# Chapter 4 Topics

- The Design Process
- A 1-bus Microarchitecture for SRC
- Data Path Implementation
- Logic Design for the 1-bus SRC
- The Control Unit
- The 2- and 3-bus Processor Designs
- The Machine Reset Process
- Machine Exceptions

# 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
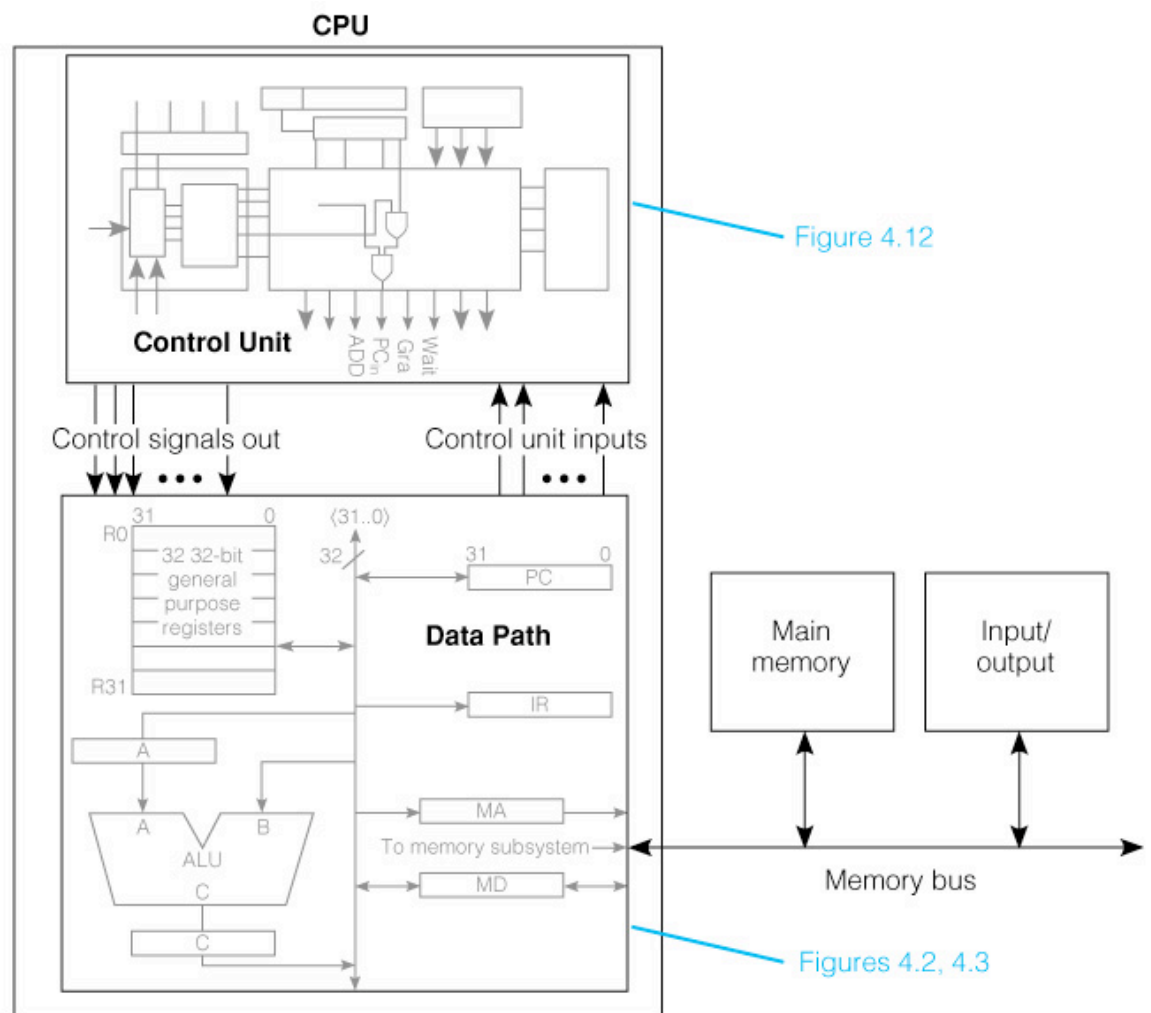
- Several concrete RTNs could implement the same ISA

# A Note on the Design Process

- In this chapter presents several SRC designs
- We started in Chap. 2 with an informal description
- In this chapter we will propose several block diagram architectures to support the abstract RTN, then we will:
  - 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
- Design a control unit to generate control signals

# Fig. 4.1 Block Diagram of 1-bus SRC



Figure 4.12

CPU

Control Unit

ADD PCin Gra Wait

Control signals out          Control unit inputs

Data Path

R0
32 32-bit
general
purpose
registers

R31

A

A        B

ALU

C

C

PC

IR

MA

To memory subsystem

MD

Main memory

Input/output

Memory bus

Figures 4.2, 4.3

Copyright © 2004 Pearson Prentice Hall, Inc.

# Fig. 4.2 High-Level View of the 1-Bus SRC Design



ADD
SUB
AND
OR
SHR
SHRA
SHL
SHC
NOT
NEG
C=B
INC4

Copyright © 2004 Pearson Prentice Hall, Inc.
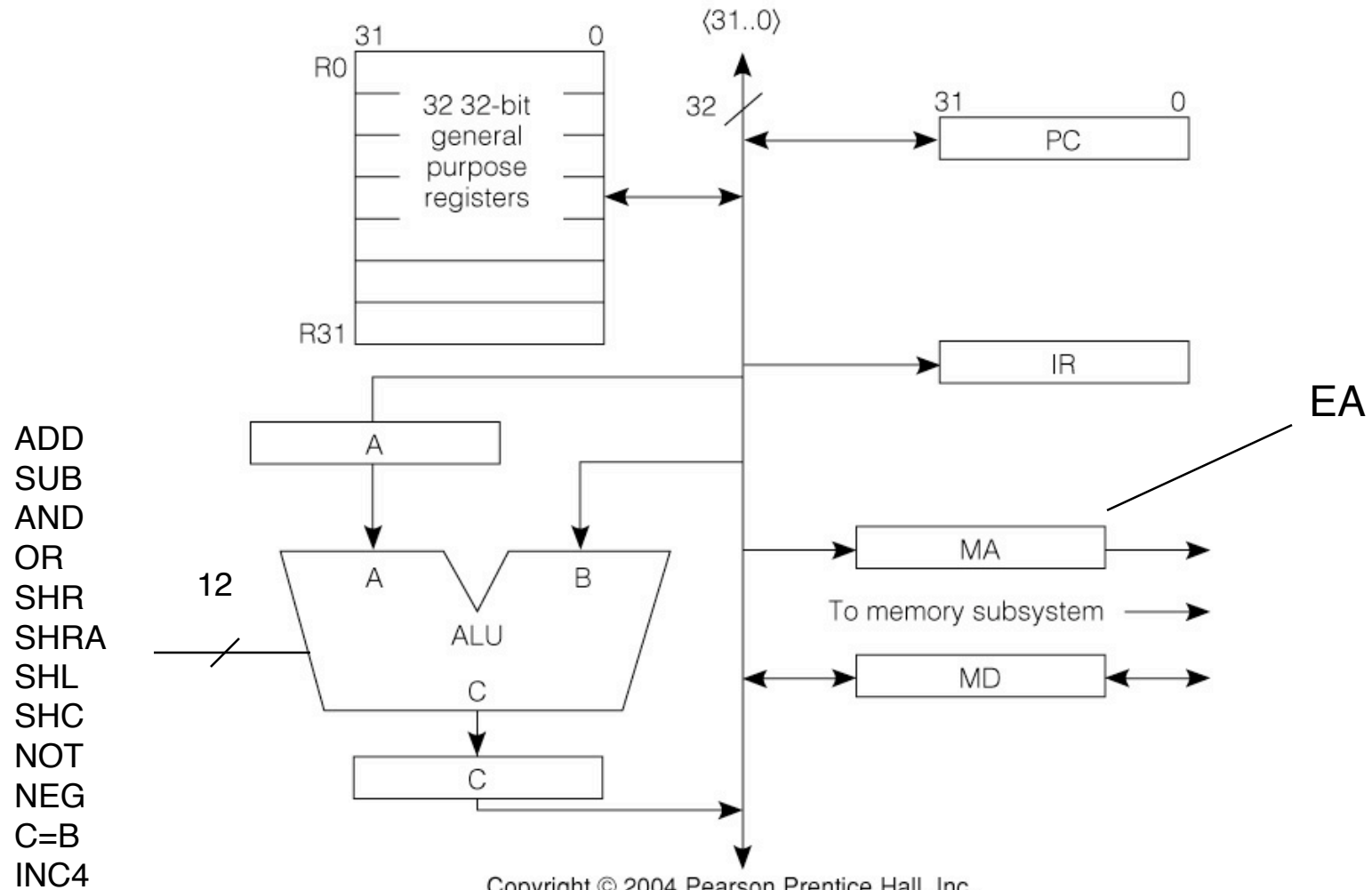
# Constraints Imposed by the Microarchitecture

