CPE300: Digital System Architecture and Design

Fall 2011

MW 17:30-18:45 CBC C316

1-Bus Architecture and Datapath

10262011

http://www.egr.unlv.edu/~b1morris/cpe300/

Outline

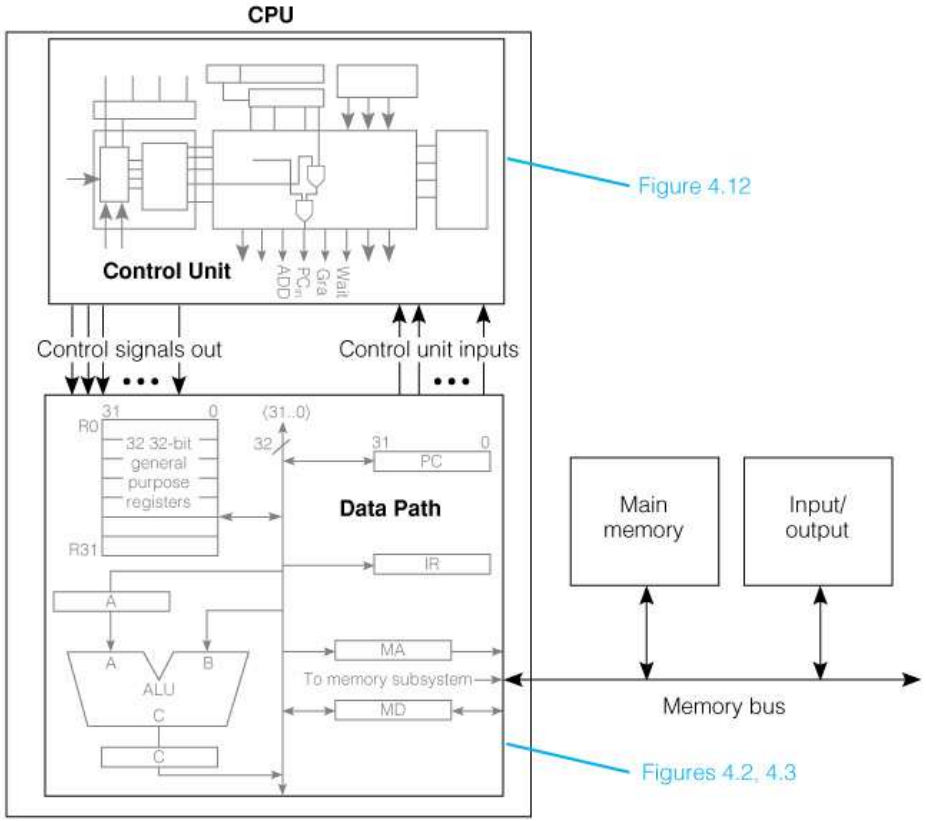
- 1-Bus Microarchitecture and Datapath Review
- 1-Bus Logic Design
- Control Unit

Register Transfer Descriptions

- Abstract RTN
  - Defines “what” not the “how” (Chapter 2)
    - Overall effect of instructions on programmer-visible registers
  - Implementation independent
    - Registers and operations
- Concrete RTN
  - Detailed register transfer steps in datapath to produce overall effect
    - Dependent on implementation details
  - Steps correspond to processor clock pulses

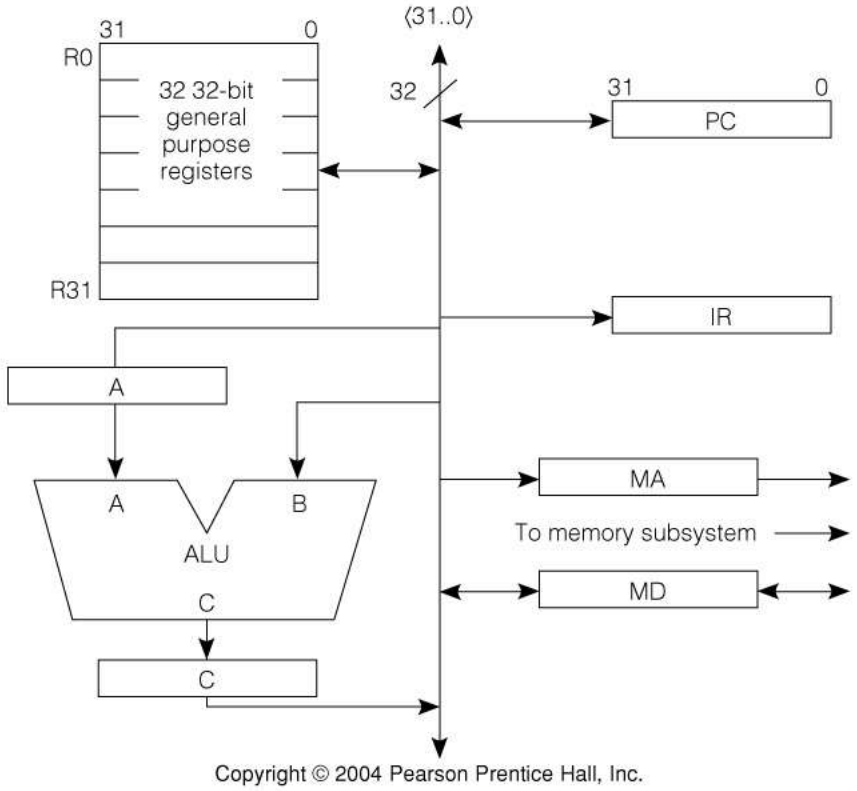
1-Bus SRC Microarchitecture

- 5 classic components of computer
  - Memory, Input, Output
  - CPU – Control and Datapath



Microarchitecture Constraints

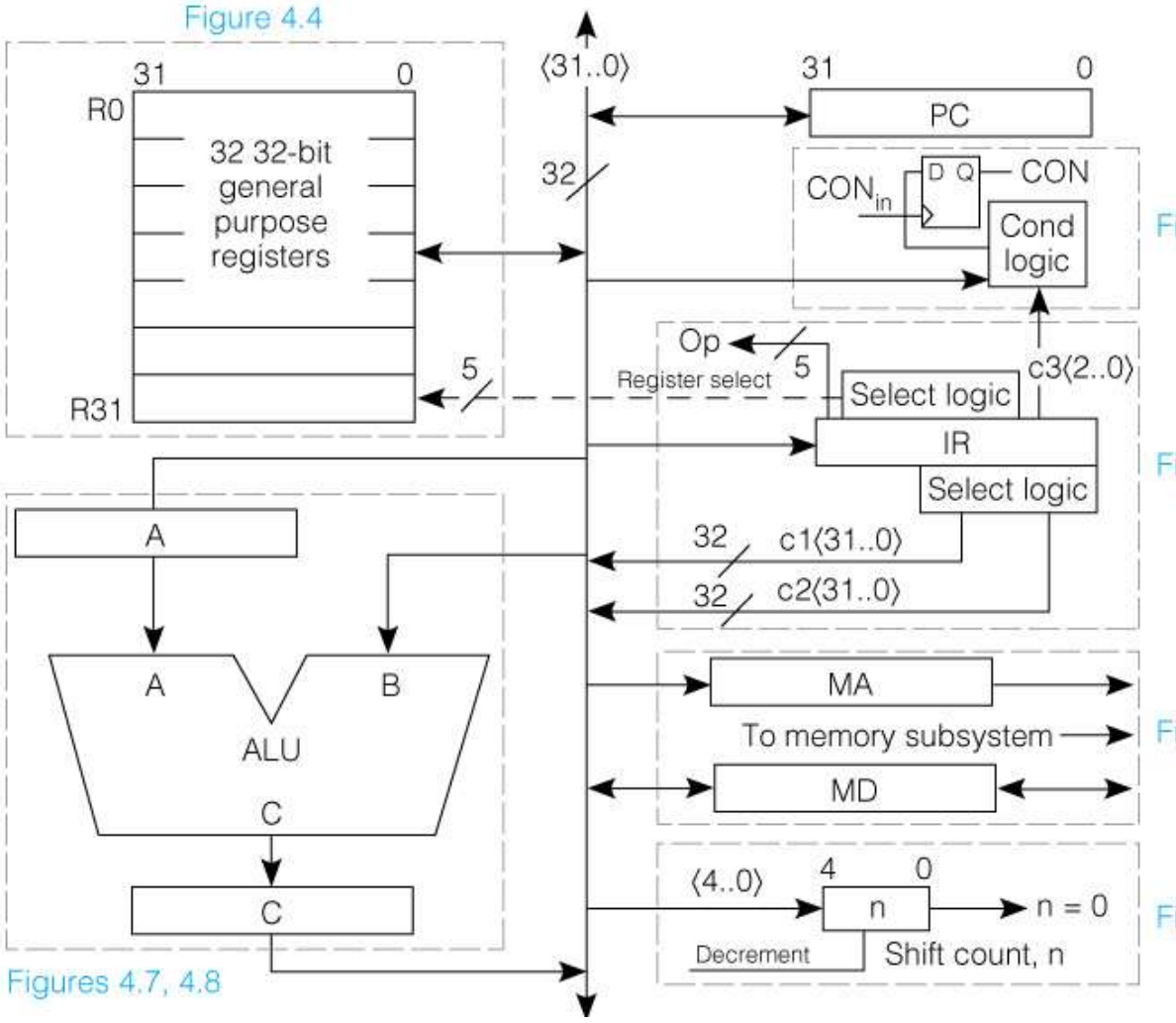
- One bus connecting registers
  - Only single register transfer at a time
- Memory address must be copied into memory address (MA) register by CPU
- Memory data written from or read into memory data (MD) register
- ALU operation
  - First operand always registered in A
  - Second operand always comes from bus
  - Result registered in C
- Information into IR and MA only from bus
  - Decoder (not shown) interprets contents of IR
  - MA supplies address to memory not CPU bus



