CPE300: Digital System Architecture and Design

Fall 2011

MW 17:30-18:45 CBC C316

Control, Reset, Exceptions

10312011

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

Outline

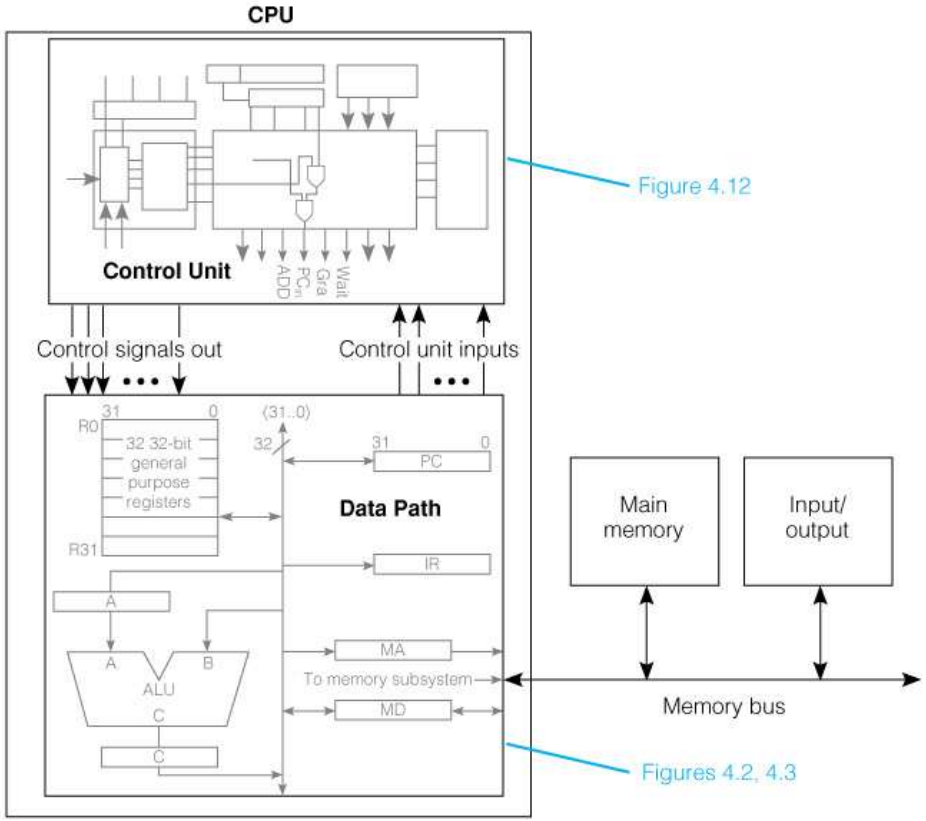
- Review Datapath/Control
- 2- and 3-Bus SRC Processor Design
- Machine Reset
- Machine Exceptions