- 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
    - A decoder (not shown) interprets contents of IR
    - MA supplies address to memory, not to CPU bus
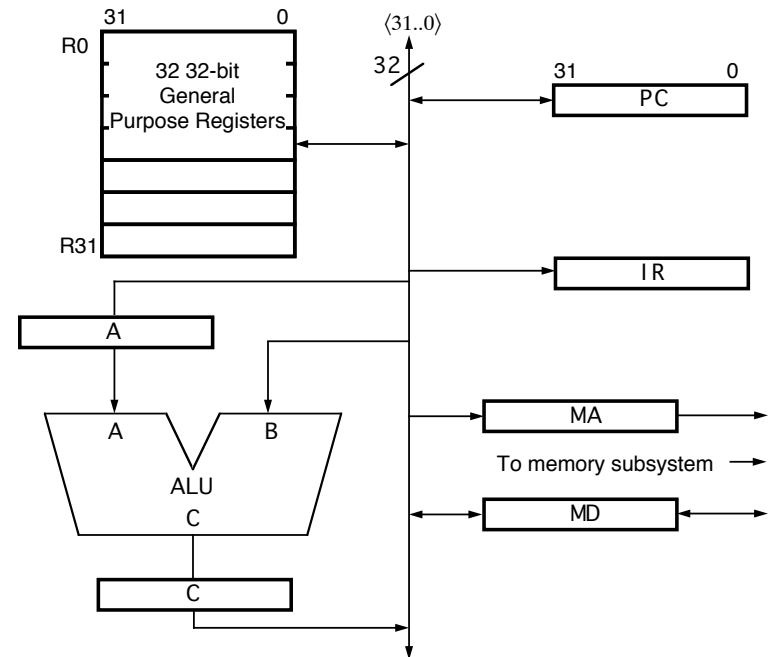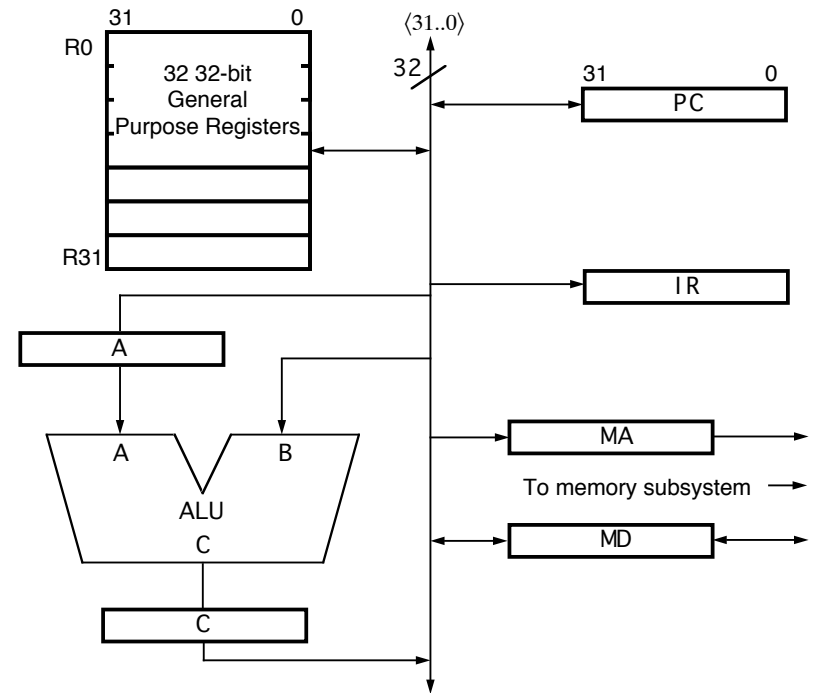
# Abstract and Concrete RTN for SRC add Instruction

**Abstract RTN:**  (IR ← M[PC]: PC ← PC + 4; instruction_execution);
instruction_execution := ( • • •
add (:= op= 12) → R[ra] ← R[rb] + R[rc]:

Tbl 4.1 Concrete RTN for add:

| Step | RTN |
|------|-----|
| T0. | MA ← PC:  C ← PC + 4; |
| T1. | MD ← M[MA]:  PC ← C; |
| T2. | IR ← MD; |
| T3. | A ← R[rb]; |
| T4. | C ← A + R[rc]; |
| T5. | R[ra] ← C; |

IF

IEx.



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

# Concrete RTN Gives Information about Sub-units

- The ALU must be able to add two 32-bit values

- ALU must also be able to increment B input by 4

- Memory read must use address from MA and return data to MD

- Two RTs separated by : in the concrete RTN, as in T0 and T1, are operations at the same clock

- Steps T0, T1, and T2 constitute instruction fetch, and will be the same for all instructions

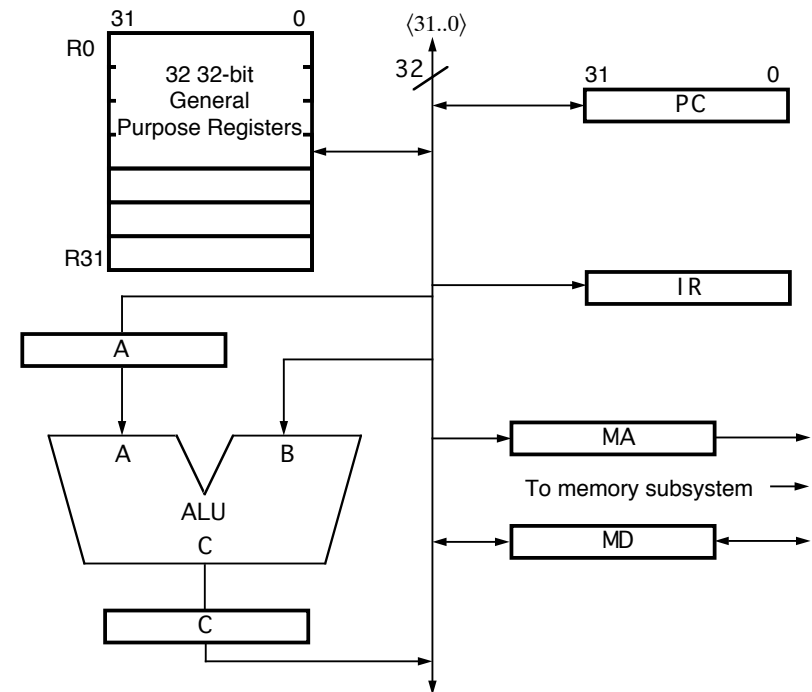- With this implementation, fetch and execute of the add instruction takes 6 clock cycles

Abstract RTN:

addi (:= op= 13) → R[ra] ← R[rb] + c2⟨16..0⟩ {2's comp.  sign extend} :

Tbl 4.2 Concrete RTN for addi:

| Step | RTN |
|------|-----|
| T0. | MA ← PC:  C ← PC + 4; |
| T1. | MD ← M[MA];  PC ← C; |
| T2. | IR ← MD; |
| T3. | A ← R[rb]; |
| T4. | C ← A +  c2⟨16..0⟩ {sign ext.}; |
| T5. | R[ra] ← C; |



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

# Fig. 4.3   More Complete view of Registers and Buses in 1-bus SRC Design—Including Some Control Signals



Figure 4.4

Copyright © 2004 Pearson Prentice Hall, Inc.

- **Concrete RTN lets us add detail to the data path**
  - Instruction register logic & new paths
  - Condition bit flip-flop
  - Shift count register

Keep this slide in mind as we discuss concrete RTN of instructions.

ld (:= op= 1) → R[ra] ← M[disp] :
st (:= op= 3) → M[disp] ← R[ra] :

where

disp$\langle 31..0 \rangle$ := ((rb=0) → c2$\langle 16..0 \rangle$ {sign ext.} :

(rb≠0) → R[rb] + c2$\langle 16..0 \rangle$ {sign extend, 2's comp.} ) :

Tbl 4.3

| Step | RTN for ld | RTN for st |
|------|------------|------------|
| T0-T2 | Instruction fetch | |
| T3. | A ← (rb=0 → 0: rb≠0 → R[rb]); | |
| T4. | C ← A + (16@IR$\langle 16 \rangle$#IR$\langle 15..0 \rangle$); | |
| T5. | MA ← C; | |
| T6. | MD ← M[MA]; | MD ← R[ra]; |
| T7. | R[ra] ← MD; | M[MA] ← MD; |

# Notes for Load and Store RTN

- Steps T0 through T2 are the same as for add and addi, and for <u>all instructions</u>

- In addition, steps T3 through T5 are the same for ld and st, because they calculate disp
- A way is needed to use 0 for R[rb] when rb=0
- 15 bit sign extension is needed for IR$\langle 16..0 \rangle$

- Memory read into MD occurs at T6 of ld
- Write of MD into memory occurs at T7 of st

# Concrete RTN for Conditional Branch

br (:= op= 8) → (cond → PC ← R[rb]):
cond := ( c3⟨2..0⟩=0 → 0:                                      never
        c3⟨2..0⟩=1 → 1:                                  always
        c3⟨2..0⟩=2 → R[rc]=0:                if register is zero
        c3⟨2..0⟩=3 → R[rc]≠0:                if register is nonzero
        c3⟨2..0⟩=4 → R[rc]⟨31⟩=0:    if positive or zero
        c3⟨2..0⟩=5 → R[rc]⟨31⟩=1 ):    if negative

Tbl 4.4

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

# Notes on Conditional Branch RTN

- c3⟨2..0⟩ are just the low order 3 bits of IR

- cond() is evaluated by a combinational logic circuit having inputs from R[rc] and c3⟨2..0⟩

- The one bit register CON is not accessible to the programmer and only holds the output of the combinational logic for the condition

- If the branch succeeds, the program counter is replaced by the contents of a general reg.

# Abstract and Concrete RTN for SRC Shift Right

$shr \, (:= op = 26) \rightarrow R[ra]\langle 31..0\rangle \leftarrow (n @ 0) \# R[rb]\langle 31..n\rangle :$

$n := ( \quad (c3\langle 4..0\rangle=0) \rightarrow R[rc]\langle 4..0\rangle :$ shift count in reg.

$\qquad (c3\langle 4..0\rangle \neq 0) \rightarrow c3\langle 4..0\rangle ):$ or const. field

Tbl 4.5

| Step | Concrete RTN |
|------|--------------|
| T0-T2 | Instruction fetch |
| T3. | $n \leftarrow IR\langle 4..0\rangle;$ |
| T4. | $(n=0) \rightarrow (n \leftarrow R[rc]\langle 4..0\rangle);$ |
| T5. | $C \leftarrow R[rb];$ |
| T6. | $Shr \, (:= (n \neq 0) \rightarrow (C\langle 31..0\rangle \leftarrow 0\#C\langle 31..1\rangle: n \leftarrow n-1; Shr) );$ |
| T7. | $R[ra] \leftarrow C;$ |

step T6 is repeated n times

# Notes on SRC Shift RTN

- In the abstract RTN, n is defined with :=

- In the concrete RTN, it is a physical register

- n not only holds the shift count but is used as a counter in step T6

- Step T6 is repeated n times as shown by the recursion in the RTN

- The control for such repeated steps will be treated later

# Data Path/Control Unit Separation

- Interface between data path and control consists of <u>gate</u> and <u>strobe</u> 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