Copyright © 2004 Pearson Prentice Hall, Inc.

More Complete View of 1-Bus SRC Design

- Concrete RTN adds detail to the datapath



Condition bit flip-flop

IR register logic and data paths

Shift counter register

RTN for ADD Instruction

- Develop steps to execute instruction
- Abstract RTN
  - (IR←M[PC]: PC←PC+4; instsruction_execution);
  - Instruction_execution := (…
    - add(:=op=12)→R[ra]←R[rb]+R[rc]:
    - …);
- Concrete RTN
  - 3 concrete RT (T3, T4, T5)
  - 2 RT in To
  - 6 total clock cycles

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

RTN for ADD Instruction

- Develop steps to execute instruction
- Abstract RTN
  - (IR←M[PC]: PC←PC+4; instsruction_execution);
  - Instruction_execution := (…
    - add(:=op=12)→R[ra]←R[rb]+R[rc]:
    - …);
- Concrete RTN
  - 3 concrete RT (T3, T4, T5)
  - 2 RT in To
  - 6 total clock cycles

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

RTN for Load/Store Instruction

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

Step	RTN ld	RTN st
T0-T2	Instruction Fetch	
T3	A←(rb=0→0: rb≠0→R[rb]);	
T4	C←A+(16@IR<16>#IR<15..0>);	
T5	MA←C;	
T6	MD←M[MA];	MD←R[ra]
T7	R[ra]←MD;	M[MA]←MD;

T3, T4 are effective address arithmetic calculation

Notes for Load/Store RTN

- T0-T2 are same as for add (all instructions)
- T3-T5 are same for `ld` and `st` – calculate `disp`
- Need way to use `0` for `R[rb]` when `rb=0`
- 15-bit sign extension of `IR<16..0>` is needed
- Memory read into `MD` at T6 of `ld`
- Write of `MD` into memory at T7 of `st`

RTN for Conditional Branch

- Abstract RTN
  - 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 register is positive or zero
      - c3<2..0>=5→R[rc]<31>=1): ;if register is negative
- Concrete RTN

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

CON is 1-bit register that is set based on condition logic: the contents of c<2..0> and R[rc]

Notes on Conditional Branch RTN

- `c3<2..0>` are just 3 low order bits of IR
- `cond()` is evaluated by combinational logic circuit having inputs R[rc] and `c3<2..0>`
- One bit `CON` register is not accessible to the programmer
  - Holds intermediate output of combinational logic for the condition
- If branch succeeds
  - `PC` is replaced by contents of a general register

RTN for SRC Shift Right

- Abstract RTN
  - 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>): ;shift cnt const. field
- Concrete RTN

Step	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

T6 is repeated n times

Notes on Shift RTN

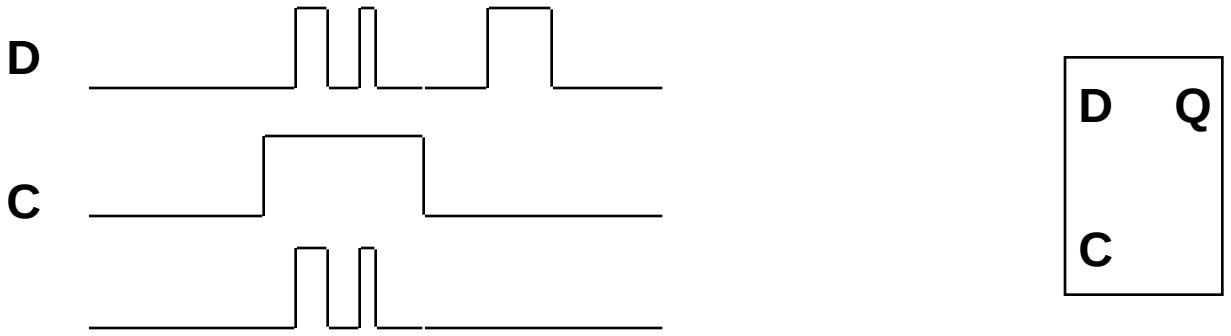
- Abstract RTN defines n with :=
- Concrete RTN has n as a physical register
- n is not only the shift count but used as a counter in step T6
  - T6 is repeated n times through recursive `Shr` call
  - Will require more complicated control, described later

Datapath/Control Unit Separation

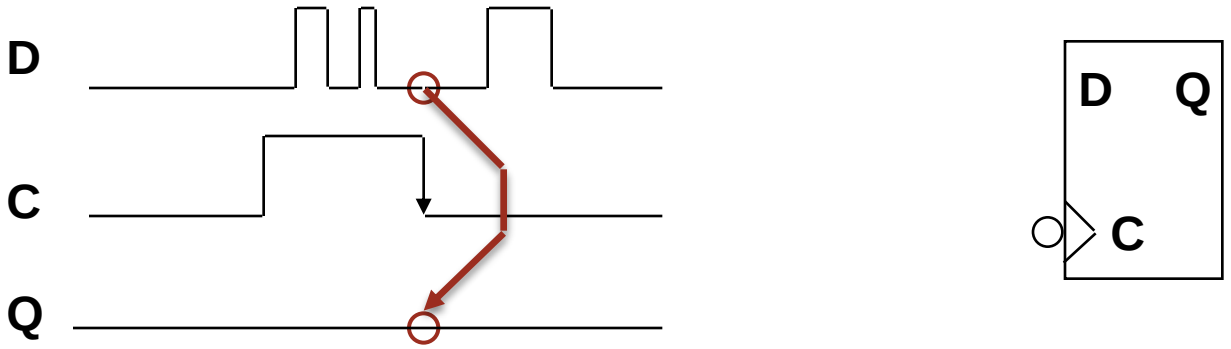
- Interface between datapath and control consists of gate and strobe signals
  - Gate – selects one of several values to apply to a common point (e.g. bus)
  - Strobe – changes the contents of a register (flip-flops) to match new inputs
- Type of flip-flop used in a register has significant impact on control and limited impact on datapath
  - Latch – simpler hardware but more complex timing
  - Edge triggered – simpler timing but approximately 2x hardware

