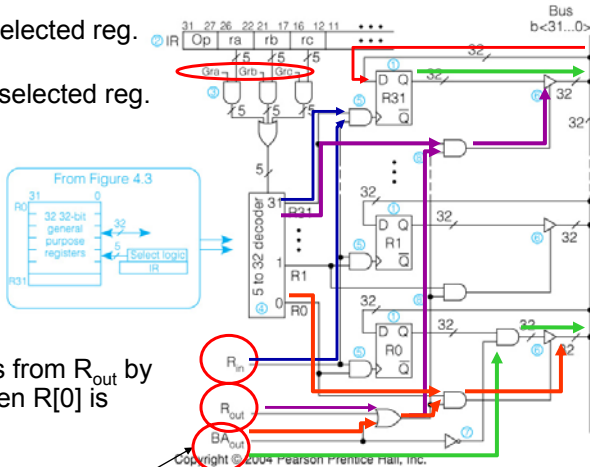


Fig. 4.4 The SRC Register File and Its Control Signals

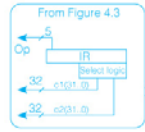
- $R_{out}$  gates selected reg. onto bus
- $R_{in}$  strobed selected reg. from bus
- $BA_{out}$  differs from  $R_{out}$  by gating 0 when  $R[0]$  is selected



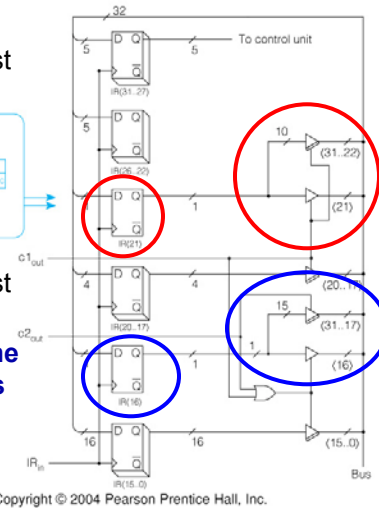
BA = Base Address

Fig. 4.5 Extracting c1, c2, and op from the Instruction Register

- $I\langle 21 \rangle$  is the sign bit of **C1** that must be extended



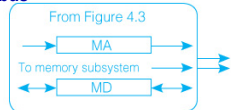
- $I\langle 16 \rangle$  is the sign bit of **C2** that must be extended
- Sign bits are **fanned out from one to several bits and gated to bus**



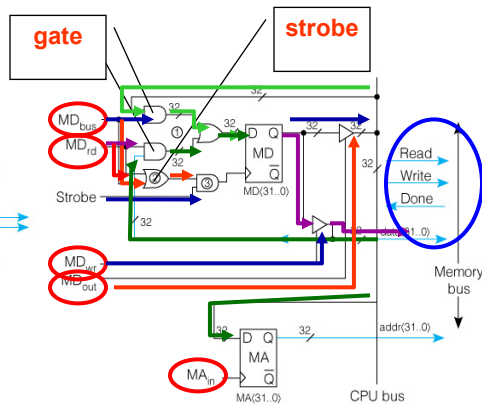
Copyright © 2004 Pearson Prentice Hall, Inc.

Fig. 4.6 CPU to Memory Interface: MA and MD Registers

- MD is loaded from memory bus ( $MD_{rd}$ ) or from CPU bus ( $MD_{bus}$ )



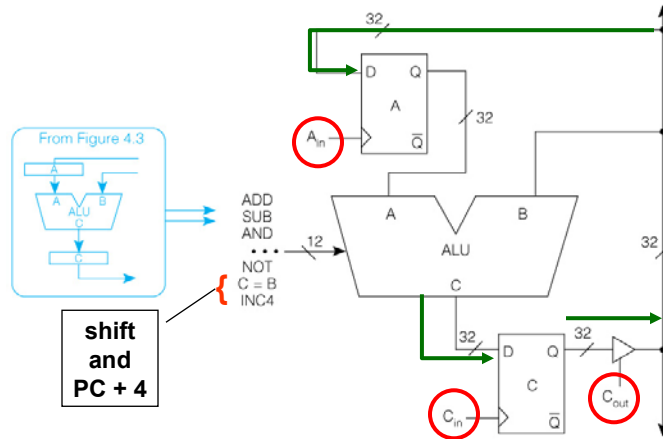
- MD can drive CPU bus or memory bus



Copyright © 2004 Pearson Prentice Hall, Inc.