# Reminder on Latch and Edge-Triggered Operation

- Latch output follows input while strobe is high
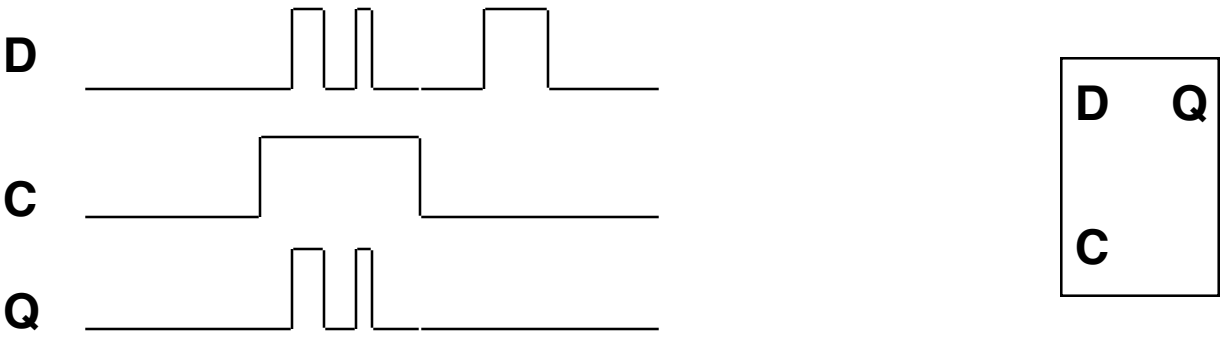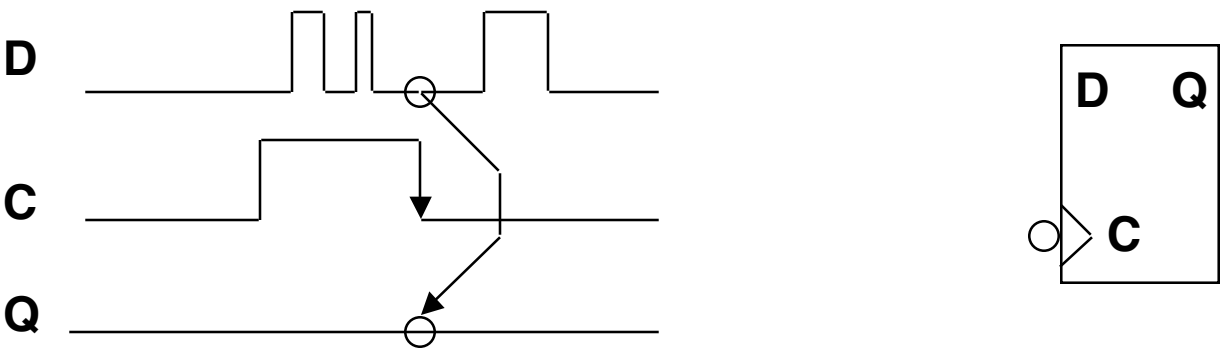


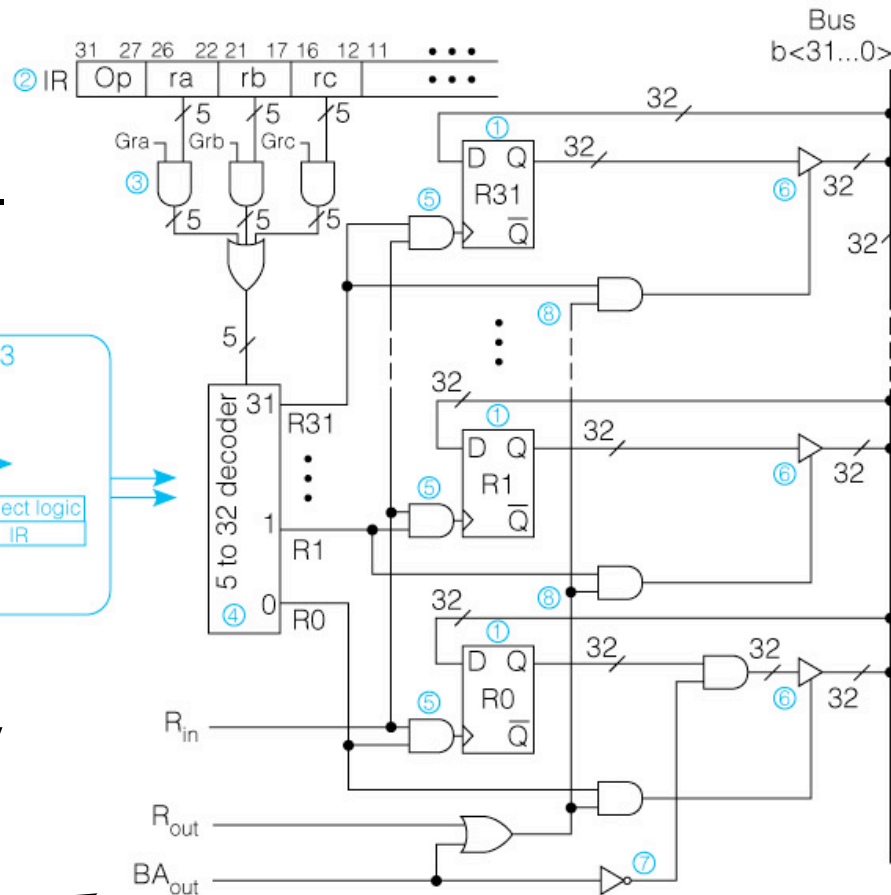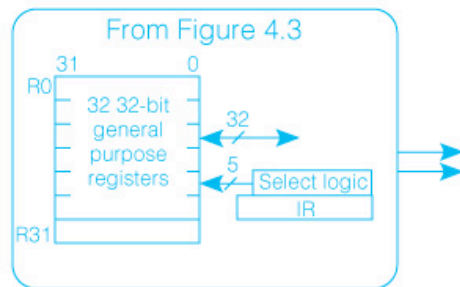- **Edge triggering samples input at edge time**

# Fig. 4.4  The SRC Register File and Its Control Signals

- $R_{out}$ gates selected reg. onto bus
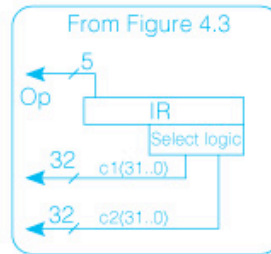
- $R_{in}$ strobed selected reg. from bus



- $BA_{out}$ 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
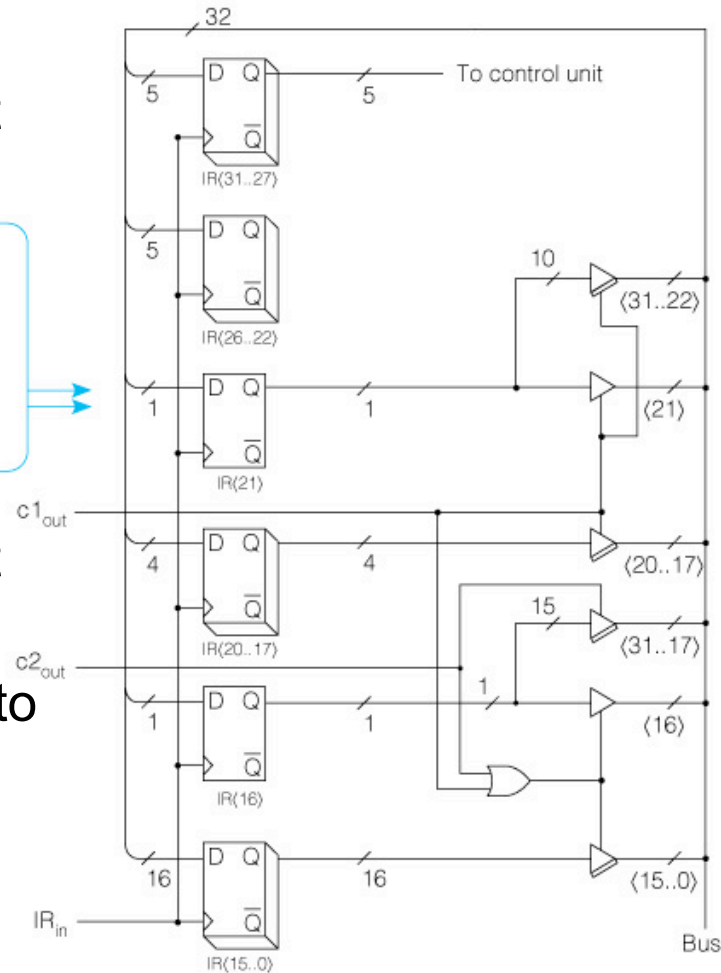
Copyright © 2004 Pearson Prentice Hall, Inc.

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

- MD is loaded from memory bus or from CPU bus



From Figure 4.3
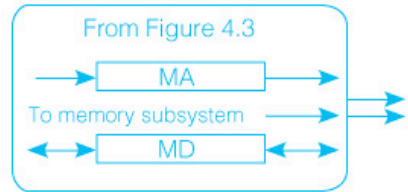
- MD can drive CPU bus or memory bus

Copyright © 2004 Pearson Prentice Hall, Inc.

# Fig. 4.7 The ALU and Its Associated Registers
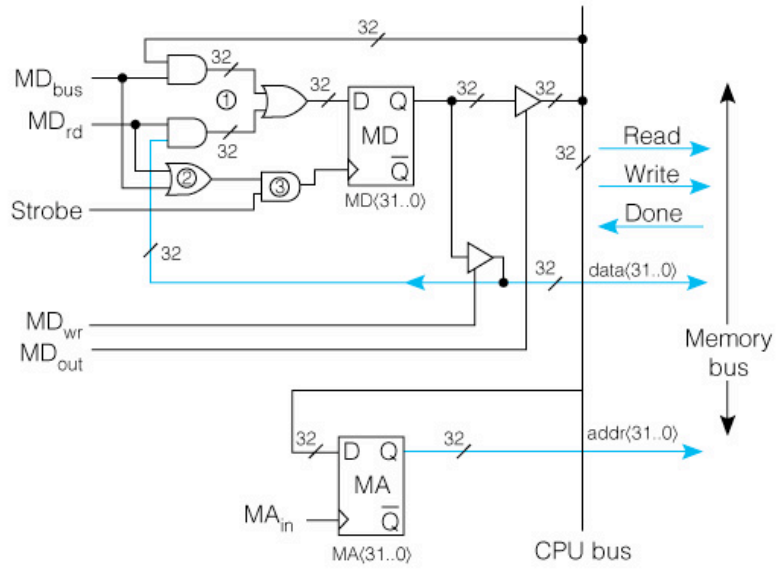


Copyright © 2004 Pearson Prentice Hall, Inc.

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



Copyright © 2004 Pearson Prentice Hall, Inc.