Latch/Edge-Triggered Operation

- Latch output follows input while strobe is high

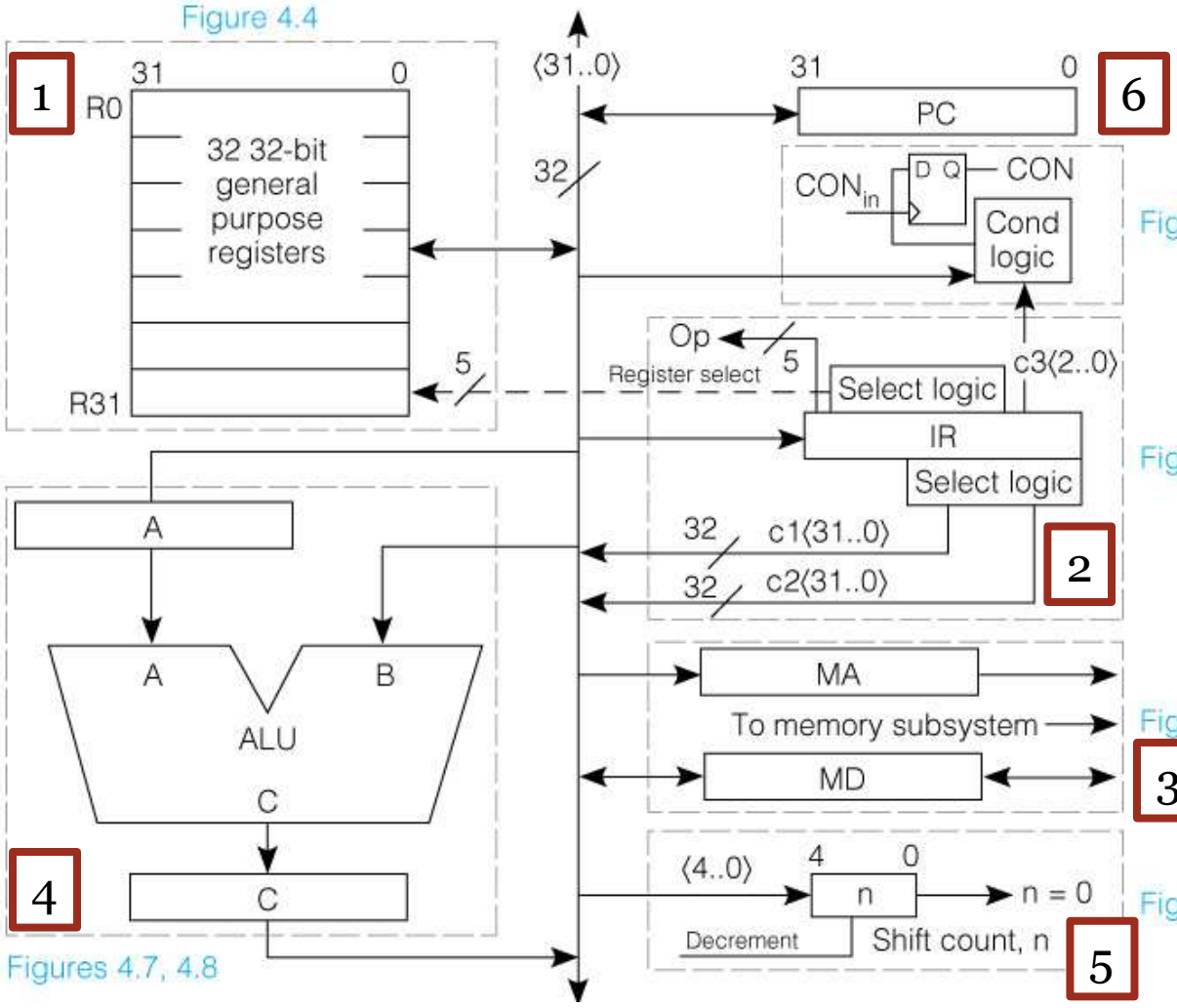


- Edge-triggering samples input at the edge time



More Complete View of 1-Bus SRC Design

- Add control signals and gate-level logic



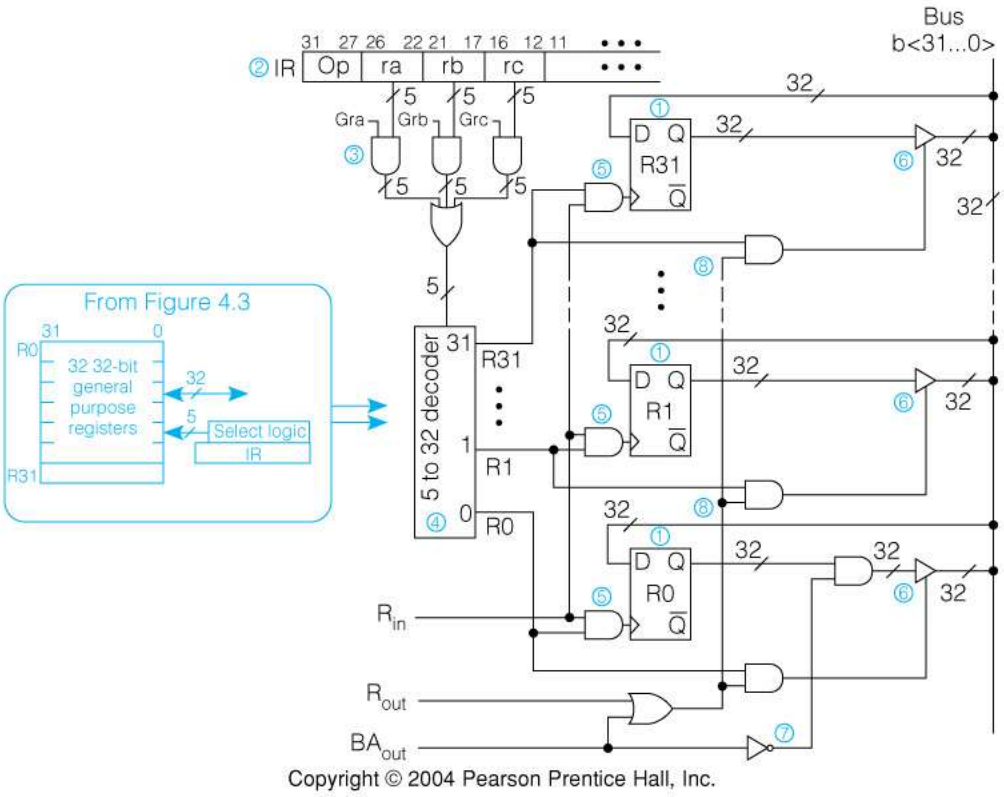
Condition bit flip-flop

IR register logic and data paths

Shift counter register

Register File and Control Signals

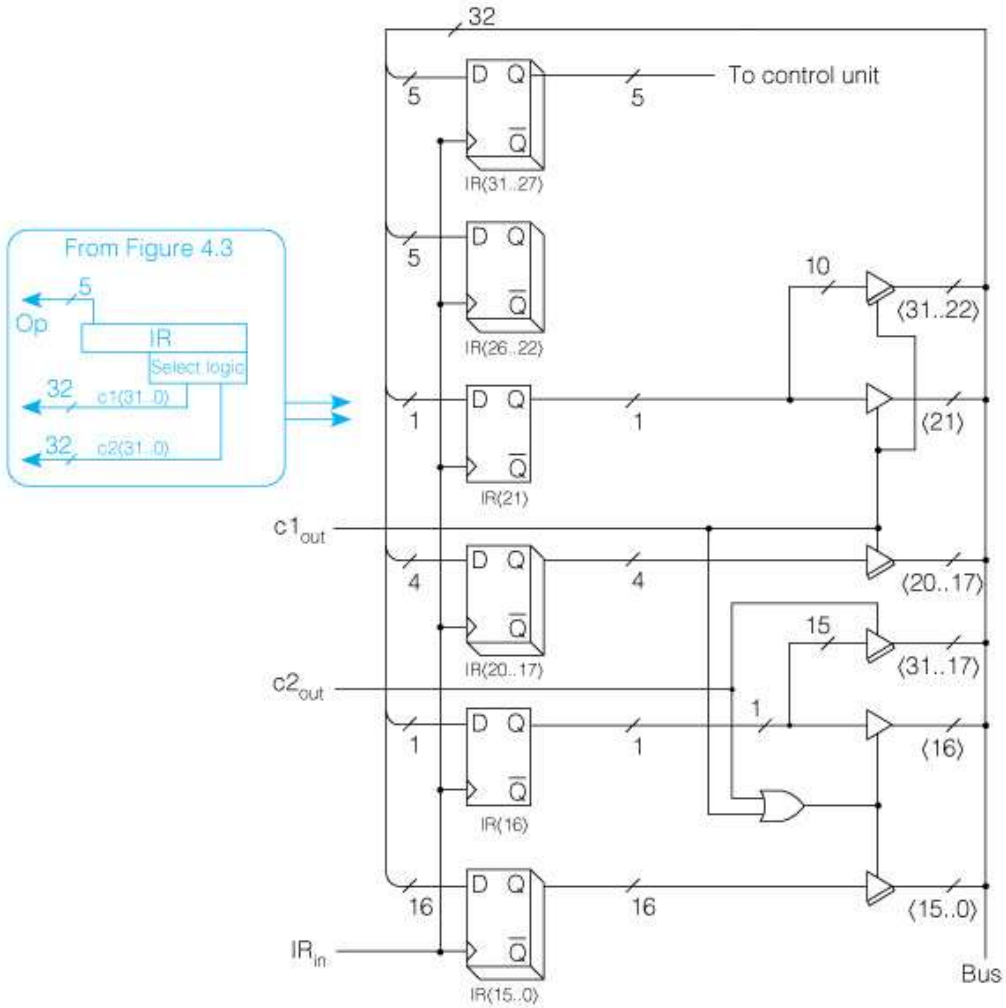
- Register selection
  - `IR` decode of register fields
  - `Grx` signal to gate register `rx` by decoder
- R_{out} gates selected register onto the bus
- R_{in} strobes selected register from the bus
- Base address out `BAout` gates zero signal when `R[0]` is selected