C  
S  
D  
A  
2/e

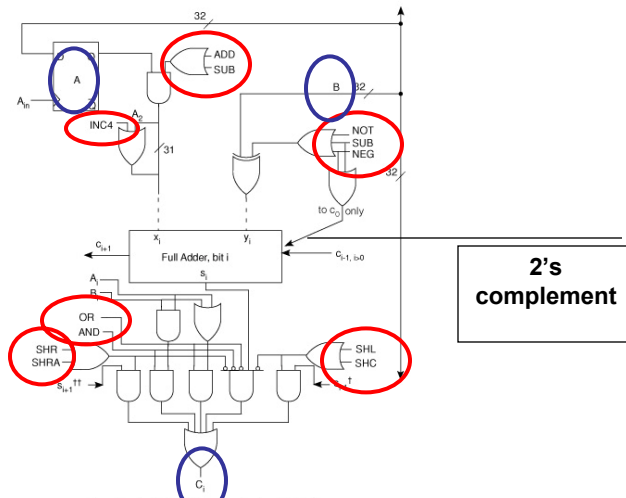
Fig. 4.7 The ALU and Its Associated Registers



Copyright © 2004 Pearson Prentice Hall, Inc.

C  
S  
D  
A  
2/e

Figure 4.8. A Logic-Level Design for One Bit of the 1-Bus SRC ALU



<sup>†</sup>( $s_{i-1}=0$ ) when SHL( $i=0$ ), ( $s_{i-1}=s_{i+1}$ ) when SHC( $i=0$ )  
<sup>††</sup>( $s_{i-1}=0$ ) when SHR( $i=31$ ), ( $s_{i-1}=s_{i+1}$ ) when SHRA( $i=31$ )

Copyright © 2004 Pearson Prentice Hall, Inc.

## Sample Problems

- Write concrete RTN for SRC **la** using 1-bus SRC

Abstract RTN: **la** ( $:= \text{op}=5$ )  $\rightarrow$  **R[ra]**  $\leftarrow$  **disp**;

T0-T2: Instruction fetch // for a test, you will need to understand how to write this

T3: **A**  $\leftarrow$  ( $(\text{rb}=0) \rightarrow 0 : (\text{rb} \neq 0) \rightarrow \text{R}[\text{rb}]$ );

T4: **C**  $\leftarrow$  **A** + **c1**{sign extend};

T5: **R[ra]**  $\leftarrow$  **C**;

## Sample Problems (cont'd)

- Write concrete RTN for SRC **str** using 1-bus SRC

Abstract RTN: **str** ( $:= \text{op}=4$ )  $\rightarrow$  **M[rel]**  $\leftarrow$  **R[ra]**;  
**rel** $\langle$ 31..0 $\rangle := \text{PC}\langle$ 31..0 $\rangle + \text{c1}\langle$ 21..0 $\rangle$ {sign-extend, 2's complement}

T0-T2: Instruction fetch // for a test, you will need to understand how to write this


T3: **A**  $\leftarrow$  **PC**

T4: **C**  $\leftarrow$  **A** + **c1**{sign extend};

T5: **MA**  $\leftarrow$  **C**;

T6: **MD**  $\leftarrow$  **R[ra]**;

T7: **M[MA]**  $\leftarrow$  **MD**;



# Control Sequences

C  
S  
D  
A  
2/e

## From Concrete RTN to Control Signals: The Control Sequence

---

Tbl 4.6—The Instruction Fetch

Step	Concrete RTN	Control Sequence
T0.	MA ← PC; C ← PC+4;	<b>PC<sub>out</sub>, MA<sub>in</sub>, Inc4, C<sub>in</sub></b>
T1.	MD ← M[MA]; PC ← C;	<b>Read, C<sub>out</sub>, PC<sub>in</sub>, Wait</b>
T2.	IR ← MD;	<b>MD<sub>out</sub>, IR<sub>in</sub></b>
T3.	Instruction_execution	

- The register transfers are the concrete RTN
- The **control signals that cause the register transfers** make up the control sequence
- Wait prevents the control from advancing to step T3 until the memory asserts Done

Computer Systems Design and Architecture Second Edition
© 2004 Prentice Hall



## Control Sequence for the SRC add Instruction

add (:= op= 12)  $\rightarrow$   $R[ra] \leftarrow R[rb] + R[rc]$ :