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

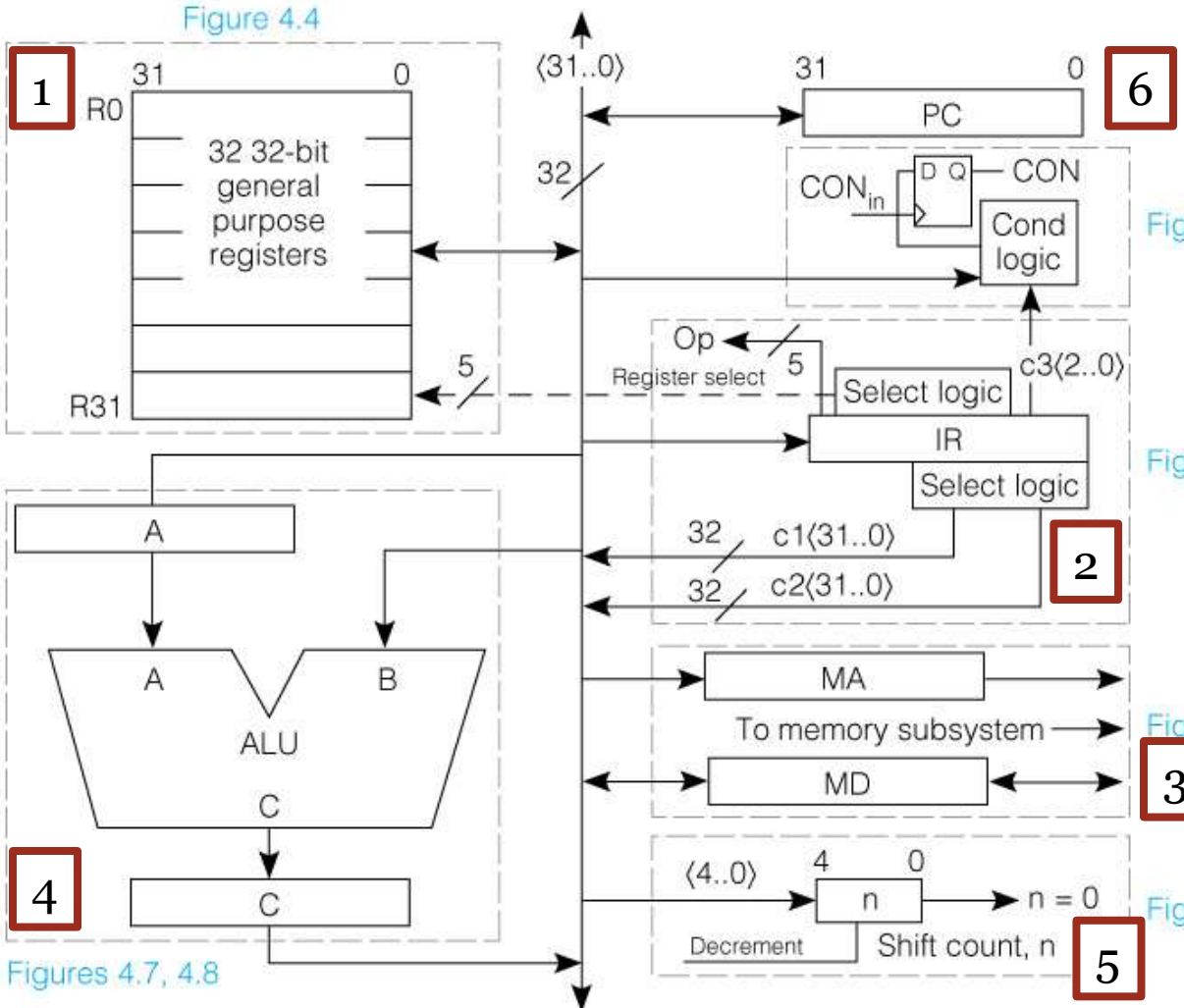
1-Bus SRC Microarchitecture

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



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

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	

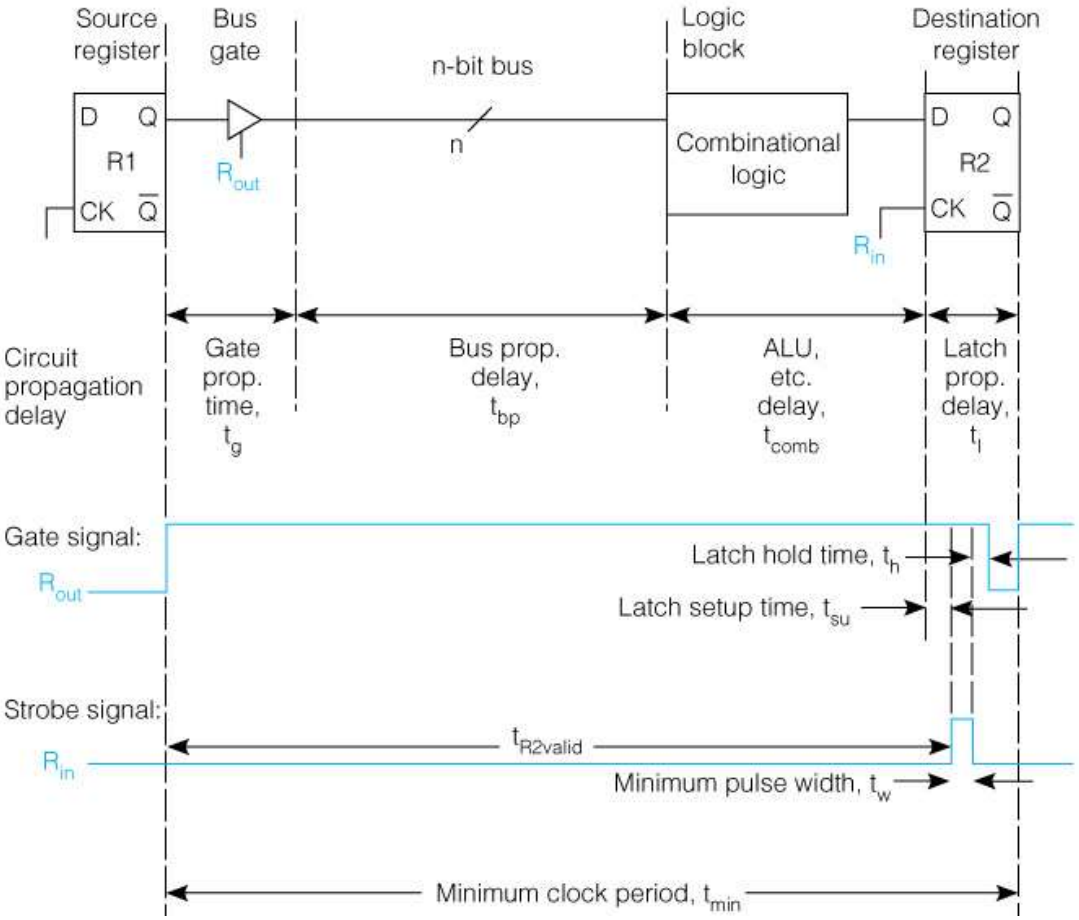
Wait prevents control sequence from advancing to step T2 until memory asserts Done