Copyright © 2004 Pearson Prentice Hall, Inc.

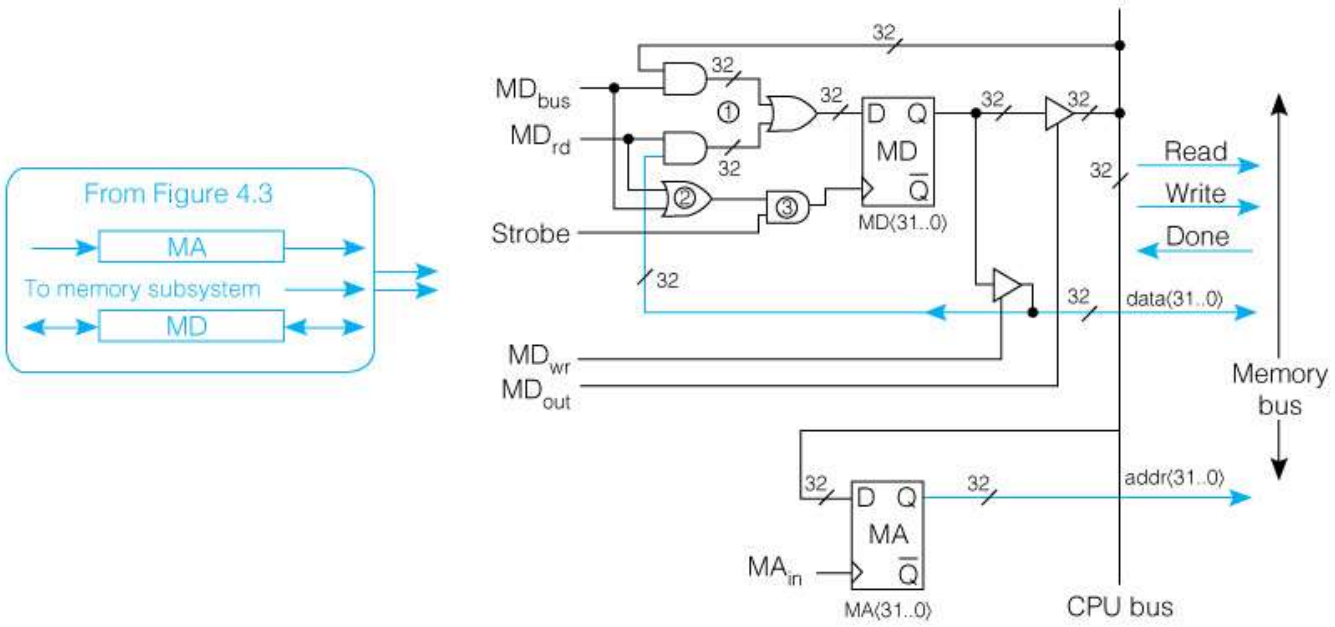
Extracting Constants/op from IR

- 3D blocks distinguish multi-bi elements
  - Register flip-flops
  - Tri-state bus drivers
- Sign bits fanned out from one to several bits and gated onto bus
  - `IR<21>` is sign bit of `c1` and must be sign extended
  - `IR<16>` is sign bit of `c2` and must be sign extended



Copyright © 2004 Pearson Prentice Hall, Inc.

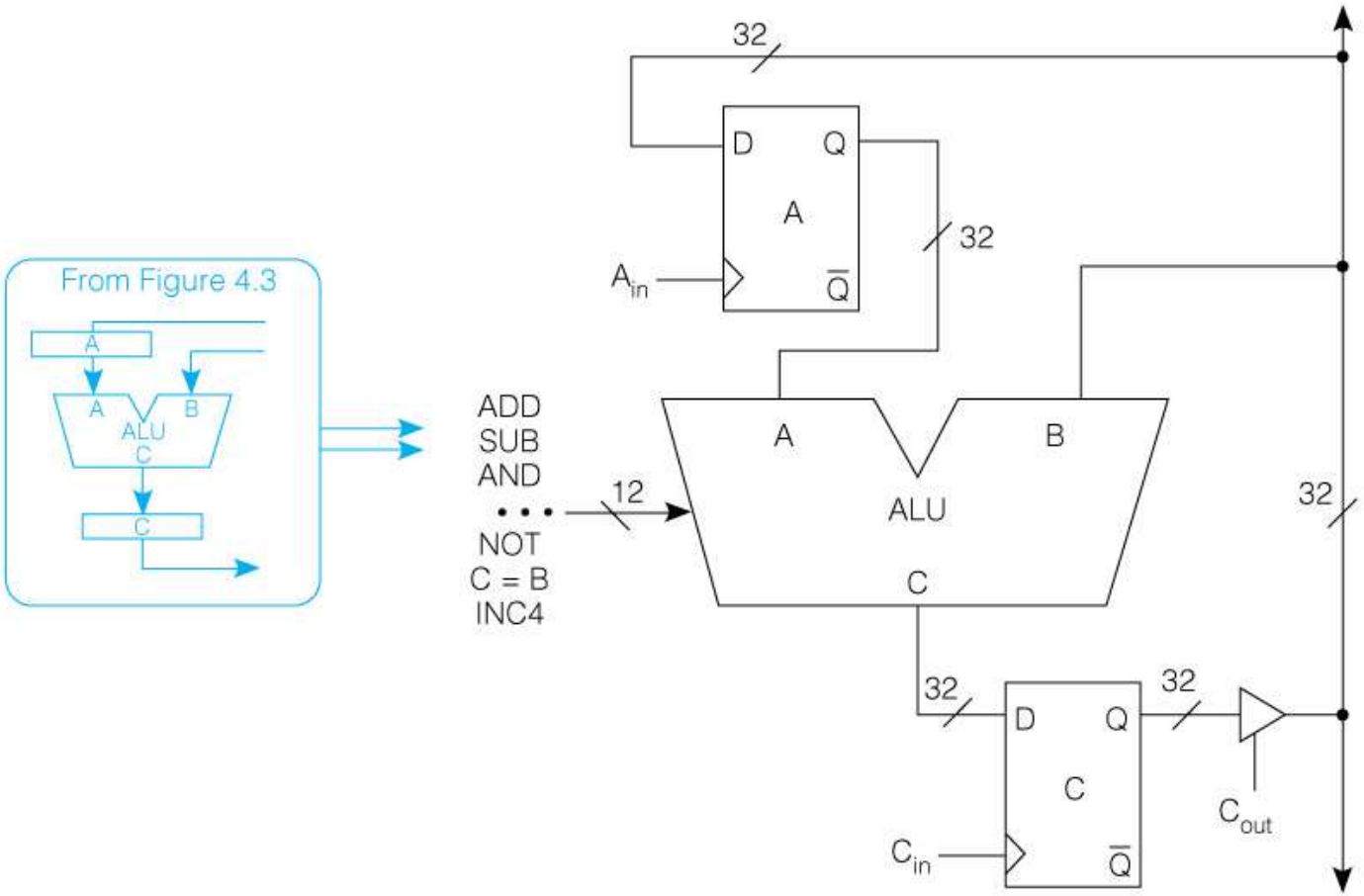
Memory Interface



- `MD` is loaded from memory bus or from CPU bus
- `MD` can drive memory bus or CPU bus
- `MA` only gets address from CPU processor bus