Tbl 4.7 The Add Instruction

Step	Concrete RTN	Control Sequence
T0.	$MA \leftarrow PC; C \leftarrow PC+4;$	$PC_{out}, MA_{in}, Inc4, C_{in}, Read$
T1.	$MD \leftarrow M[MA]; PC \leftarrow C;$	$C_{out}, PC_{in}, Wait$
T2.	$IR \leftarrow MD;$	$MD_{out}, IR_{in}$
T3.	$A \leftarrow R[rb];$	$Grb, R_{out}, A_{in}$
T4.	$C \leftarrow A + R[rc];$	$Grc, R_{out}, ADD, C_{in}$
T5.	$R[ra] \leftarrow C;$	$C_{out}, Gra, R_{in}, End$

- Note the use of Gra, Grb, & Grc to gate the correct 5 bit register select code to the regs.
- End signals the control to start over at step T0

## Control Sequence for the SRC addi Instruction

addi (:= op= 13)  $\rightarrow$   $R[ra] \leftarrow R[rb] + c2\langle 16..0 \rangle$  {2's comp., sign ext.} :

Tbl 4.8 The addi Instruction

Step	Concrete RTN	Control Sequence
T0.	$MA \leftarrow PC; C \leftarrow PC + 4;$	$PC_{out}, MA_{in}, Inc4, C_{in}, Read$
T1.	$MD \leftarrow M[MA]; PC \leftarrow C;$	$C_{out}, PC_{in}, Wait$
T2.	$IR \leftarrow MD;$	$MD_{out}, IR_{in}$
T3.	$A \leftarrow R[rb];$	$Grb, R_{out}, A_{in}$
T4.	$C \leftarrow A + c2\langle 16..0 \rangle$ {sign ext.};	$c2_{out}, ADD, C_{in}$
T5.	$R[ra] \leftarrow C;$	$C_{out}, Gra, R_{in}, End$

- The  $c2_{out}$  signal sign extends  $IR\langle 16..0 \rangle$  and gates it to the bus

C  
S  
D  
A  
2/e

## Control Sequence for the SRC st Instruction

st ( $:= op= 3$ )  $\rightarrow M[disp] \leftarrow R[ra]$  :  
 $disp\langle 31..0 \rangle := ((rb=0) \rightarrow c2\langle 16..0 \rangle \{sign\ ext.\})$  :  
 $(rb\neq 0) \rightarrow R[rb] + c2\langle 16..0 \rangle \{sign\ extend,\ 2's\ comp.\}$  ) :

### The st Instruction

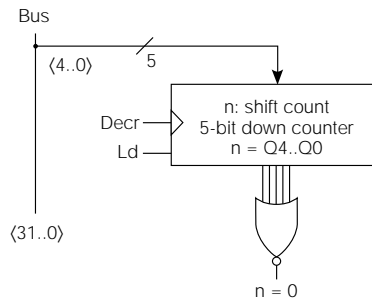
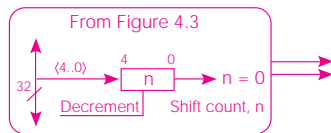
Step	Concrete RTN	Control Sequence
T0-T2	Instruction fetch	Instruction fetch
T3.	$A \leftarrow (rb=0) \rightarrow 0; rb\neq 0 \rightarrow R[rb];$	$\left. \begin{array}{l} \mathbf{Grb, BA_{out}, A_{in}} \\ \mathbf{c2_{out}, ADD, C_{in}} \end{array} \right\}$ address arithmetic
T4.	$C \leftarrow A + c2\langle 16..0 \rangle \{sign\ ext.\};$	
T5.	$MA \leftarrow C;$	$\mathbf{C_{out}, MA_{in}}$
T6.	$MD \leftarrow R[ra];$	$\mathbf{Gra, R_{out}, MD_{in}, Write}$
T7.	$M[MA] \leftarrow MD;$	$\mathbf{Wait, End}$

- Note  $BA_{out}$  in T3 compared to  $R_{out}$  in T3 of addi

C  
S  
D  
A  
2/e

## Fig. 4.9 The Shift Counter

- The concrete RTN for shr relies upon a 5 bit register to hold the shift count
- It must load, decrement, and have an  $n = 0$  test