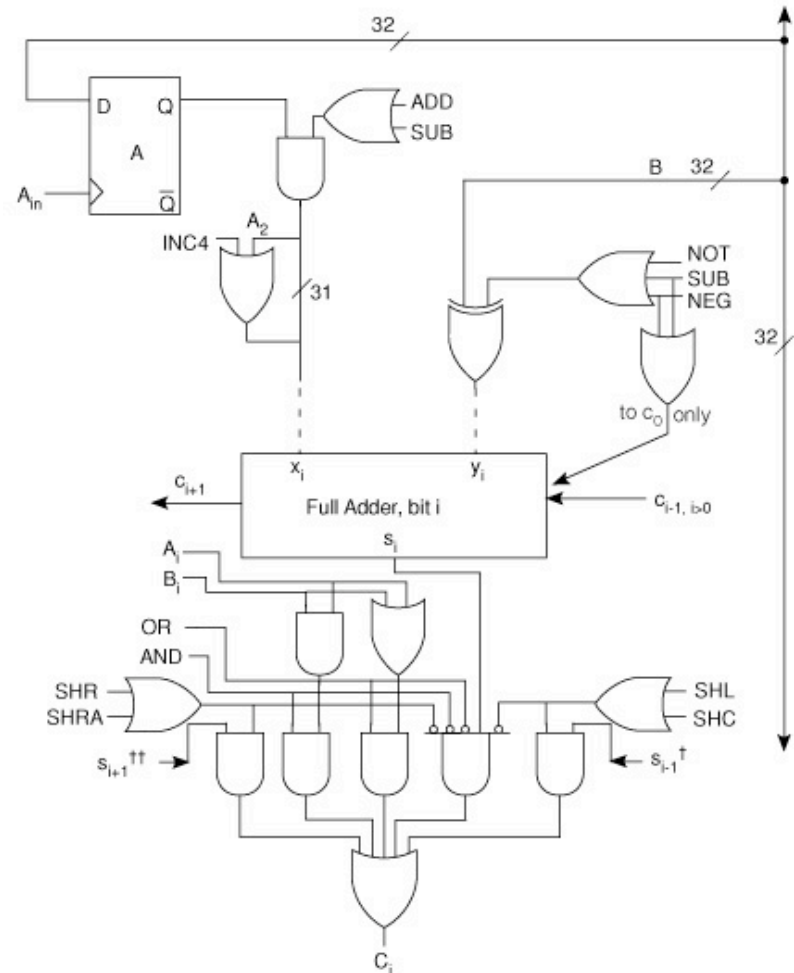# From Concrete RTN to Control Signals: The Control Sequence

Tbl 4.6—The Instruction Fetch

| Step | Concrete RTN | Control Sequence |
|------|-------------|------------------|
| T0. | MA ← PC: C ← PC+4; | $PC_{out}$, $MA_{in}$, Inc4, $C_{in}$ |
| T1. | MD ← M[MA]: PC ← C; | Read, $C_{out}$, $PC_{in}$, Wait |
| T2. | IR ← MD; | $MD_{out}$, $IR_{in}$ |
| 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

# Control Steps, Control Signals, and Timing

- Within a given time step, the order in which control signals are written is irrelevant

  - In step T0, $C_{in}$, Inc4, $MA_{in}$, $PC_{out}$ == $PC_{out}$, $MA_{in}$, Inc4, $C_{in}$

- The only timing distinction within a step is between gates and strobes

- The memory read should be started as early as possible to reduce the wait

- MA must have the right value before being used for the read

- Depending on memory timing, Read could be in T0

# Control Sequence for the SRC add Instruction

add (:= op= 12) → R[ra] ← R[rb] + R[rc]:

Tbl 4.7 The Add Instruction

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

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

# Control Sequence for the SRC st Instruction

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

<u>The st Instruction</u>

| Step | Concrete RTN | Control Sequence | |
|------|-------------|------------------|---|
| T0-T2 | Instruction fetch | Instruction fetch | |
| T3. | A ← (rb=0) → 0: rb≠0 → R[rb]; | Grb, BA$_{out}$, A$_{in}$ | $\Big\}$ address arithmetic |
| T4. | C ← A + c2⟨16..0⟩ {sign ext.}; | c2$_{out}$, ADD, C$_{in}$ | |
| T5. | MA ← C; | C$_{out}$, MA$_{in}$ | |
| T6. | MD ← R[ra]; | Gra, R$_{out}$, MD$_{in}$, Write | |
| T7. | M[MA] ← MD; | Wait, End | |

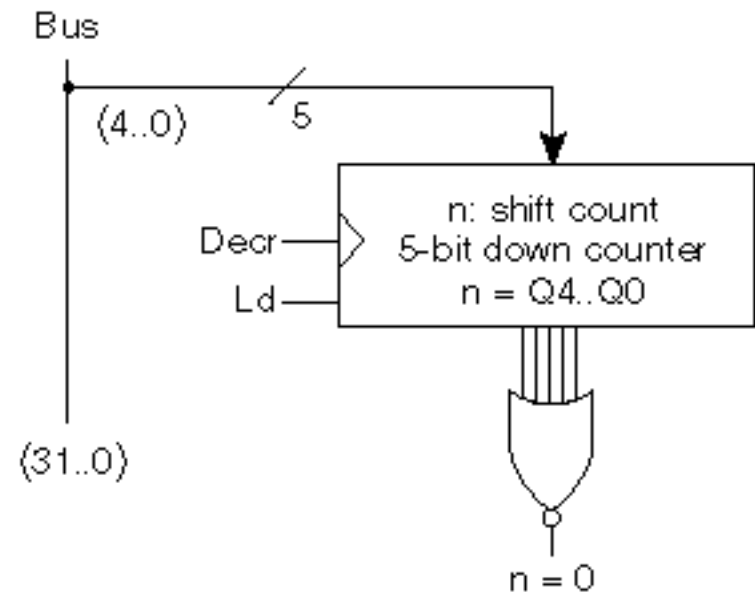- Note BA$_{out}$ in T3 compared to R$_{out}$ in T3 of addi

# 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  = 0 test



Computer Systems Design and Architecture Second Edition                    © 2004 Prentice Hall

| 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 (:= $(n\neq 0) \rightarrow$ <br> $(C\langle 31..0\rangle \leftarrow 0\#C\langle 31..1\rangle$: <br> $n \leftarrow n-1$; Shr) ); | $n\neq 0 \rightarrow (C_{out}, SHR, C_{in},$ <br> Decr, Goto6) |
| T7. | $R[ra] \leftarrow C$; | $C_{out}, Gra, R_{in}, End$ |

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

$$\text{cond} := (\ c3\langle 2..0\rangle = 0 \rightarrow 0:$$
$$c3\langle 2..0\rangle = 1 \rightarrow 1:$$
$$c3\langle 2..0\rangle = 2 \rightarrow R[rc] = 0:$$
$$c3\langle 2..0\rangle = 3 \rightarrow R[rc] \neq 0:$$
$$c3\langle 2..0\rangle = 4 \rightarrow R[rc]\langle 31\rangle = 0:$$
$$c3\langle 2..0\rangle = 5 \rightarrow R[rc]\langle 31\rangle = 1\ ):$$

- This is equivalent to the logic expression

$$\text{cond} = (c3\langle 2..0\rangle = 1) \lor (c3\langle 2..0\rangle = 2) \land (R[rc] = 0) \lor$$
$$(c3\langle 2..0\rangle = 3) \land \neg (R[rc] = 0) \lor (c3\langle 2..0\rangle = 4) \land \neg R[rc]\langle 31\rangle \lor$$
$$(c3\langle 2..0\rangle = 5) \land R[rc]\langle 31\rangle$$

# Fig. 4.10   Computation of the Conditional Value CON
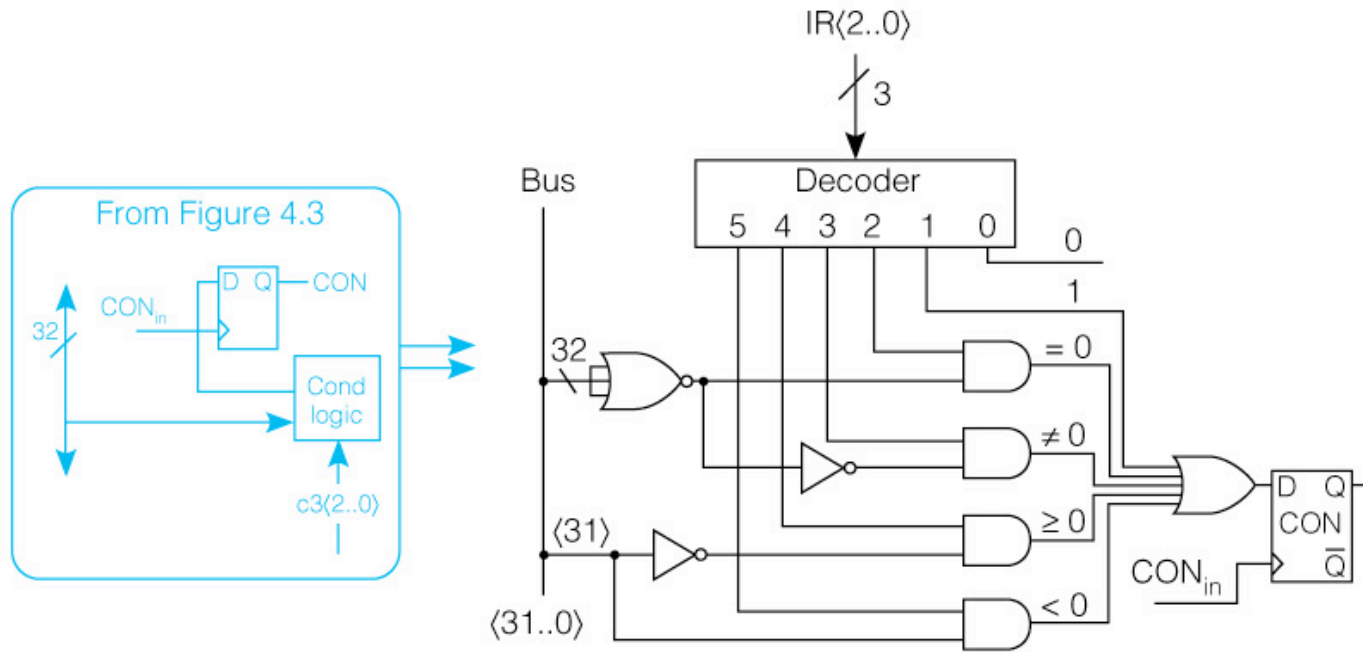
Copyright © 2004 Pearson Prentice Hall, Inc.