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

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

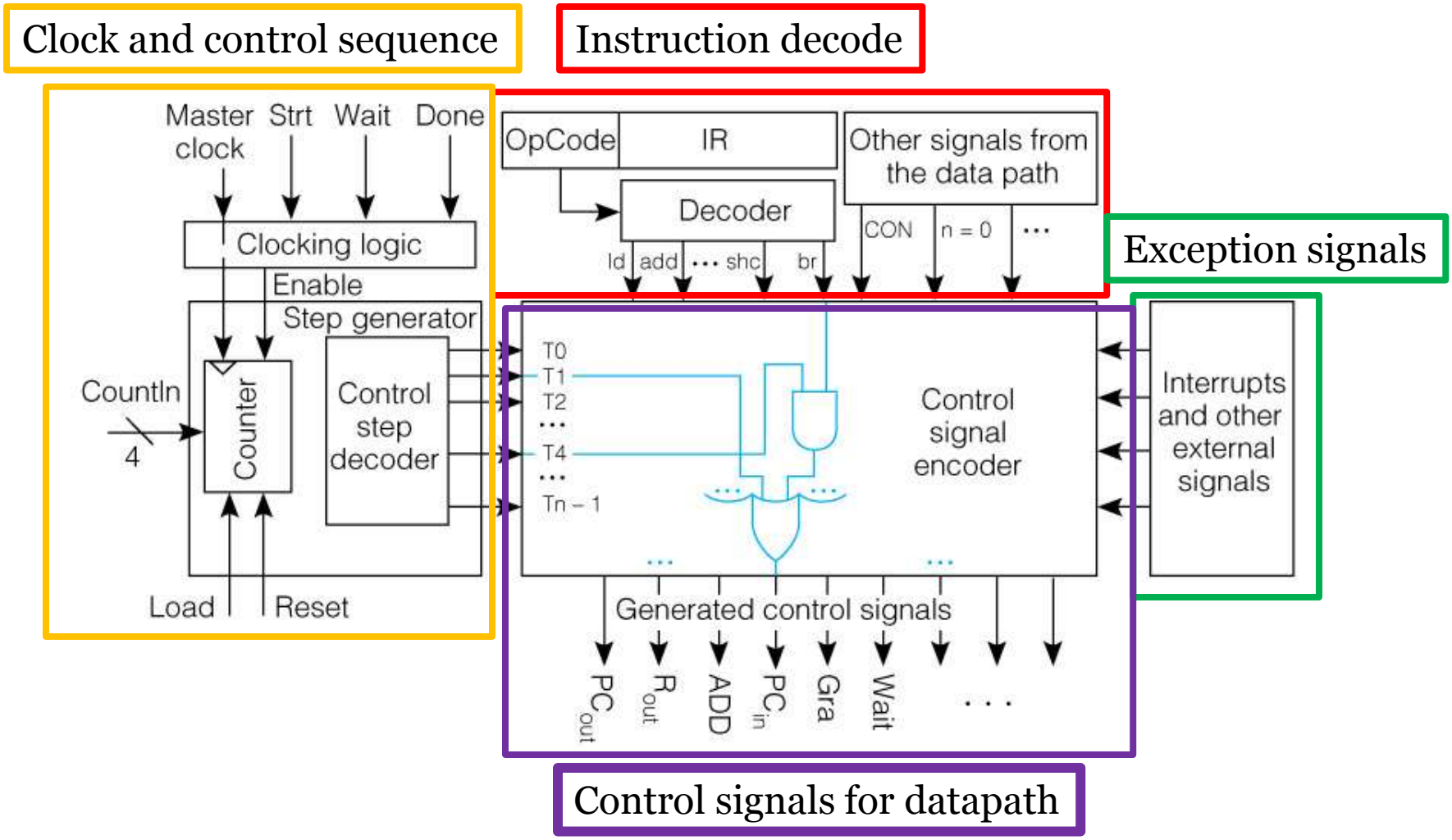
- Flip-flop types (Appendix A.12)
  - Level-triggered (latch) – state can change while clock is high
  - Edge-triggered – state changes only on a clock transition (high-to-low or low-to-high)
  - Master-slave – breaks feedback from output/input of register allowing on a single state change per clock 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		