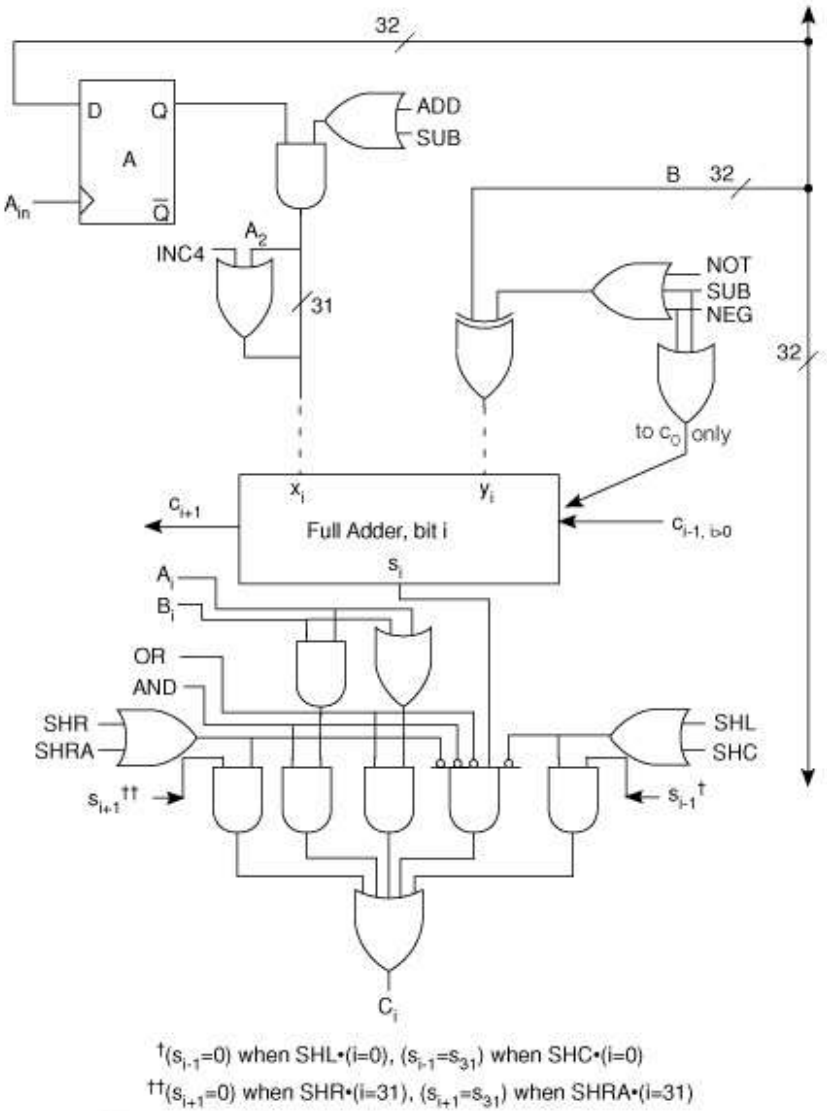
ALU and Associated Registers

- Add control lines to select ALU function
  - `INC4` for hardware supported `PC` increment



1-Bit ALU Logic-Level Design

`PC` increment



Negative numbers in `B`

`AND` gates select appropriate output

Control Sequences

- Register transfers are the concrete RTN
- Control sequence are the control signals that cause the RT

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	

Wait prevents control sequence from advancing to step T2 until memory asserts Done

Control Steps, Control Signals, and Timing

- Order control signals are written is irrelevant for a given time step
  - Step To:
    - $(\mathtt{Inc4},\ C_{in},\ PC_{out},\ MA_{in}) = (PC_{out},\ MA_{in},\ \mathtt{Inc4},\ C_{in})$
- Timing distinction is made between gates and strobes
  - Gates early, strobes late in clock cycle
- Memory read should start as early as possible to reduce wait time
- `MA` must have correct value before being used for a read

Control for ADD Instruction

- add(:=op=12)→R[ra]←R[rb]+R[rc]:

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	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

- Grx used to gate correct 5-bit register select code
- End signals the control to start over at step To

RTN for ADDI Instruction

- addi(:=op=13)→R[ra]←R[rb]+c2<16..0> {two's complement, sign-extend}:

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	A←R[rb]	Grb, R_{out}, A_{in}
T4	C←A+c2 {sign-extend};	$c2_{out}$, ADD, C_{in}
T5	R[ra]←C	C_{out}, Gra, R_{in}, End

- $C2_{out}$ signal sign extends IR<16..0> and gates it to the bus

RTN for `st` Instruction

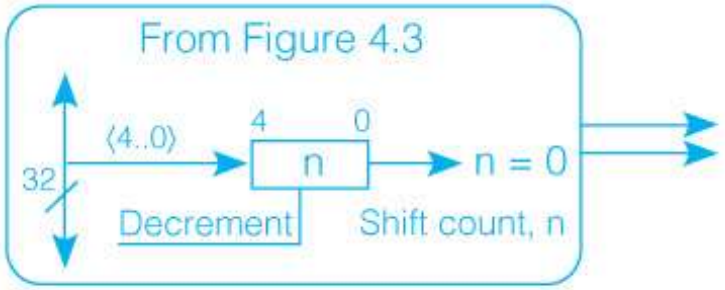
- `st(:=op=3)→M[disp]←R[ra]:`
  - `disp<31..0>:=((rb=0)→c2<16..0> {sign-extend}:`
    `(rb≠0)→R[rb]+c2<16..0>{sign-ext,2's comp}`

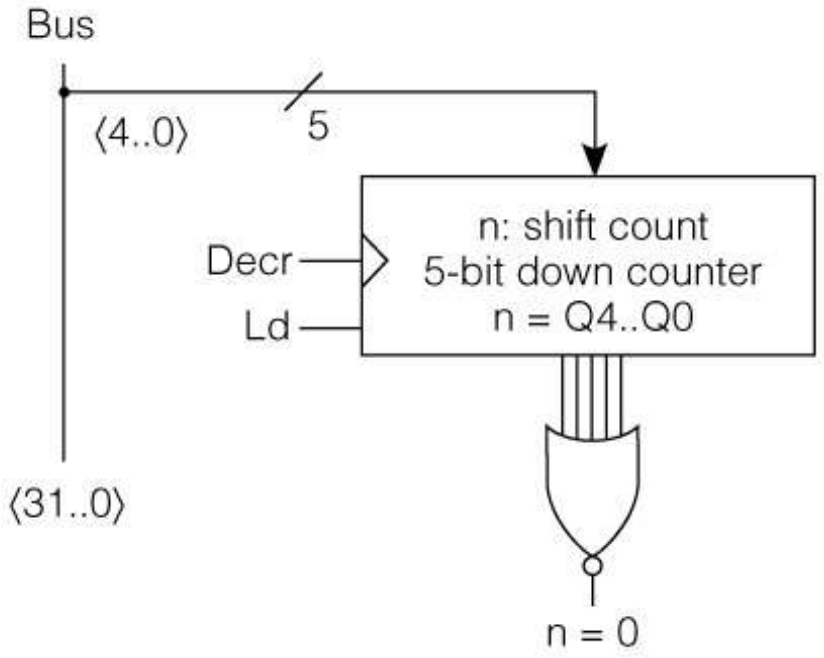
Step	Concrete RTN	Control Sequence
T0-T2	instruction_fetch	
T3	A←(rb=0→0: rb≠0→R[rb]);	Grb, BA_{out}, A_{in}
T4	C←A+(16@IR<16>#IR<15..0>);	$c2_{out}$, ADD, C_{in}
T5	MA←C;	C_{out}, MA_{in}
T6	MD←R[ra]	Gra, R_{out}, MD_{in}, Write
T7	M[MA]←MD;	Wait, End

- Notice the use of BA_{out} in step T3 not R_{out} as done in `addi`

Shift Counter

- Concrete RTN for `shr` relies upon a 5-bit register to hold the shift count
- Must load, decrement, and have a way to test if the contents equal 0