## Tbl 4.10 Control Sequence for the SRC shr Instruction—Looping

Step	Concrete RTN	Control Sequence
T0-T2	Instruction fetch	Instruction fetch
T3.	$n \leftarrow IR\langle 4..0 \rangle;$	$c1_{out}, Ld$
T4.	$(n=0) \rightarrow (n \leftarrow R[rc]\langle 4..0 \rangle);$	$n=0 \rightarrow (Grc, R_{out}, Ld)$
T5.	$C \leftarrow R[rb];$	$Grb, R_{out}, C=B, C_{in}$
T6.	$Shr (:= (n \neq 0) \rightarrow (C\langle 31..0 \rangle \leftarrow 0\#C\langle 31..1 \rangle; n \leftarrow n-1; Shr));$	$n \neq 0 \rightarrow (C_{out}, SHR, C_{in}, Decr, Goto6)$
T7.	$R[ra] \leftarrow C;$	$C_{out}, Gra, R_{in}, End$

timing important

- Conditional control signals and repeating a control step are new concepts

## Branching

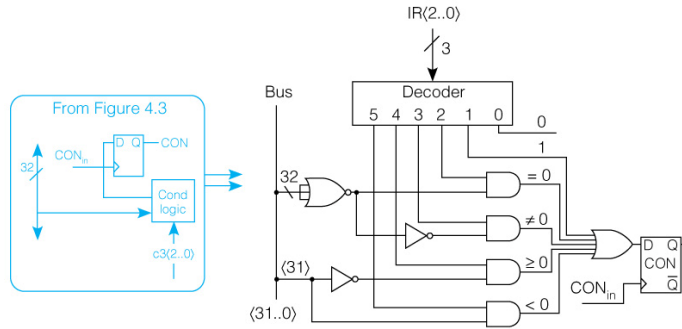
```

cond := ( c3<2..0>=0 → 0:
        c3<2..0>=1 → 1:
        c3<2..0>=2 → R[rc]=0:
        c3<2..0>=3 → R[rc]≠0:
        c3<2..0>=4 → R[rc]<31>=0:
        c3<2..0>=5 → R[rc]<31>=1 );
    
```

- This is equivalent to the logic expression

$$\begin{aligned}
 \text{cond} = & (c3\langle 2..0 \rangle = 1) \vee (c3\langle 2..0 \rangle = 2) \wedge (R[rc] = 0) \vee \\
 & (c3\langle 2..0 \rangle = 3) \wedge \neg (R[rc] = 0) \vee (c3\langle 2..0 \rangle = 4) \wedge \neg R[rc]\langle 31 \rangle \vee \\
 & (c3\langle 2..0 \rangle = 5) \wedge R[rc]\langle 31 \rangle
 \end{aligned}$$

Fig. 4.10 Computation of the Conditional Value CON



Copyright © 2004 Pearson Prentice Hall, Inc.

- NOR gate does =0 test of R[rc] on bus

Tbl 4.11 Control Sequence for SRC Branch Instruction, br

br (:= op= 8) → (cond → PC ← R[rb]):

Step	Concrete RTN	Control Sequence
T0-T2	Instruction fetch	Instruction fetch
T3.	CON ← cond(R[rc]);	<b>Gr</b> , R <sub>out</sub> , <b>CON<sub>in</sub></b>
T4.	CON → PC ← R[rb];	<b>Gr</b> , R <sub>out</sub> , <b>CON → PC<sub>in</sub></b> , <b>End</b>

- Condition logic is always connected to CON, so R[rc] only needs to be put on bus in T3
- Only PC<sub>in</sub> is conditional in T4 since gating R[rb] to bus makes no difference if it is not used

## Summary of the Design Process

**Informal** description  $\Rightarrow$  **formal** RTN description  $\Rightarrow$  **block diagram** arch.  $\Rightarrow$  **concrete** RTN steps  $\Rightarrow$  **hardware design** of blocks  $\Rightarrow$  **control sequences**  $\Rightarrow$  **control unit and timing**

- At each level, more decisions must be made
  - These decisions refine the design
  - Also place requirements on hardware still to be designed
- The nice one way process above has **circularity**
  - Decisions at later stages cause changes in earlier ones