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

# Tbl 4.11 Control Sequence for SRC Branch Instruction, br

$$br \; (:= op= 8) \rightarrow (cond \rightarrow PC \leftarrow R[rb]):$$

| Step | Concrete RTN | Control Sequence |
|------|--------------|------------------|
| T0-T2 | Instruction fetch | Instruction fetch |
| T3. | $CON \leftarrow cond(R[rc])$; | Grc, $R_{out}$, $CON_{in}$ |
| T4. | $CON \rightarrow PC \leftarrow R[rb]$; | Grb, $R_{out}$, $CON \rightarrow PC_{in}$, End |

- Condition logic is always connected to CON, so R[rc] only needs to be put on bus in T3
- Only $PC_{in}$ 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
  - Happens less in a text than in reality because
    - Can be fixed on re-reading
    - Confusing to first time student

# Fig. 4.11 Clocking the Data Path: Register Transfer Timing
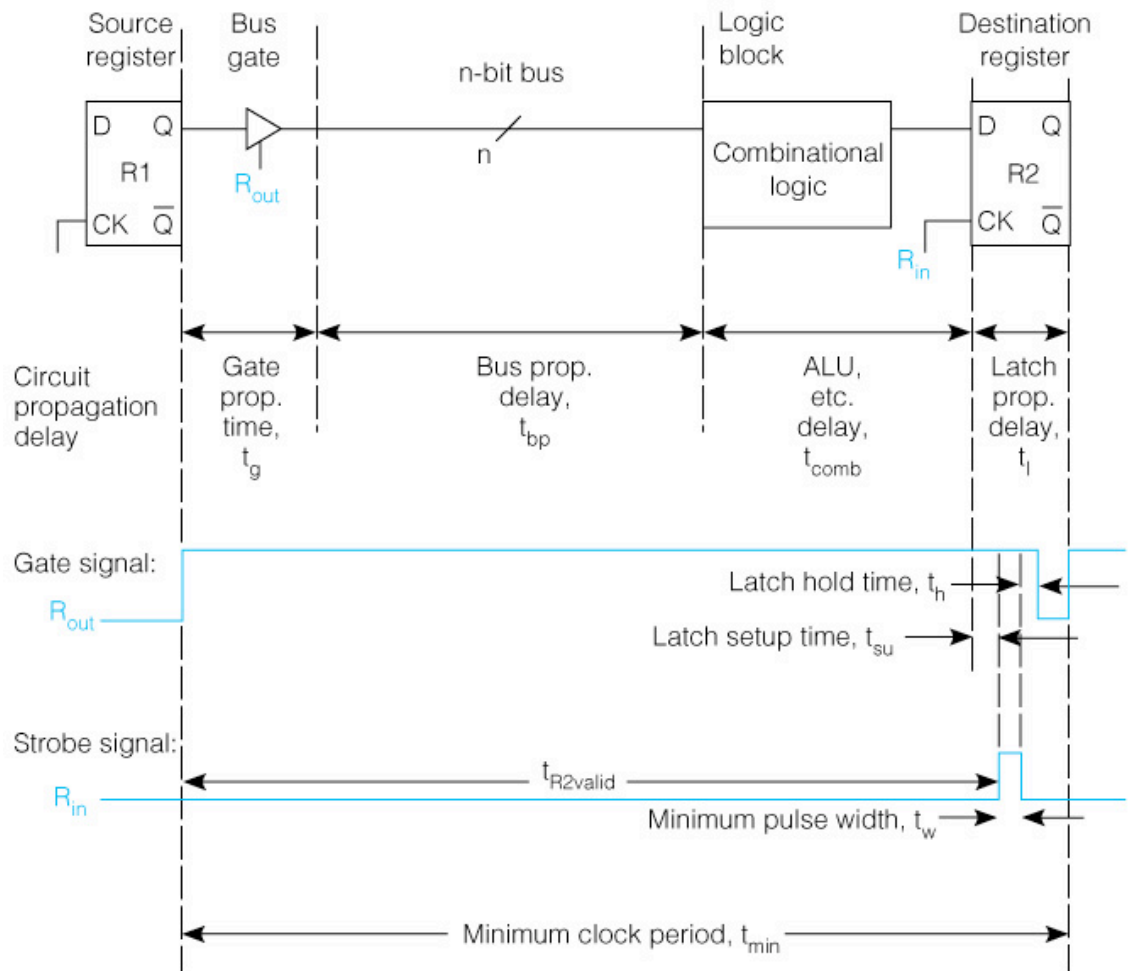


- $t_{R2valid}$ is the period from begin of gate signal till inputs to R2 are valid
- $t_{comb}$ is delay through combinational logic, such as ALU or cond logic

Copyright © 2004 Pearson Prentice Hall, Inc.

# 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 Ti
  - The instruction op code (for steps other than T0, T1, T2)
  - 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
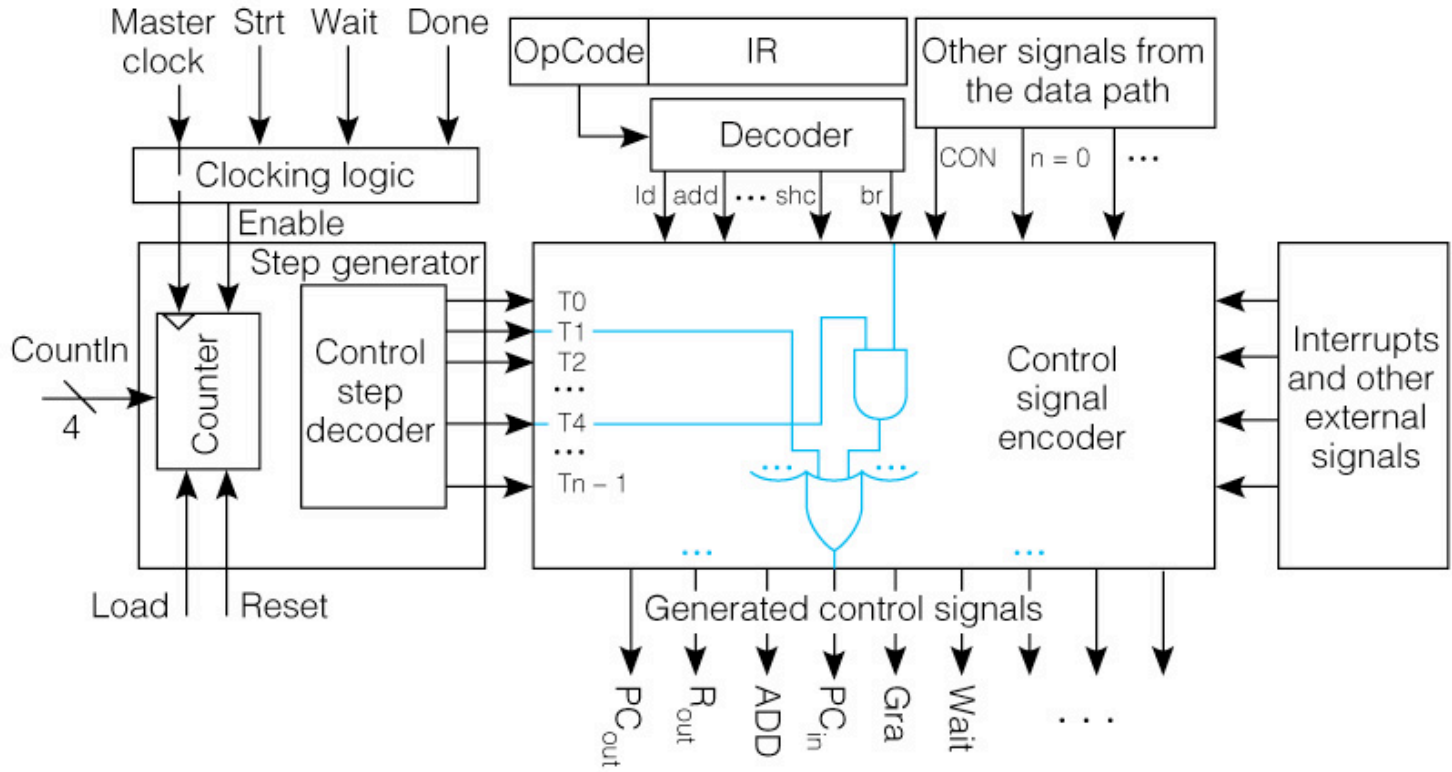


Copyright © 2004 Pearson Prentice Hall, Inc.

# Synthesizing Control Signal Encoder Logic

| Step | Control Sequence |
|------|------------------|
| T0. | $PC_{out}$, $MA_{in}$, Inc4, $C_{in}$, Read |
| T1. | $C_{out}$, $PC_{in}$, Wait |
| T2. | $MD_{out}$, $IR_{in}$ |

**add**

| Step | Control Sequence |
|------|------------------|
| T3. | Grb, $R_{out}$, $A_{in}$ |
| T4. | Grc, $R_{out}$, ADD, $C_{in}$ |
| T5. | $C_{out}$, **Gra,** $R_{in}$, End |

**addi**

| Step | Control Sequence |
|------|------------------|
| T3. | Grb, $R_{out}$, $A_{in}$ |
| T4. | $c2_{out}$, ADD, $C_{in}$ |
| T5. | $C_{out}$, **Gra,** $R_{in}$, End |

**st**

| Step | Control Sequence |
|------|------------------|
| T3. | Grb, $BA_{out}$, $A_{in}$ |
| T4. | $c2_{out}$, ADD, $C_{in}$ |
| T5. | $C_{out}$, $MA_{in}$ |
| T6. | **Gra,** $R_{out}$, $MD_{in}$, Write |
| T7. | Wait, End |

**shr**

| Step | Control Sequence |
|------|------------------|
| T3. | $c1_{out}$, Ld |
| T4. | n=0 → (Grc, $R_{out}$, Ld) |
| T5. | Grb, $R_{out}$, C=B |
| T6. | n≠0 → ($C_{out}$, SHR, $C_{in}$, Decr, Goto7) |
| T7. | $C_{out}$, **Gra,** $R_{in}$, 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·(add + addi) + T6·st + T7·shr + ...
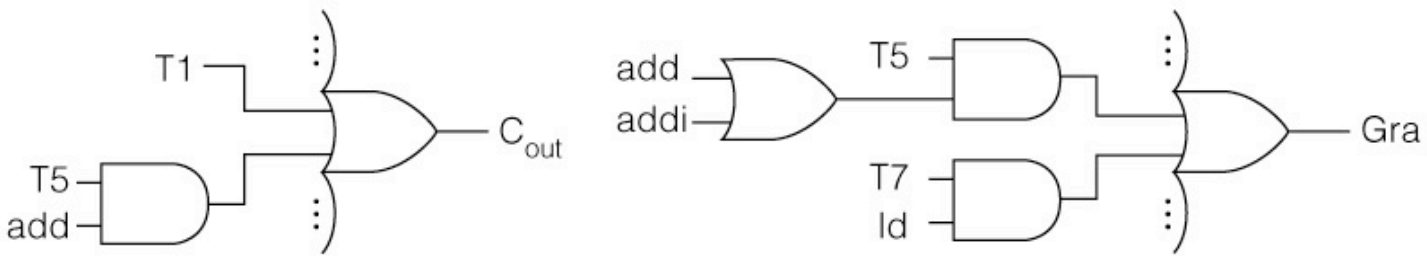
# Use of Data Path Conditions in Control Signal Logic

| Step | Control Sequence |
|------|------------------|
| T0. | $PC_{out}$, $MA_{in}$, Inc4, $C_{in}$, Read |
| T1. | $C_{out}$, $PC_{in}$, Wait |
| T2. | $MD_{out}$, $IR_{in}$ |

### add

| Step | Control Sequence |
|------|------------------|
| T3. | Grb, $R_{out}$, $A_{in}$ |
| T4. | Grc, $R_{out}$, ADD, $C_{in}$ |
| T5. | $C_{out}$, Gra, $R_{in}$, End |

### addi

| Step | Control Sequence |
|------|------------------|
| T3. | Grb, $R_{out}$, $A_{in}$ |
| T4. | $c2_{out}$, ADD, $C_{in}$ |
| T5. | $C_{out}$, Gra, $R_{in}$, End |

### st

| Step | Control Sequence |
|------|------------------|
| T3. | Grb, $BA_{out}$, $A_{in}$ |
| T4. | $c2_{out}$, ADD, $C_{in}$ |
| T5. | $C_{out}$, $MA_{in}$ |
| T6. | Gra, $R_{out}$, $MD_{in}$, Write |
| T7. | Wait, End |

### shr

| Step | Control Sequence |
|------|------------------|
| T3. | $c1_{out}$, Ld |
| T4. | $n=0 \rightarrow$ (Grc, $R_{out}$, Ld) |
| T5. | Grb, $R_{out}$, C=B |
| T6. | $n \neq 0 \rightarrow$ ($C_{out}$, SHR, $C_{in}$, Decr, Goto7) |
| T7. | $C_{out}$, Gra, $R_{in}$, End |

● ● ●

Example: Grc = T4·add + T4·(n=0)·shr + ...