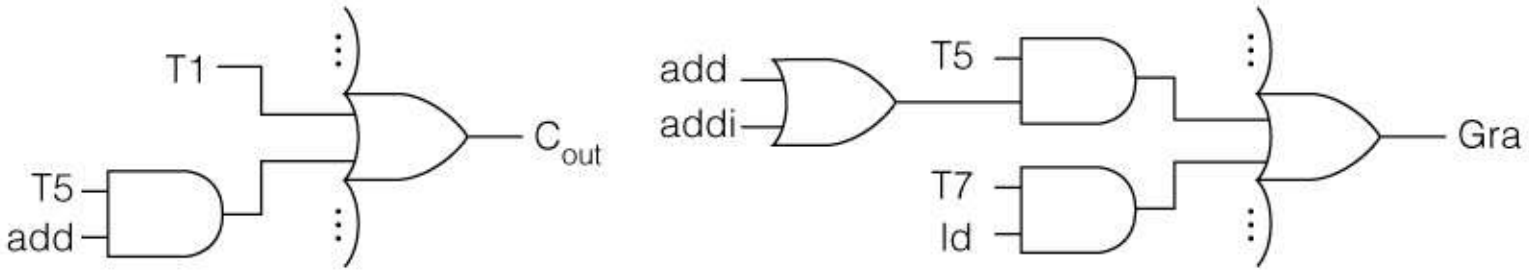
Control Signal Examples

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

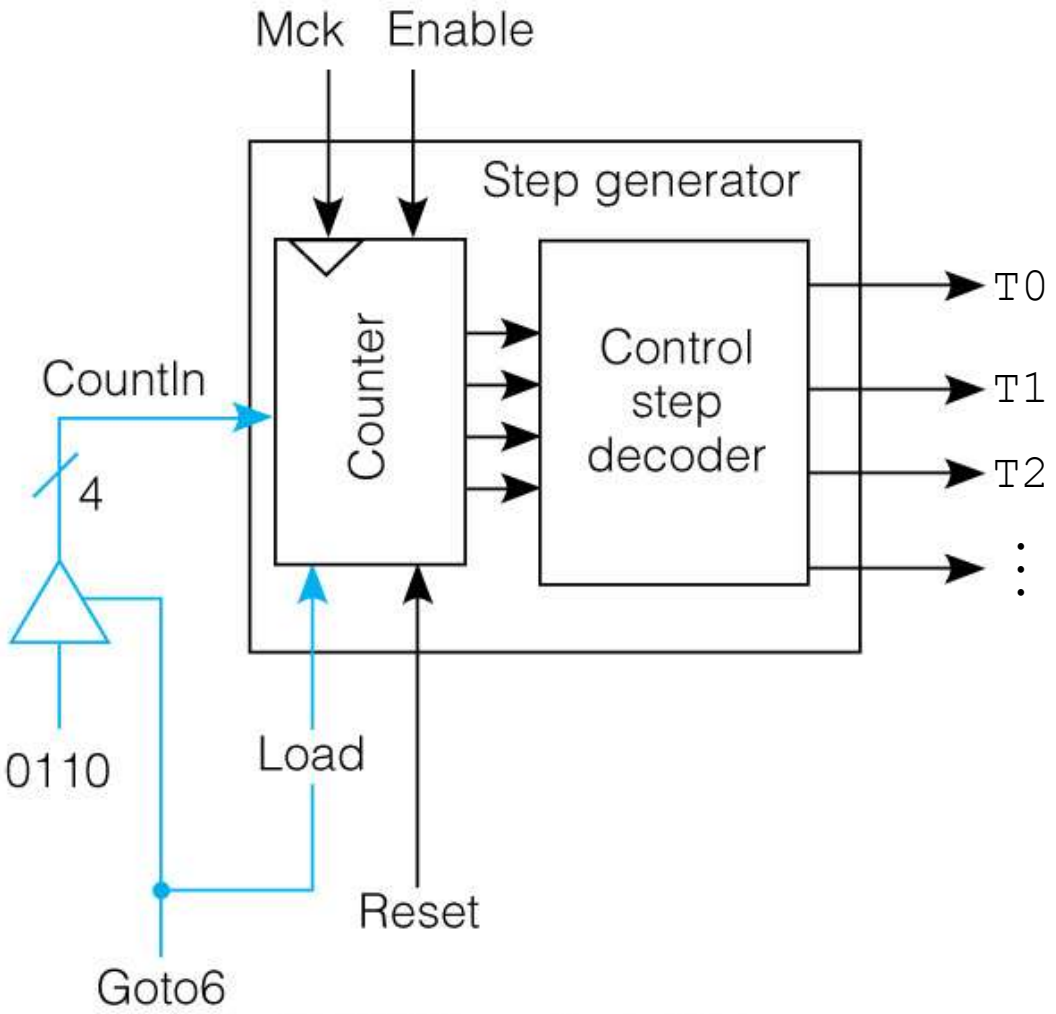
Step	SHR Control Sequence	Step	ST Control Sequence
T3	c1out, Ld	T3	Grb, BA_{out}, A_{in}
T4	n=0→(Grc, R_{out}, Ld)	T4	$c2_{out}$, ADD, C_{in}
T5	Grb, R_{out}, C=B, C_{in}	T5	C_{out}, MA_{in}
T6	n≠0→(C_{out}, SHR, C_{in}, Decr, Goto6)	T6	Gra, R_{out}, MD_{in}, Write
T7	C_{out}, Gra, R_{in}, End	T7	Wait,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