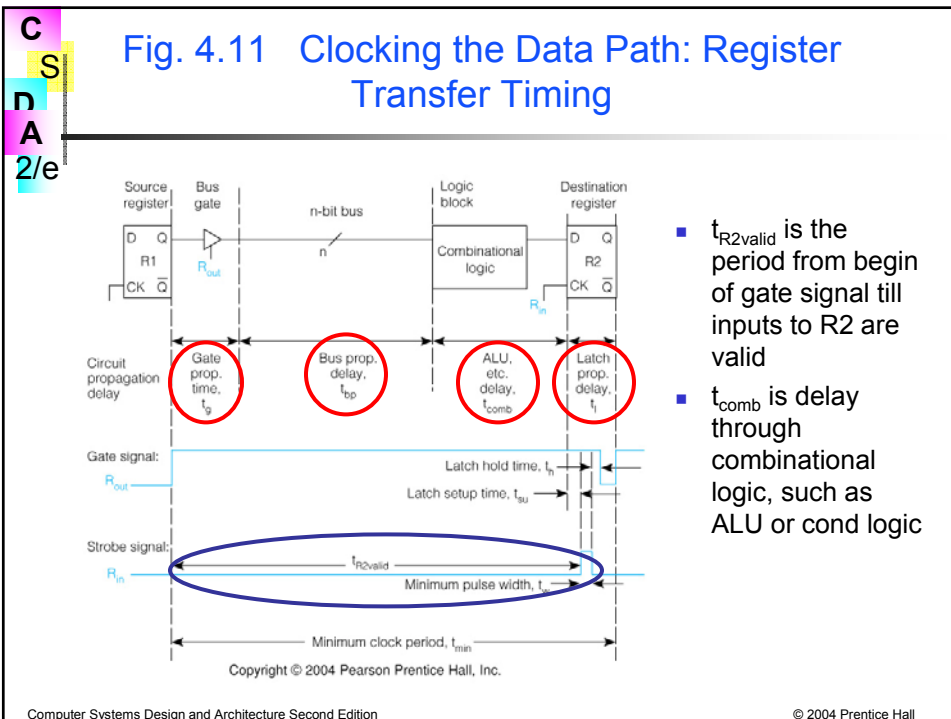
## Another Sample Problem

- Write concrete RTN steps and control sequences for the **not** instruction on a 1-bus SRC architecture

Step	RTN	Control Seq
T0 – T2	Inst. Fetch	Inst Ctrl Seq.
T3	$C \leftarrow \neg R[rc];$	<b>Grc, R<sub>out</sub>, NOT, C<sub>in</sub></b>
T4	$R[ra] \leftarrow C;$	<b>C<sub>out</sub>, Gra, R<sub>in</sub>, END</b>



# Register Transfer Timing and Designing the Control Unit



## Signal Timing on the Data Path

- Several delays occur in getting data from R1 to R2
- Gate delay** through the 3-state bus driver— $t_g$
- Worst case **propagation delay** on bus— $t_{bp}$
- Delay through any **logic**, such as ALU— $t_{comb}$
- Set up time** for data to affect state of R2— $t_{su}$
- Data can be strobed into R2 after this time
 
$$t_{R2valid} = t_g + t_{bp} + t_{comb} + t_{su}$$
- Diagram shows strobe signal in the form for a latch. It must be high for a minimum time— $t_w$
- There is a **hold time**,  $t_h$ , for data after strobe ends

## Effect of Signal Timing on Minimum Clock Cycle

- A total **latch propagation delay** is the sum
 
$$T_l = t_{su} + t_w + t_h$$
  - All above times are specified for latch
  - $t_h$  may be very small or zero
- The minimum clock period is determined by finding longest path from ff output to ff input
  - This is usually a path through the ALU
  - Conditional signals add a little gate delay
- Using this path, the minimum clock period is

$$t_{min} = t_g + t_{bp} + t_{comb} + t_l$$

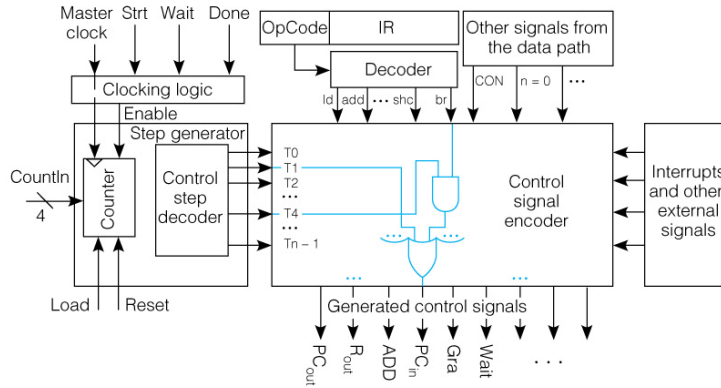
## Latches Versus Edge Triggered or Master Slave Flip-Flops

- During the high part of a strobe a latch changes its output
- If this output can affect its input, an error can occur
- This can influence even the kind of concrete RTs that can be written for a data path
- If the C register is implemented with latches, then  $C \leftarrow C + MD$ ; is not legal
- If the C register is implemented with master-slave or edge triggered flip-flops, it is OK