Fig. 4.13   Generation of the logic for $C_{out}$ and $G_{ra}$



Copyright © 2004 Pearson Prentice Hall, Inc.

# Fig. 4.14   Branching in the Control Unit



Mck   Enable

Step generator

CountIn

Counter

Control
step
decoder

4

0110

Load

Reset
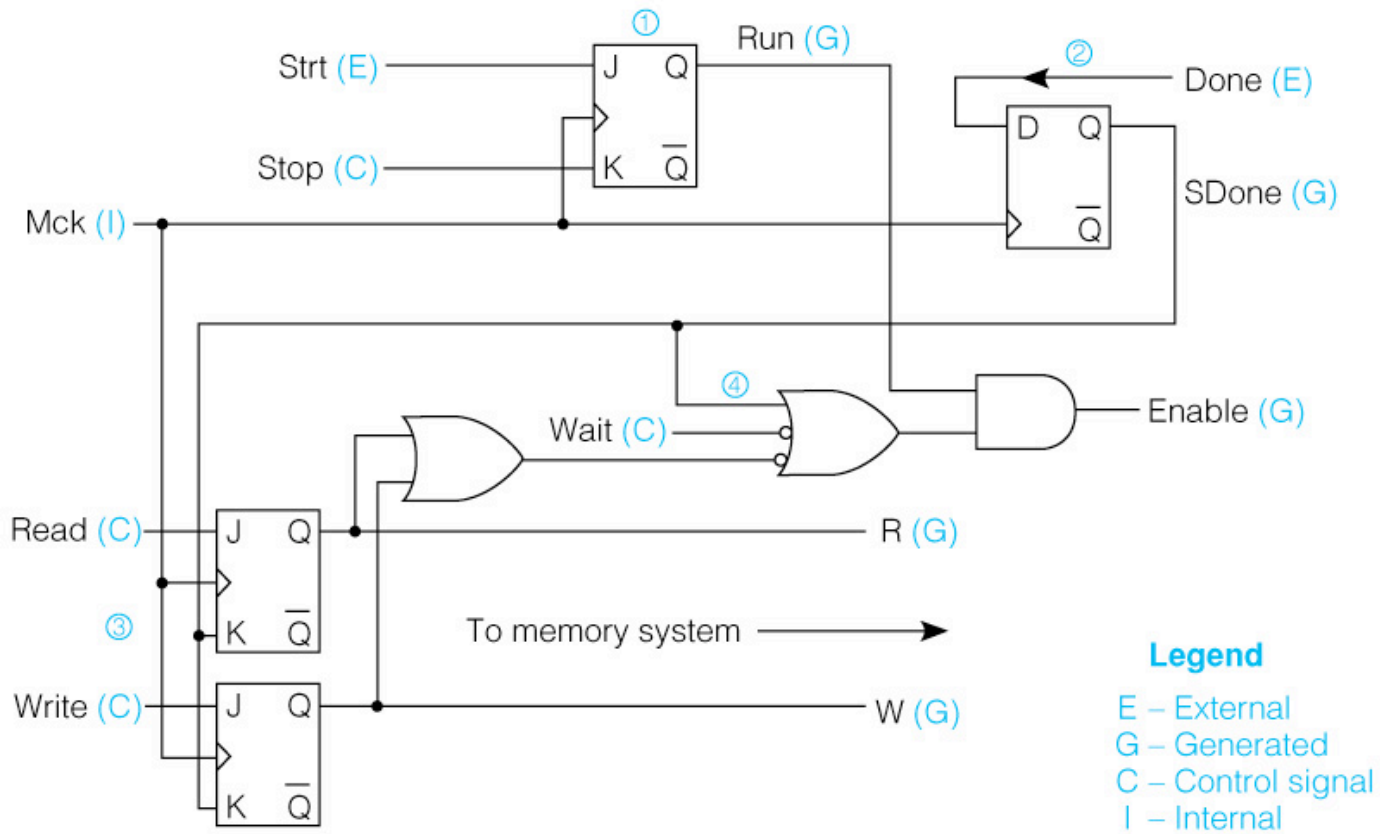
Goto6
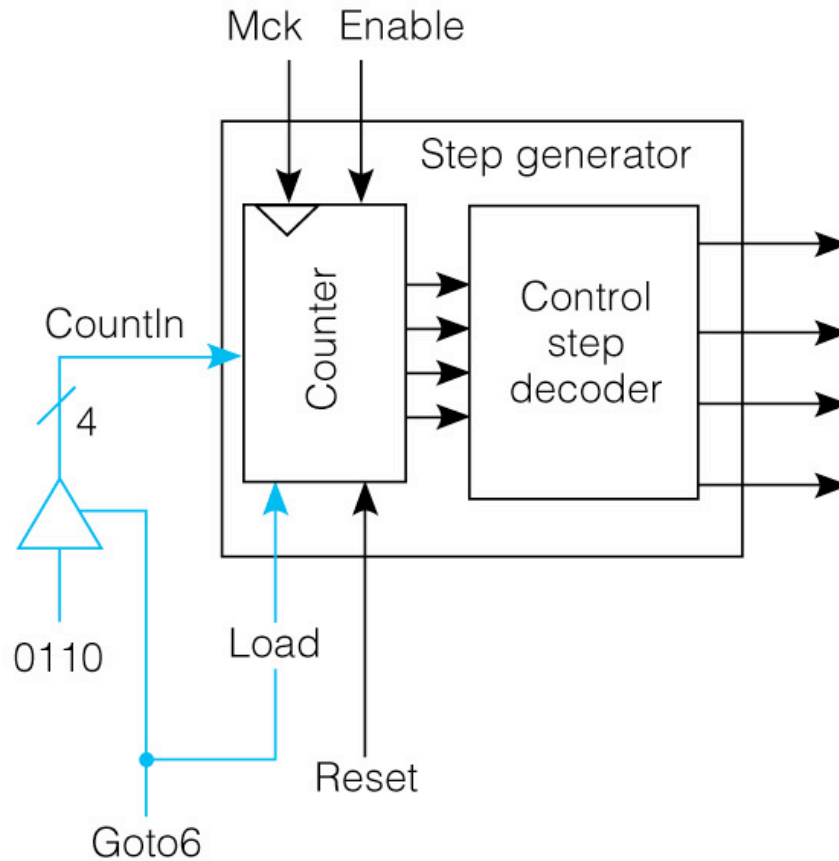
Copyright © 2004 Pearson Prentice Hall, Inc.

- 3-state gates allow 6 to be applied to counter input
- Reset will synchronously reset counter to step T0

# Fig. 4.15 Clocking Logic: Start, Stop, and Memory Synchronization



Copyright © 2004 Pearson Prentice Hall, Inc.

■ **Mck is master clock oscillator**

# Have Completed One-Bus Design of SRC

- High level architecture block diagram
- Concrete RTN steps
- Hardware design of registers and data path logic
- Revision of concrete RTN steps where needed
- Control sequences
- Register clocking decisions
- Logic equations for control signals
- Time step generator design
- Clock run, stop, and synchronization logic

# Other Architectural designs will require a different RTN

- More data paths allow more things to be done in one step

- Consider a two bus design

- By separating input and output of ALU on different buses, the C register is eliminated

- Steps can be saved by strobing ALU results directly into their destinations

# Fig. 4.16 The 2-bus Microarchitecture
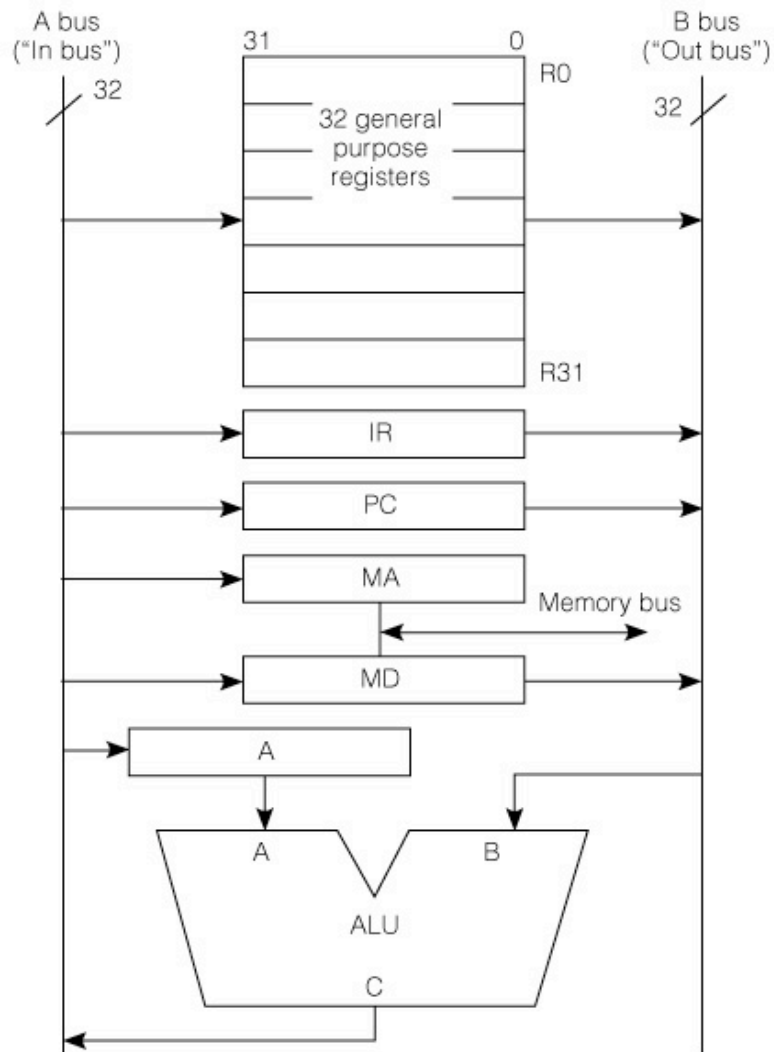
- Bus A carries data going into registers
- Bus B carries data being gated out of registers
- ALU function C=B is used for all simple register transfers

A bus ("In bus")

31                0

32 general purpose registers

R0

R31

IR

PC

MA

MD

Memory bus

A

B bus ("Out bus")

32

A    B

ALU

C

Copyright © 2004 Pearson Prentice Hall, Inc.