Control for Shift Instruction

- 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>): ;count const. field

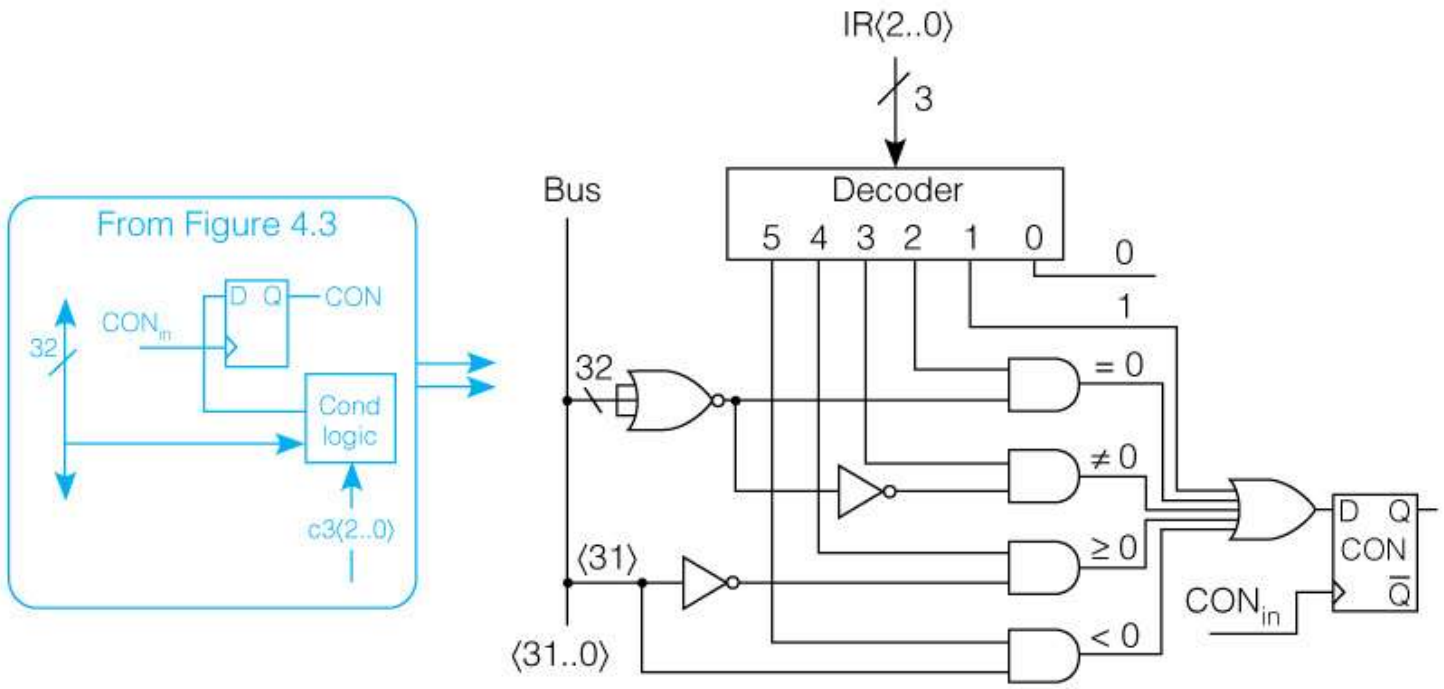
Step	Concrete RTN	Control Sequence
T0-T2	Instruction Fetch	
T3	n←IR<4..0>	c1out, Ld
T4	(n=0)→(n←R[rc]<4..0>);	n=0→(Grc, R_{out}, Ld)
T5	C←R[rb]	Grb, R_{out}, C=B, C_{in}
T6	Shr(:=n≠0→ (C<31..0>←0#C<31..1>: n←n-1; Shr));	n≠0→(C_{out}, SHR, C_{in}, Decr, Goto6)
T7	R[ra]←C	C_{out}, Gra, R_{in}, End

- Conditional control signals and repeating control are new concepts
  - Goto6 – repeats step T6 but must be carefully timed for the looping

Branching

- Branch conditions dependent on cond field an a register value (not flag or flag register)
  - 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 register is positive or zero
    - c3<2..0>=5→R[rc]<31>=1): ;if register is negative
- Logic expression for condition
  - cond = (c3<2..0>=1) ∨ (c3<2..0>=2) ∧ (R[rc]=0) ∨ (c3<2..0>=3) ∧ ¬(R[rc]=0) ∨ (c3<2..0>=4) ∧ ¬R[rc]<31> ∨ (c3<2..0>=5) ∧ R[rc]<31>

Conditional Value Computation



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

Control for Branch Instruction

- br(:=op=8)→(cond→PC←R[rb]):

Step	Concrete RTN	Control Sequence
T0-T2	Instruction Fetch	
T3	CON←cond(R[rc]);	Grc, R_{out}, CON_{in}
T4	CON→PC←R[rb];	Grb, R_{out}, CON→PC_{in}, End

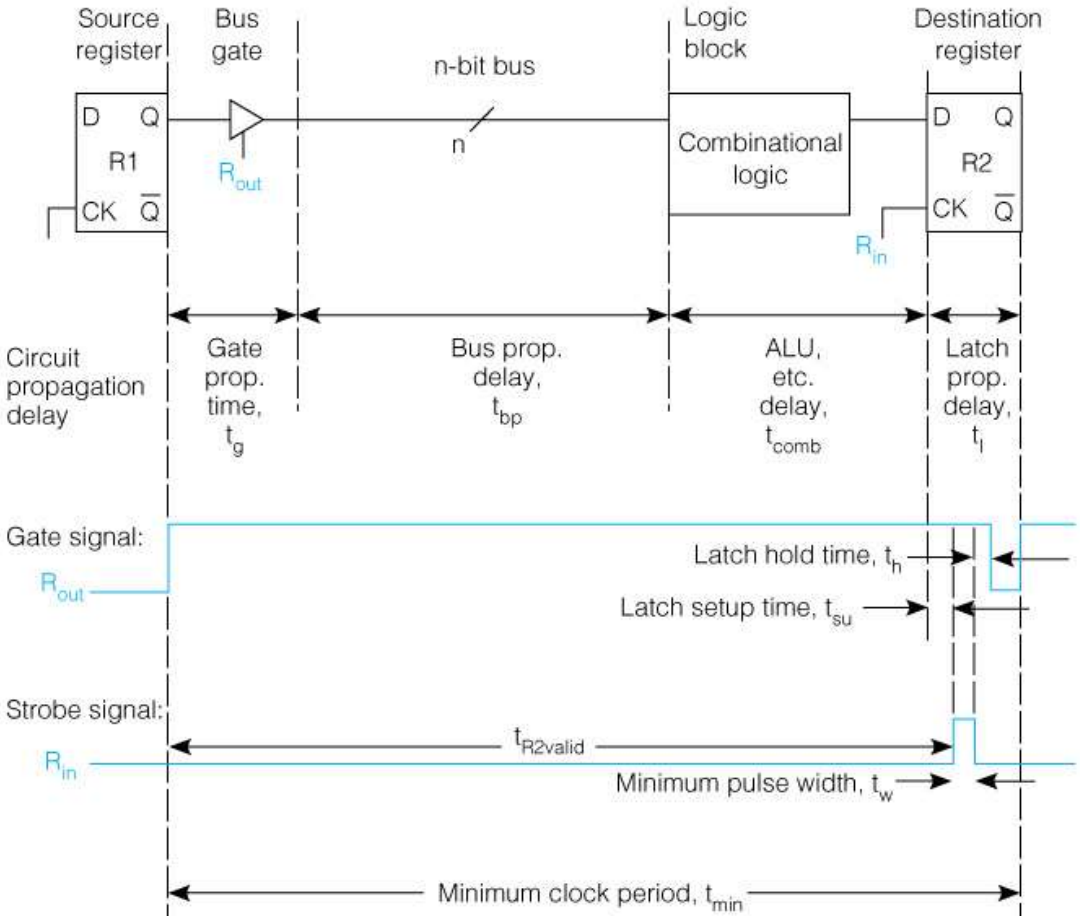
- Condition logic always connected to CON
  - R[rc] only needs to be placed on bus in T3
- Only PC_{in} is conditional in T4 since gating R[rb] to bus makes no difference if it is not used