## The Control Unit

- The control unit's job is to generate the **control signals in the proper sequence**
- Things the control signals depend on
  - The **time step**  $T_i$
  - The instruction **op code** (for steps other than  $T_0, T_1, T_2$ )
  - Some few **data path signals** like CON,  $n=0$ , etc.
  - Some **external signals**: reset, interrupt, etc. (to be covered)
- The components of the control unit are: a **time state generator**, **instruction decoder**, and **combinational logic** to generate control signals

Fig. 4.12 Control Unit Detail with Inputs and Outputs



hard-wired

Copyright © 2004 Pearson Prentice Hall, Inc.

Synthesizing Control Signal Encoder Logic

Step	Control Sequence
T0.	PC <sub>out</sub> , MA <sub>in</sub> , Inc4, C <sub>in</sub> , Read
T1.	C <sub>out</sub> , PC <sub>in</sub> , Wait
T2.	MD <sub>out</sub> , IR <sub>in</sub>

add		addi		st		shr	
Step	Control Sequence	Step	Control Sequence	Step	Control Sequence	Step	Control Sequence
T3.	Grb, R <sub>out</sub> , A <sub>in</sub>	T3.	Grb, R <sub>out</sub> , A <sub>in</sub>	T3.	Grb, BA <sub>out</sub> , A <sub>in</sub>	T3.	C <sub>1out</sub> , Ld
T4.	GrC, R <sub>out</sub> , ADD, C <sub>in</sub>	T4.	c <sub>2out</sub> , ADD, C <sub>in</sub>	T4.	c <sub>2out</sub> , ADD, C <sub>in</sub>	T4.	n=0 → (GrC, R <sub>out</sub> , Ld)
T5.	C <sub>out</sub> , Gra, R <sub>in</sub> , End	T5.	C <sub>out</sub> , Gra, R <sub>in</sub> , End	T5.	C <sub>out</sub> , MA <sub>in</sub>	T5.	Grb, R <sub>out</sub> , C=B
				T6.	Gra, R <sub>out</sub> , MD <sub>in</sub> , Write	T6.	n≠0 → (C <sub>out</sub> , SHR, C <sub>in</sub> , Decr, Goto7)
				T7.	Wait, End	T7.	C <sub>out</sub> , Gra, R <sub>in</sub> , End

Design process:

- Comb through the entire set of control sequences.
- Find all occurrences of each control signal.
- Write an equation describing that signal.

Example:  $Gra = T5 \cdot (add + addi) + T6 \cdot st + T7 \cdot shr + \dots$

## Use of Data Path Conditions in Control Signal Logic

Step	Control Sequence
T0.	PC <sub>out</sub> : MA <sub>in</sub> , Inc4, C <sub>in</sub> , Read
T1.	C <sub>out</sub> : PC <sub>in</sub> , Wait
T2.	MD <sub>out</sub> : IR <sub>in</sub>

add		addi		st		shr	
Step	Control Sequence	Step	Control Sequence	Step	Control Sequence	Step	Control Sequence
T3.	Grb, R <sub>out</sub> : A <sub>in</sub>	T3.	Grb, R <sub>out</sub> : A <sub>in</sub>	T3.	Grb, BA <sub>out</sub> : A <sub>in</sub>	T3.	c <sub>1out</sub> : Ld
T4.	Grc, R <sub>out</sub> : ADD, C <sub>in</sub>	T4.	c <sub>2out</sub> : ADD, C <sub>in</sub>	T4.	c <sub>2out</sub> : ADD, C <sub>in</sub>	T4.	n=0 → (Grc, R <sub>out</sub> : Ld) ● ● ●
T5.	C <sub>out</sub> : Gra, R <sub>in</sub> : End	T5.	C <sub>out</sub> : Gra, R <sub>in</sub> : End	T5.	C <sub>out</sub> : MA <sub>in</sub>	T5.	Grb, R <sub>out</sub> : C=B
				T6.	Gra, R <sub>out</sub> : MD <sub>in</sub> : Write	T6.	n=0 → (C <sub>out</sub> : SHR, C <sub>in</sub> : Decr, Goto7)
				T7.	Wait, End	T7.	C <sub>out</sub> : Gra, R <sub>in</sub> : End

Example:  $Grc = T4 \cdot add + T4 \cdot (n=0) \cdot shr + \dots$

Fig. 4.13 Generation of the logic for C<sub>out</sub> and G<sub>ra</sub>