# Tbl 4.13  Concrete RTN and Control Sequence for 2-bus SRC add

| Step | Concrete RTN | Control Sequence |
|------|--------------|------------------|
| T0. | MA ← PC; | $PC_{out}$, C=B, $MA_{in}$, Read |
| T1. | PC ← PC + 4: MD ← M[MA]; | $PC_{out}$, Inc4, $PC_{in}$, Wait |
| T2. | IR ← MD; | $MD_{out}$, C=B, $IR_{in}$ |
| T3. | A ← R[rb]; | Grb, $R_{out}$, C=B, $A_{in}$ |
| T4. | R[ra] ← A + R[rc]; | Grc, $R_{out}$, ADD, Sra, $R_{in}$, End |

- Note the appearance of Grc to gate the output of the register rc onto the B bus and Sra to select ra to receive data strobed from the A bus

- Two register select decoders will be needed

- Transparent latches will be required for MA at step T0

C
S
D
A
2/e

$$%Speedup \quad = \quad \frac{T_{1-bus} - T_{2-bus}}{T_{2-bus}} \times 100$$

$Where$

$$T \quad = \quad Exec'n.Time \quad = IC \quad \times \quad CPI \quad \times \quad \tau$$

# Speedup Due To Going to 2 Buses

• Assume for now that IC and t don't change in going from 1 bus to 2 buses
• Naively assume that CPI goes from 8 to 7 clocks.

$$\%Speedup = \frac{T_{1-bus} - T_{2-bus}}{T_{2-bus}} \times 100$$

$$= \frac{IC \times 8 \times \tau - IC \times 7 \times \tau}{IC \times 7 \times \tau} \times 100 = \frac{8-7}{7} \times 100 = 14\%$$

Class Problem:
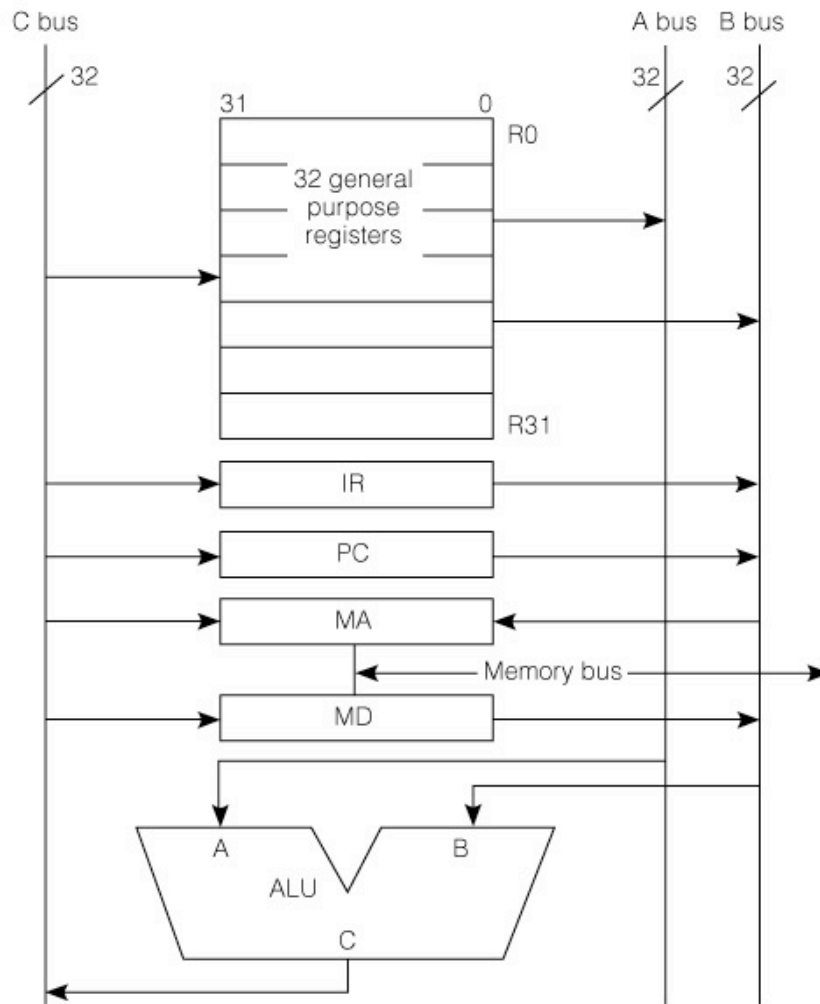How will this speedup change if clock period of 2-bus machine is increased by 10%?

Computer Systems Design and Architecture Second Edition © 2004 Prentice Hall

# 3-bus Architecture Shortens Sequences Even More

- A 3-bus architecture allows both operand inputs and the output of the ALU to be connected to buses

- Both the C output register and the A input register are eliminated

- Careful connection of register inputs and outputs can allow multiple RTs in a step

# Fig. 4.17   The 3-Bus SRC Design

C bus

A bus   B bus

31                    0

R0

32 general
purpose
registers

R31

IR

PC

MA

Memory bus

MD

A          B

ALU

C

Copyright © 2004 Pearson Prentice Hall, Inc.

- A-bus is ALU operand 1, B-bus is ALU operand 2, and C-bus is ALU output

- Note MA input connected to the B-bus

# Tbl 4.15  SRC add Instruction for the 3-bus Microarchitecture

| Step | Concrete RTN | Control Sequence |
|------|-------------|------------------|
| T0. | MA ← PC: PC ← PC + 4: MD ← M[MA]; | $PC_{out}$, $MA_{in}$, Inc4, $PC_{in}$, Read, Wait |
| T1. | IR ← MD; | $MD_{out}$, C=B, $IR_{in}$ |
| T2. | R[ra] ← R[rb] + R[rc]; | GArc, $RA_{out}$, GBrb, $RB_{out}$, ADD, Sra, $R_{in}$, End |

- Note the use of 3 register selection signals in step T2: GArc, GBrb, and Sra

- In step T0, PC moves to MA over bus B and goes through the ALU Inc4 operation to reach PC again by way of bus C
  - PC must be edge triggered or master-slave

- Once more MA must be a transparent latch

# Performance and Design

- How does going to three buses affect performance?
- Assume average CPI goes from 8 to 4, while $\tau$ increases by 10%:

$$\%Speedup = \frac{IC \times 8 \times \tau - IC \times 4 \times 1.1\tau}{IC \times 4 \times 1.1\tau} \times 100 = \frac{8 - 4.4}{4.4} \times 100 = 82\%$$

# Processor Reset Function

- Reset sets program counter to a fixed value
  - May be a hardwired value, or
  - contents of a memory cell whose address is hardwired
- The control step counter is reset
- Pending exceptions are prevented, so initialization code is not interrupted
- It may set condition codes (if any) to known state
- It may clear some processor state registers
- A "soft" reset makes minimal changes: PC, T (T-step counter)
- A "hard" reset initializes more processor state

# SRC Reset Capability

- We specify both a hard and soft reset for SRC
- The Strt signal will do a hard reset
  - It is effective only when machine is stopped
  - It resets the PC to zero
  - It resets all 32 general registers to zero
- The Soft Reset signal is effective when the machine is running
  - It sets PC to zero
  - It restarts instruction fetch
  - It clears the Reset signal
- Actions are described in instruction_interpretation

Processor State

Strt:                                   Start signal
Rst:                                    External reset signal

instruction_interpretation := (
        ¬Run∧Strt → (Run ← 1: PC, R[0..31] ← 0);
Run∧¬Rst → (IR ← M[PC]: PC ← PC + 4;
        instruction_execution):
Run∧Rst → ( Rst ← 0: PC ← 0); instruction_interpretation):

# Resetting in the Middle of Instruction Execution

- The abstract RTN implies that reset takes effect after the current instruction is done

- To describe reset during an instruction, we must go from abstract to concrete RTN

  - Questions for discussion:
    - Why might we want to reset in the middle of an instruction?
    - How would we reset in the middle of an instruction?

# Tbl 4.17 Concrete RTN Describing Reset During add Instruction Execution

| Step | Concrete RTN |
|------|--------------|
| T0 | ¬Reset → (MA ← PC: C ← PC + 4): |
|    | Reset → (Reset ← 0: PC ← 0: T ←0): |
| T1 | ¬Reset → (MD ← M[MA]: P ← C): |
|    | Reset → (Reset ← 0: PC ← 0: T ← 0): |
| T2 | ¬Reset → (IR ← MD): |
|    | Reset → (Reset ← 0: PC ← 0: T ← 0): |
| T3 | ¬Reset → (A ← R[rb]): |
|    | Reset → (Reset ← 0: PC ← 0: T ← 0): |
| T4 | ¬Reset → (C ← A + R[rc]): |
|    | Reset → (Reset ← 0: PC ← 0: T ← 0): |
| T5 | ¬Reset →  (R[ra ] ← C): |
|    | Reset → (Reset ← 0: PC ← 0: T ← 0): |

# Control Sequences Including the Reset Function

Step      Control Sequence
T0.       $\neg$Reset $\rightarrow$ (PC$_{out}$, MA$_{in}$, Inc4, C$_{in}$, Read):
          Reset $\rightarrow$ (ClrPC, ClrR, Goto0):
T1        $\neg$Reset $\rightarrow$ (C$_{out}$, PC$_{in}$, Wait):
          Reset $\rightarrow$ (ClrPC, ClrR, Goto0):
                    • • •

- ClrPC clears the program counter to all zeros, and ClrR clears the one bit Reset flip-flop

- Because the same reset actions are in every step of every instruction, their control signals are independent of time step or op code

# General Comments on Exceptions

- An exception is an event that causes a change in the program specified flow of control

- Because normal program execution is interrupted, they are often called interrupts

- We will use exception for the general term and use interrupt for an exception caused by an external event, such as an I/O device condition

- The usage is not standard. Other books use these words with other distinctions, or none

# Combined Hardware/Software Response to an Exception

- The system must control the type of exceptions it will process at any given time

- The state of the running program is saved when an allowed exception occurs

- Control is transferred to the correct software routine, or "handler" for this exception

- This exception, and others of less or equal importance are disallowed during the handler

- The state of the interrupted program is restored at the end of execution of the handler