Step	BR Control Sequence
T3	Grc, R_{out}, CON_{in}
T4	Grb, R_{out}, CON→PC_{in}, 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

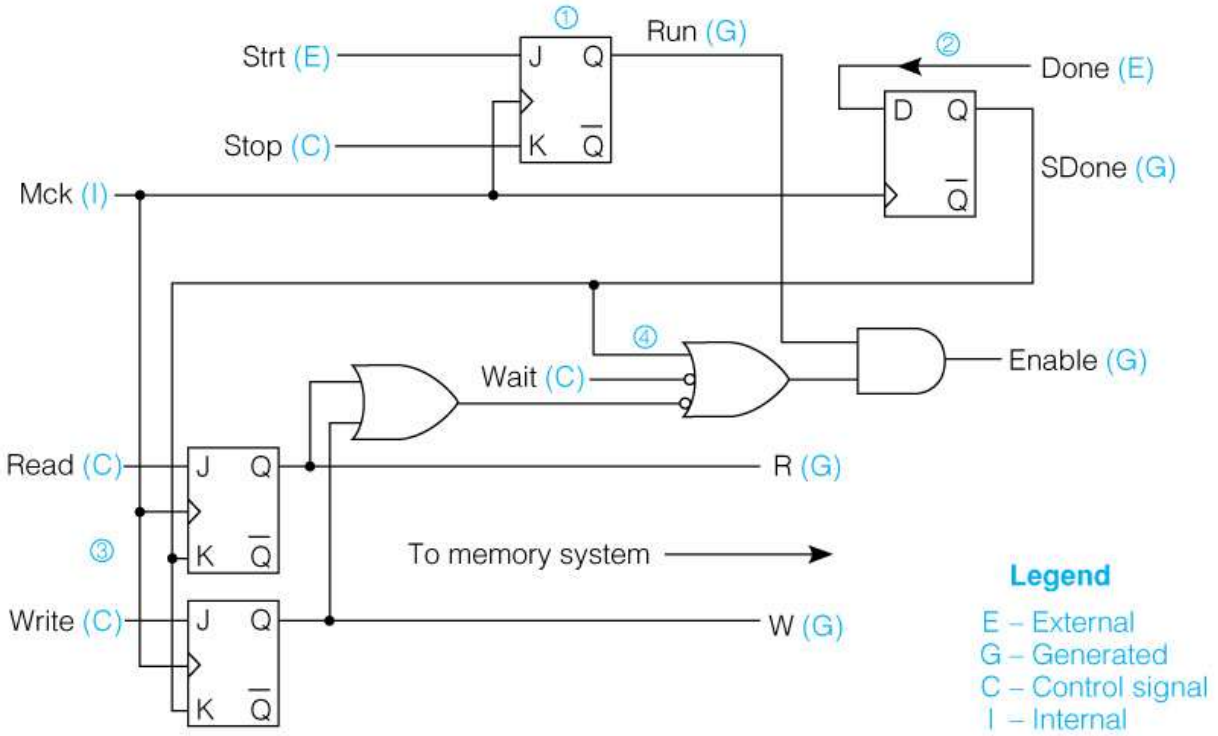


Copyright © 2004 Pearson Prentice Hall, Inc.