Summary of Design Process

- Informal description ⇒ formal RTN description ⇒ block diagram arch. ⇒ concrete RTN steps ⇒ hardware design of blocks ⇒ control sequences ⇒ 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

Clocking the Datapath

- Register transfers result from information processing
  - Register transfer timing – register to register
- Level sensitive latch flip-flops in example
- $t_{R2valid}$ is the period from begin of gate signal till inputs at R2 are valid
- t_{comb} is delay through combinational logic, such as ALU or cond logic



Signal Timing on the Datapath

- 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

Signal Timing and 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 flip-flop output to flip-flop input
  - This is usually a path through the ALU
  - Conditional signals add a little gate delay
  - Minimum clock period is

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

Consequences of Flip-Flop Type

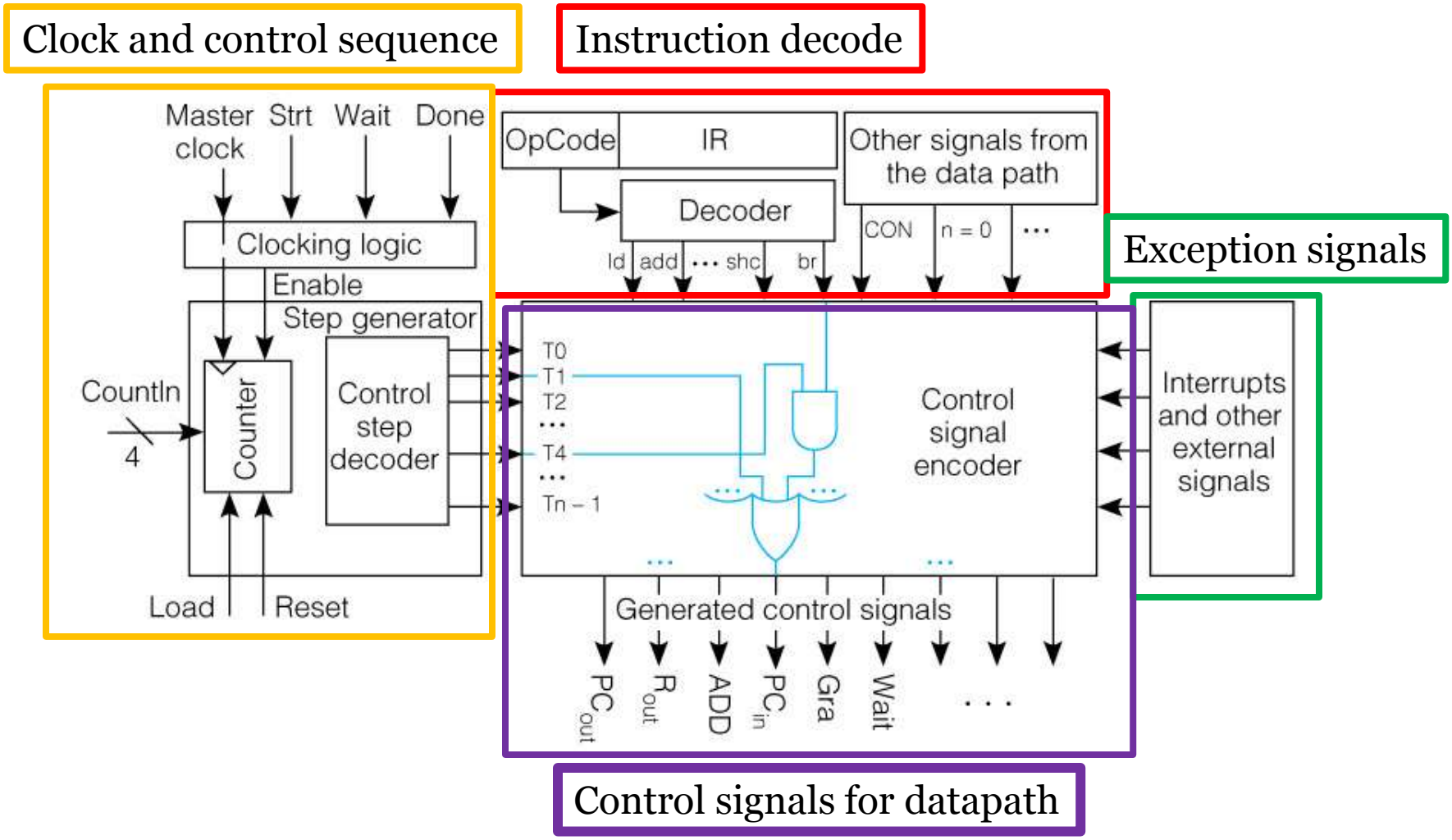
- Flip-flop types (Appendix A.12)
  - Level-triggered (latch) – state can change while clock is high
  - Edge-triggered – state changes only on a clock transition (high-to-low or low-to-high)
  - Master-slave – breaks feedback from output/input of register allowing on a single state change per clock cycle
- During the high part of a strobe a latch changes its output
  - If this output can affect its input, an error can occur (feeback)
- 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 ← 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

- Brain of a machine
- Datapath implementation led to control sequences to implement instructions
- Control unit will generate the control sequences
  - Logic to enable control signal
  - Timing of signals
- 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 datapath 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

Detailed Control Unit



Control Signal Encoder Logic

- Write equation describing control signal
  - Find all occurrences of control signal in entire set of control sequences
  - Equation implemented by digital logic gates

Step	Fetch Control Sequence	Step	ADD Control Sequence	Step	ADDI Control Sequence
T0	PC_{out}, MA_{in}, Inc4, C_{in}	T3	Grb, R_{out}, A_{in}	T3	Grb, R_{out}, A_{in}
T1	Read, C_{out}, PC_{in}, Wait	T4	Grc, R_{out}, ADD, C_{in}	T4	$c2_{out}$, ADD, C_{in}
T2	MD_{out}, IR_{in}	T5	C_{out}, Gra, R_{in}, End	T5	C_{out}, Gra, R_{in}, End

Step	SHR Control Sequence	Step	ST Control Sequence	Step	BR Control Sequence
T3	c1out, Ld	T3	Grb, BA_{out}, A_{in}	T3	Grc, R_{out}, CON_{in}
T4	n=0→(Grc, R_{out}, Ld)	T4	$c2_{out}$, ADD, C_{in}	T4	Grb, R_{out}, CON→PC_{in}, End
T5	Grb, R_{out}, C=B, C_{in}	T5	C_{out}, MA_{in}		
T6	n≠0→(C_{out}, SHR, C_{in}, Decr, Goto6)	T6	Gra, R_{out}, MD_{in}, Write		
T7	C_{out}, Gra, R_{in}, End	T7	Wait, End		

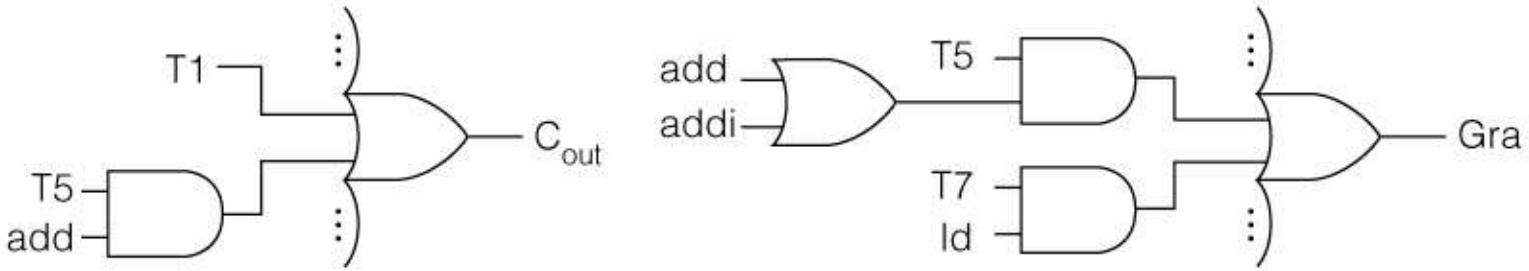
Control Signal Examples

Step	Fetch Control Sequence	Step	ADD Control Sequence	Step	ADDI Control Sequence
T0	PC_{out}, MA_{in}, Inc4, C_{in}	T3	Grb, R_{out}, A_{in}	T3	Grb, R_{out}, A_{in}
T1	Read, C_{out}, PC_{in}, Wait	T4	Grc, R_{out}, ADD, C_{in}	T4	$c2_{out}$, ADD, C_{in}
T2	MD_{out}, IR_{in}	T5	C_{out}, Gra, R_{in}, End	T5	C_{out}, Gra, R_{in}, End