# Hardware Required to Support Exceptions

- To determine relative importance, a priority number is associated with every exception

- Hardware must save and change the PC, since without it no program execution is possible

- Hardware must disable the current exception lest is interrupt the handler before it can start

- Address of the handler is called the exception vector and is a hardware function of the exception type

- Exceptions must access a save area for PC and other hardware saved items

  - Choices are special registers or a hardware stack

# New Instructions Needed to Support Exceptions

- An instruction executed at the end of the handler must reverse the state changes done by hardware when the exception occurred

- There must be instructions to control what exceptions are allowed

    - The simplest of these enable or disable all exceptions

- If processor state is stored in special registers on an exception, instructions are needed to save and restore these registers

# Kinds of Exceptions

- System reset

- Exceptions associated with memory access

  - Machine check exceptions

  - Data access exceptions

  - Instruction access exceptions

  - Alignment exceptions

- Program exceptions

- Miscellaneous hardware exceptions

- Trace and debugging exceptions

- Non-maskable exceptions
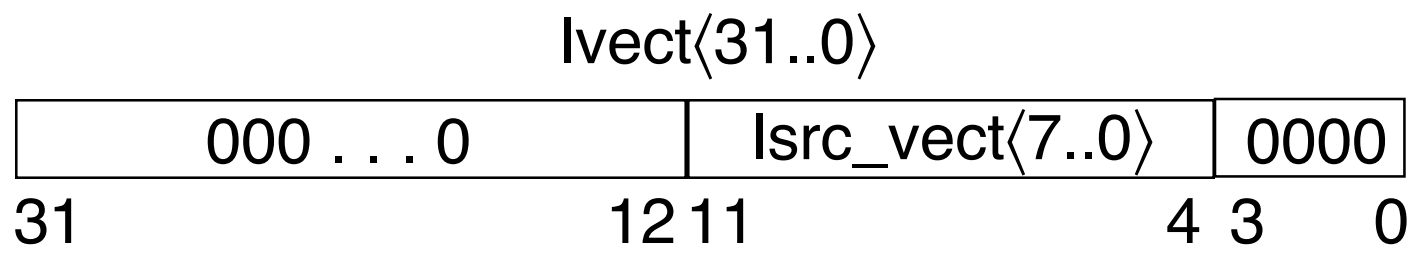
- External exceptions—interrupts

# An Interrupt Facility for SRC

- The exception mechanism for SRC handles external interrupts
- There are no priorities, but only a simple enable and disable mechanism
- The PC and information about the source of the interrupt are stored in special registers
  - Any other state saving is done by software
- The interrupt source supplies 8 bits that are used to generate the interrupt vector
- It also supplies a 16 bit code carrying information about the cause of the interrupt

Processor interrupt mechanism

From Dev. → ireq:                interrupt request signal
To Dev. → iack:                interrupt acknowledge signal
Internal → IE:                one bit interrupt enable flag
to CPU → IPC$\langle 31..0 \rangle$:                storage for PC saved upon interrupt
" → II$\langle 15..0 \rangle$:                info. on source of last interrupt
From Dev. → Isrc_info$\langle 15..0 \rangle$: information from interrupt source
From Dev → Isrc_vect$\langle 7..0 \rangle$:                type code from interrupt source
Internal → Ivect$\langle 31..0 \rangle$:= 20@0#Isrc_vect$\langle 7..0 \rangle$#4@0:

Ivect$\langle 31..0 \rangle$

| 000 . . . 0 | Isrc_vect$\langle 7..0 \rangle$ | 0000 |
|---|---|---|
| 31                              12 | 11                              4 | 3        0 |

# SRC Instruction Interpretation Modified for Interrupts

instruction_interpretation :=
(¬Run∧Strt → Run ← 1:
Run∧¬(ireq∧IE) → (IR ← M[PC]: PC ← PC + 4; instruction_execution):
Run∧(ireq∧IE) → (IPC ← PC⟨31..0⟩:
  II⟨15..0⟩ ← Isrc_info⟨15..0⟩: iack ← 1:
  IE ← 0: PC ← Ivect⟨31..0⟩; iack ← 0);
  instruction_interpretation);

- If interrupts are enabled, PC and interrupt info. are stored in IPC and II, respectively
  - With multiple requests, external priority circuit (discussed in later chapter) determines which vector & info. are returned
- Interrupts are disabled
- The acknowledge signal is pulsed

# SRC Instructions to Support Interrupts

Return from interrupt instruction
rfi (:= op = 29 ) → (PC ← IPC: IE ← 1):

Save and restore interrupt state
svi (:= op = 16) → (R[ra]⟨15..0⟩ ← II⟨15..0⟩: R[rb] ← IPC⟨31..0⟩):
ri (:= op = 17) → (II⟨15..0⟩ ← R[ra]⟨15..0⟩ : IPC⟨31..0⟩ ← R[rb]):

Enable and disable interrupt system
een (:= op = 10 ) → (IE ← 1):
edi (:= op = 11 ) → (IE ← 0):

- The 2 rfi actions are indivisible, can't een & branch

| Step | ¬(ireq∧IE) | Concrete RTN | (ireq∧IE) |
|------|-----------|--------------|-----------|
| T0. | (¬(ireq∧IE) → (<br>    MA ← PC: C ← PC+4): | | (ireq∧IE) → (IPC ← PC: II ← Isrc_info:<br>    IE ← 0: PC← 20@0#Isrc_vect⟨7..0⟩#0000:<br>    Iack←1); |
| T1. | MD ← M[MA] : PC ← C; | | Iack ← 0: End; |
| T2. | IR ← MD; | | |

- PC could be transferred to IPC over the bus
- II and IPC probably have separate inputs for the externally supplied values
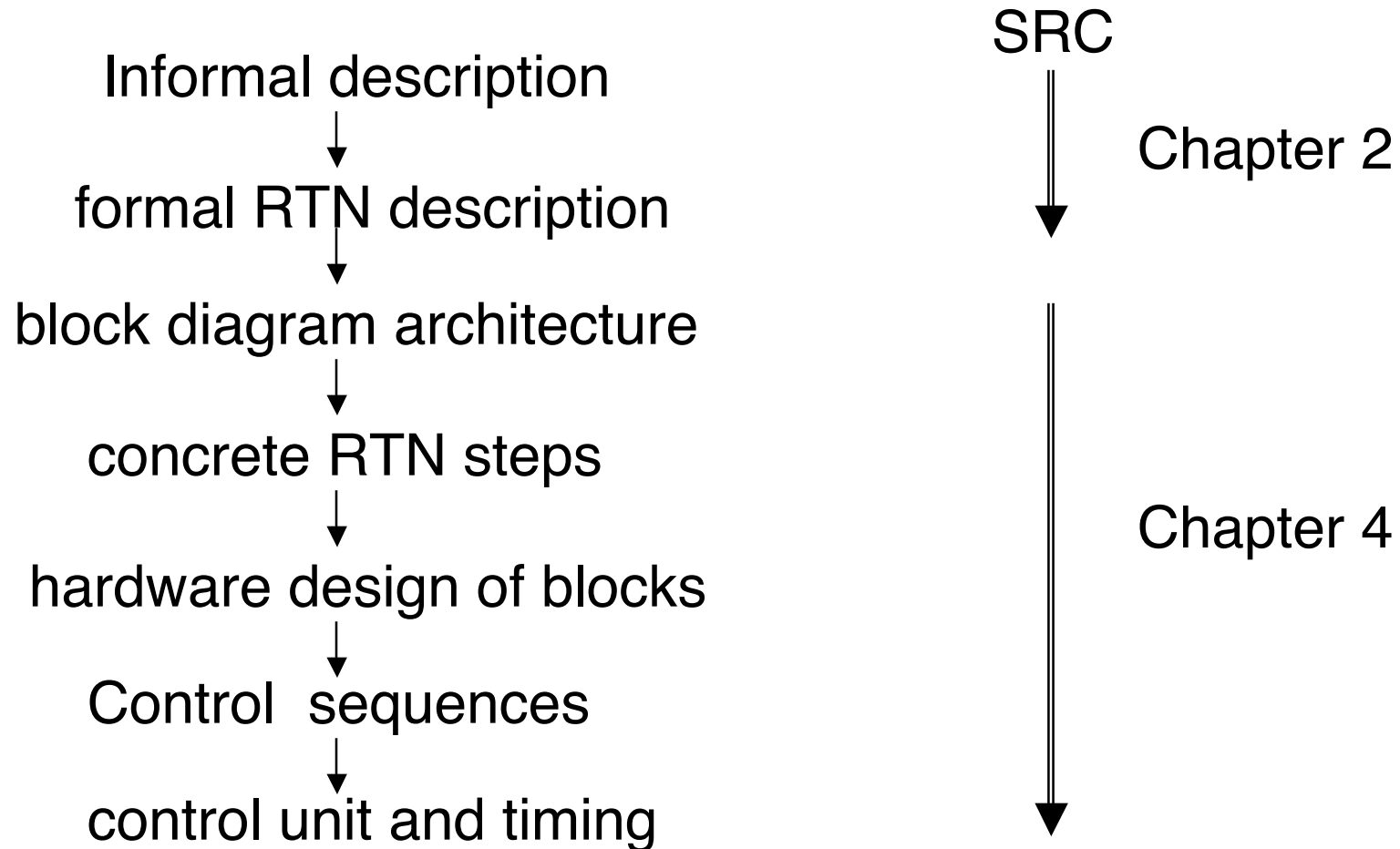- Iack is pulsed, described as ←1; ←0, which is easier as a control signal than in RTN

# Exceptions During Instruction Execution

- Some exceptions occur in the middle of instructions
  - Some CISCs have very long instructions, like string move
  - Some exception conditions prevent instruction completion, like uninstalled memory
- To handle this sort of exception, the CPU must make special provision for restarting
  - Partially completed actions must be reversed so the instruction can be re-executed after exception handling
  - Information about the internal CPU state must be saved so that the instruction can resume where it left off
- We will see that this problem is acute with pipeline designs—always in middle of instructions.

# Recap of the Design Process: the Main Topic of Chap. 4

Informal description

↓

formal RTN description

↓

block diagram architecture

↓

concrete RTN steps

↓

hardware design of blocks

↓

Control  sequences

↓

control unit and timing

SRC

Chapter 2

Chapter 4

# Chapter 4 Summary

- Chapter 4 has done a non pipelined data path, and a hardwired controller design for SRC

- The concepts of data path block diagrams, concrete RTN, control sequences, control logic equations, step counter control, and clocking have been introduced

- The effect of different data path architectures on the concrete RTN was briefly explored

- We have begun to make simple, quantitative estimates of the impact of hardware design on performance

- Hard and soft resets were designed

- A simple exception mechanism was supplied for SRC