- 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



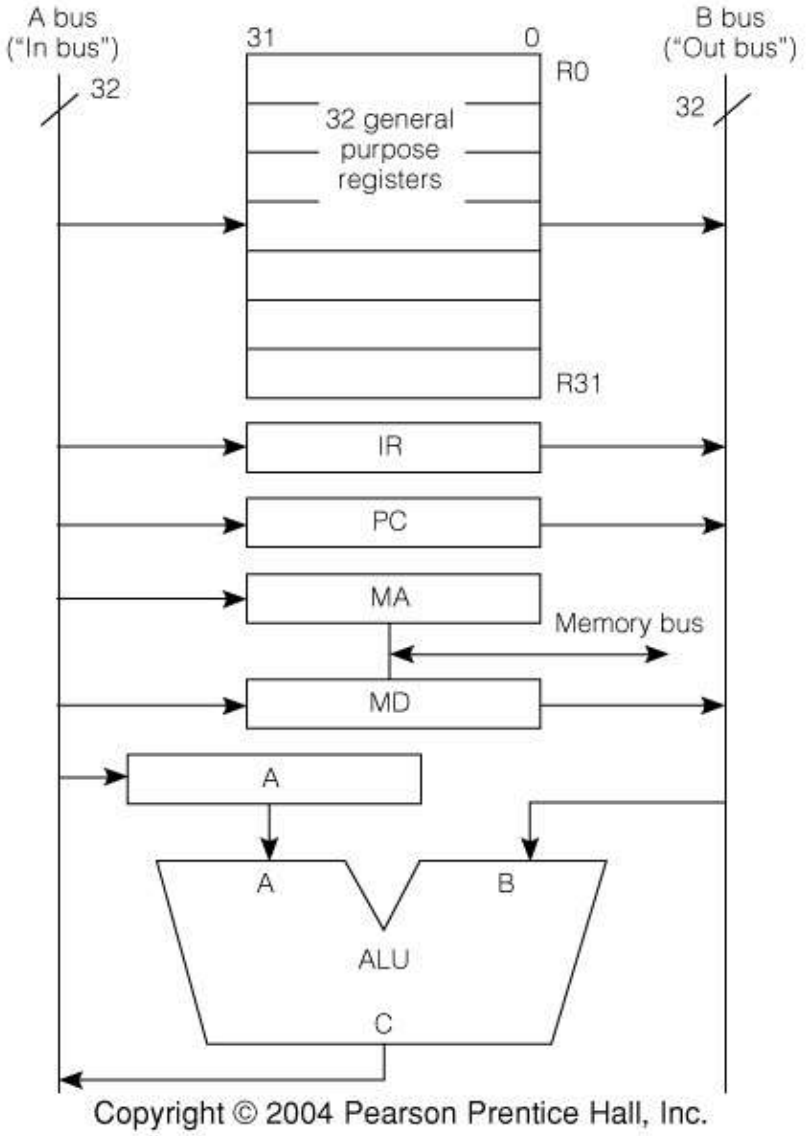
Completed 1-Bus Design

- 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

Alternate Architectural Design

- Require different RTN than 1-bus design
- More datapaths allow more things to be done in a single step
- 2-bus example that separates input and output of ALU on different buses
  - C register can be eliminated
  - Control steps can be reduced by strobing ALU results directly into their destinations

2-Bus SRC Microarchitecture



Copyright © 2004 Pearson Prentice Hall, Inc.

- A bus carries data going into registers
- B bus carries data being gated out of registers
- ALU function C=B is used for all simple register transfers
  - `R[a]←R[b]`
- Allows increment transfers
  - `R[n]←R[m]+1`

2-Bus Control for ADD Instruction

- add(:=op=12)→R[ra]←R[rb]+R[rc]:

Step	Concrete RTN	Control Sequence
T0	MA←PC;	PC_{out}, C=B, MA_{in},
T1	MD←M[MA]: PC←PC+4	Read, Wait, PC_{out}, INC4, PC_{in}
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 To