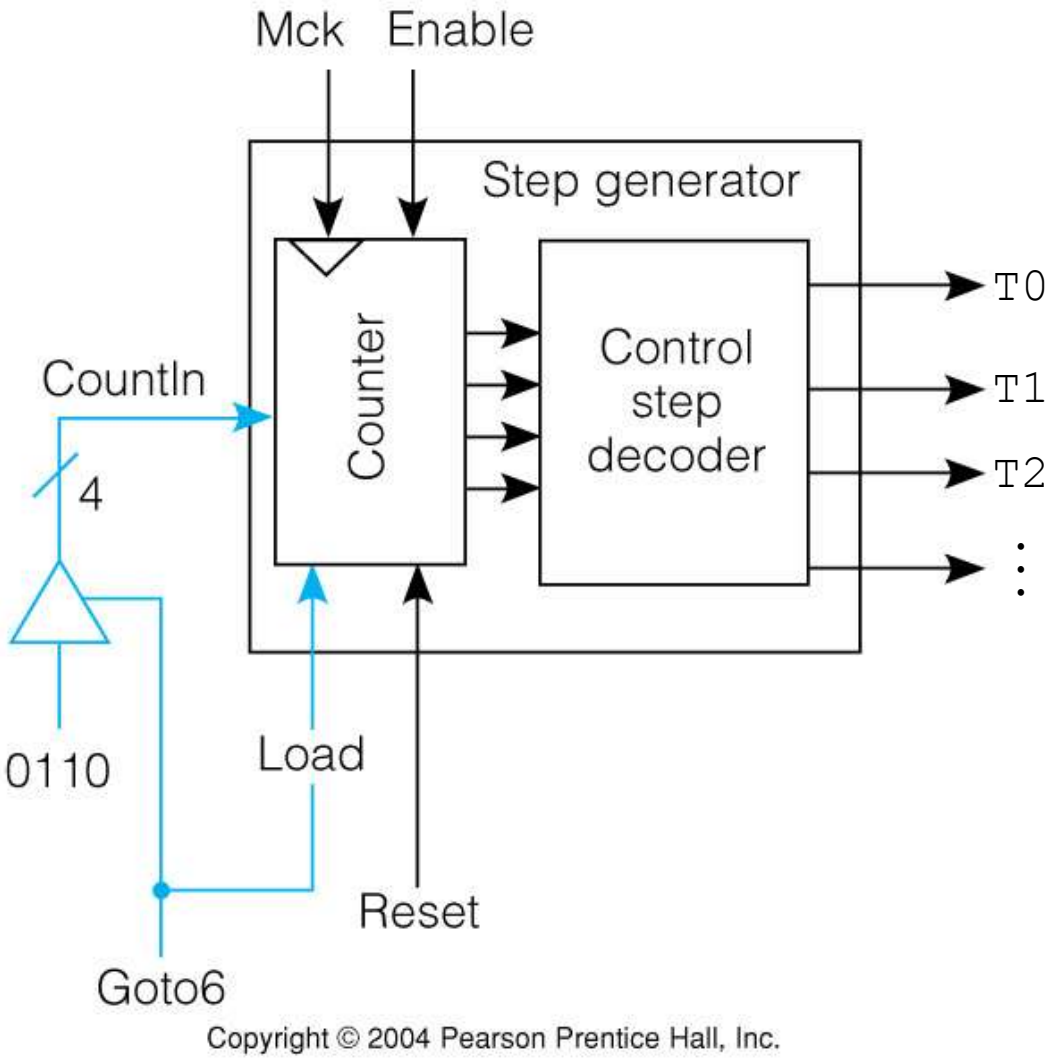
Step	SHR Control Sequence	Step	ST Control Sequence	Step	BR Control Sequence
T3	c1out, Ld	T3	Grb, BA_{out}, A_{in}	T3	Grc, R_{out}, CON_{in}
T4	n=0→(Grc, R_{out}, Ld)	T4	$c2_{out}$, ADD, C_{in}	T4	Grb, R_{out}, CON→PC_{in}, End
T5	Grb, R_{out}, C=B, C_{in}	T5	C_{out}, MA_{in}		
T6	n≠0→(C_{out}, SHR, C_{in}, Decr, Goto6)	T6	Gra, R_{out}, MD_{in}, Write		
T7	C_{out}, Gra, R_{in}, End	T7	Wait, End		

- Gra = T5·(add + addi) + T6·st +T7·shr + …

- Use of datapath conditions
  - Grc = T4·add + T4·(n=0)·shr + …



Branching in the Control Unit

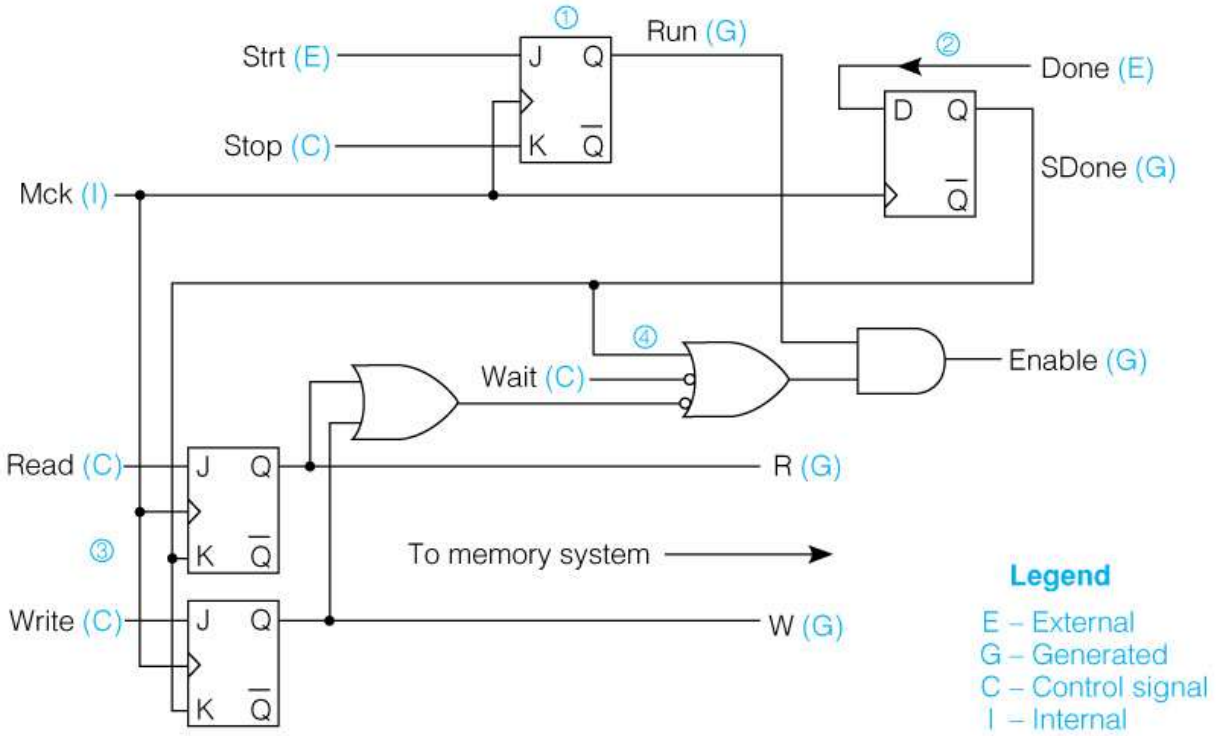


Copyright © 2004 Pearson Prentice Hall, Inc.

- Tri-state gates allow 6 to be applied to counter input
- Reset will synchronously reset counter to step To
- `Mck` is the master clock oscillator signal

Clocking Logic

- Generates `Run` signal
- Generate synchronized done signal `SDone`
- Generates `R`, `W` from `Read`, `Write` control
- Generates `Enable` which controls counter



Completed 1-Bus Design

- 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