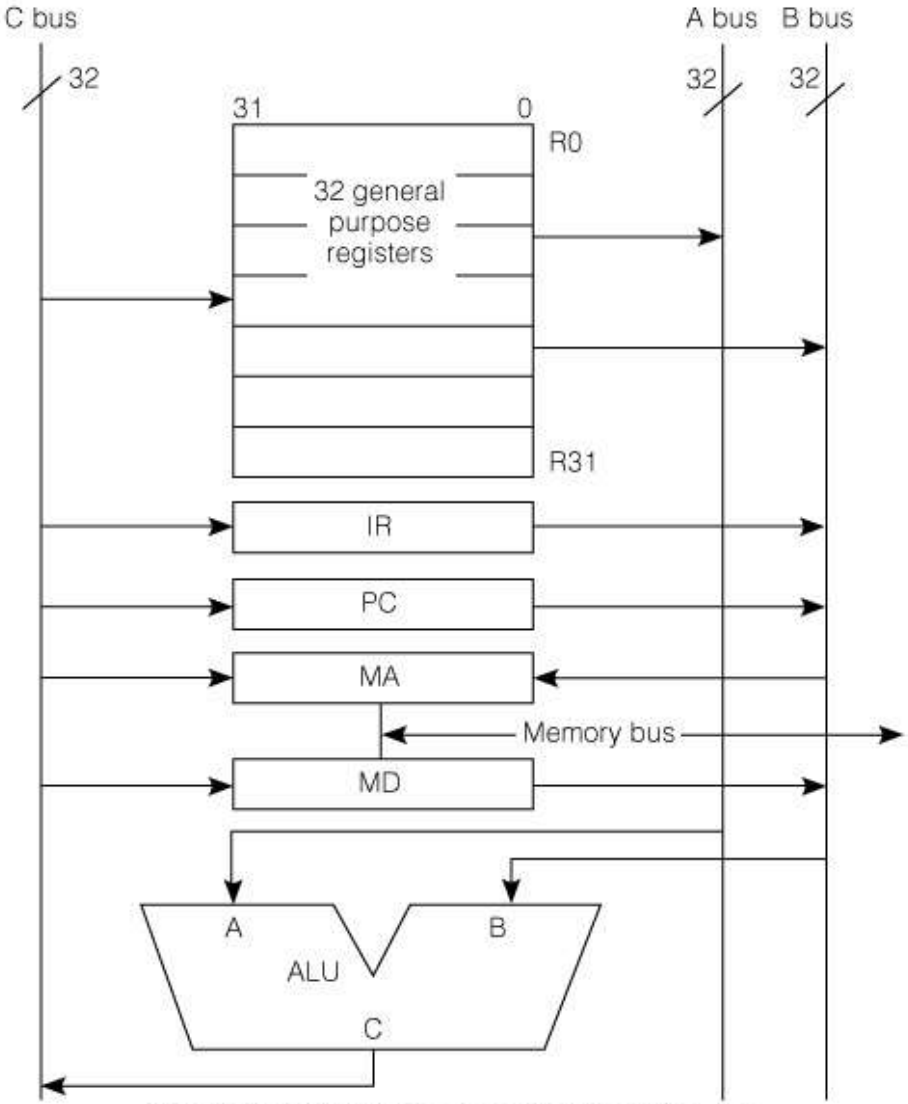
2-Bus Performance

- Speedup $= \frac{T_1}{T_2}$
- $T_i =$ execution time $= IC \times CPI \times \tau$
  - i is 1- or 2-bus
- Assumptions
  - IC and t don't change in going from 1 bus to 2 buses
  - CPI goes from 8 to 7 clocks (naïve assumption)
- Speedup $= \frac{T_1}{T_2} = \frac{IC \times 8 \times \tau}{IC \times 7 \times \tau} = \frac{8}{7} = 1.143 = 14.3\%$
- What happens if clock also changes?

3-Bus Design

- A 3-bus architecture allows both operand inputs and the output of the ALU to be connected to buses
  - Shortens control sequences even further than 2-bus design
- 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

3-Bus SRC Microarchitecture



Copyright © 2004 Pearson Prentice Hall, Inc.

- A-bus is ALU operand 1
- B-bus is ALU operand 2
- C-bus is ALU output
- Note MA input connected to the B-bus
- Allows operations such as `R[n]←R[m]+R[k]` to complete in one cycle.
- What are cost implications?

3-Bus Control for ADD Instruction

- add(:=op=12)→R[ra]←R[rb]+R[rc]:

Step	Concrete RTN	Control Sequence
T0	MA←PC: MD←M[MA]: PC←PC+4	PC_{out}, MAB_{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 To, 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

3-Bus Performance

- Speedup $= \frac{T_1}{T_3}$
- T_i = execution time $= IC \times CPI \times \tau$
  - i is 1- or 2-bus
- Assumptions
  - IC and t don't change in going from 1 bus to 3 buses
  - CPI goes from 8 to 4 clocks (naïve assumption)
  - τ increases by 10%
- Speedup $= \frac{T_1}{T_3} = \frac{IC \times 8 \times \tau}{IC \times 4 \times 1.1\tau} = \frac{8}{4.4} = 1.818 = 81.8\%$

Machine Reset

- Reset sets program counter to a fixed value
  - May be a hardwired value
  - 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

- Both hard and soft reset specified
- `Strt` signal will do a hard reset
  - Effective only when machine is stopped
  - Resets the PC to zero
  - Resets all 32 general registers to zero
- Soft `Rst` signal is effective when the machine is running
  - Resets PC to zero
  - Restarts instruction fetch
  - Clears the Reset signal
- Actions on reset are described in `instruction_interpretation`

Abstract RTN for SRC Reset and Start

- 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):`

Resets 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

- Why might we want to reset in the middle of an instruction?
  - Long instructions
- How would we reset in the middle of an instruction?
  - Check for Rst at each control time step

Concrete RTN and Control with Reset

Step	Concrete RTN	Control Sequence
T0	¬Rst→(MA←PC: C←PC+4): Rst→(Rst←0:PC←0: T←0);	¬Rst→(PC_{out}, MA_{in}, Inc4, C_{in}) Rst→(ClrPC, ClrR, Goto0);
T1	¬Rst→(MD←M[MA]: PC←C): Rst→(Rst←0:PC←0: T←0);	¬Rst→(Read, C_{out}, PC_{in}, Wait) Rst→(ClrPC, ClrR, Goto0);
T2	¬Rst→(IR←MD): Rst→(Rst←0:PC←0: T←0);	¬Rst→(MD_{out}, IR_{in}) Rst→(ClrPC, ClrR, Goto0);
T3	¬Rst→(A←R[rb]): Rst→(Rst←0:PC←0: T←0);	¬Rst→(Grb, R_{out}, A_{in}) Rst→(ClrPC, ClrR, Goto0);
T4	¬Rst→(C←A+R[rc]): Rst→(Rst←0:PC←0: T←0);	¬Rst→(Grc, R_{out}, ADD, C_{in}) Rst→(ClrPC, ClrR, Goto0);
T5	¬Rst→(R[ra]←C): Rst→(Rst←0:PC←0: T←0);	¬Rst→(C_{out}, Gra, R_{in}, End) Rst→(ClrPC, ClrR, Goto0);

- Same RTN/control as before but must check `Rst`
  - Reset actions are the same for every step of every instruction → control signals are independent of time step or opcode
- `ClrPC` clears the program counter to all zeros
- `ClrR` clears the one bit Reset flip-flop

Machine Exceptions

- An exception is an event that causes a change in the program specified flow of control
  - Internal are usually synchronous (overflow)
  - External often asynchronous (keyboard)
- Often called interrupts
  - Normal program execution is interrupted
- No standard naming conventions
  - Exception for general term
  - Interrupt for an exception caused by an external event, such as an I/O device condition

Hardware/Software Exception Response

- 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 Support of Exceptions

- To determine relative importance, a priority number is associated with every exception
- Hardware must save and change the PC
  - Required for program execution
- Hardware must disable the current exception
  - Could 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

Instruction Support of 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

Types of Exceptions

- System reset
- Exceptions associated with memory access
  - Machine check – memory failure
  - Data access – memory not available
  - Instruction access – instruction not available (similar to data access)
  - Alignment – improperly aligned access
- Program exceptions
  - Illegal instruction – instruction not in IS
  - Unimplemented instruction – legal but not in IS
  - Privileged instructions – instruction not available
  - Arithmetic errors
- Miscellaneous hardware exceptions – (e.g. watchdog)
- Trace and debugging exceptions
- Non-maskable exceptions (NMI) – very bad cases
- External exceptions—interrupts

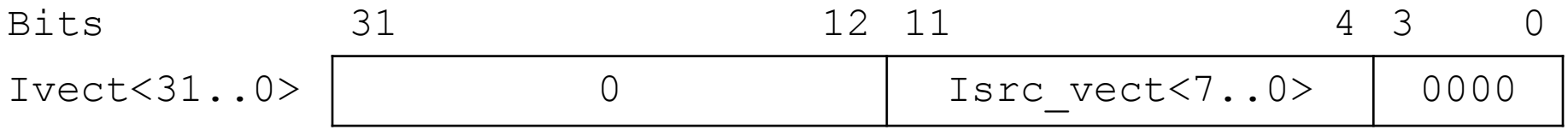
SRC Exception Processing

- The exception mechanism for SRC handles external interrupts
- There are no priorities
  - 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

SRC Interrupt Processor State

- 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<31..0>:` ;storage for PC saved upon interrupt
to CPU → `II<15..0>:` ;info. on source of last interrupt
From Dev. → `Isrc_info<15..0>:` ;information from interrupt source
From Dev → `Isrc_vect<7..0>:` ;type code from interrupt source
Internal → `Ivect<31..0>:= 20@0#Isrc_vect<7..0>#4@0:`



SRC Instruction Interpretation with 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 Instruction to Support Interrupts

- Return from interrupt
  - `rfi(:=op=29)→(PC←IPC: IE←1):`
  - Two RT actions must occur together
    - Cannot be accomplished with branch and `ee` instruction combination
- Save and restore interrupt state
  - `svi(:=op=16)→(R[ra]<15..0>←II<15..0>: R[rb]←IPC<31..0>);`
  - `rvi(:=op=17)→(II<15..0>← R[ra]<15..0> : IPC<31..0>← R[rb]):`
- Enable/disable interrupt system
  - `een(:=op=10)→(IE←1)`
  - `edn(:=op=11)→(IE←0)`

Concrete RTN with Interrupt

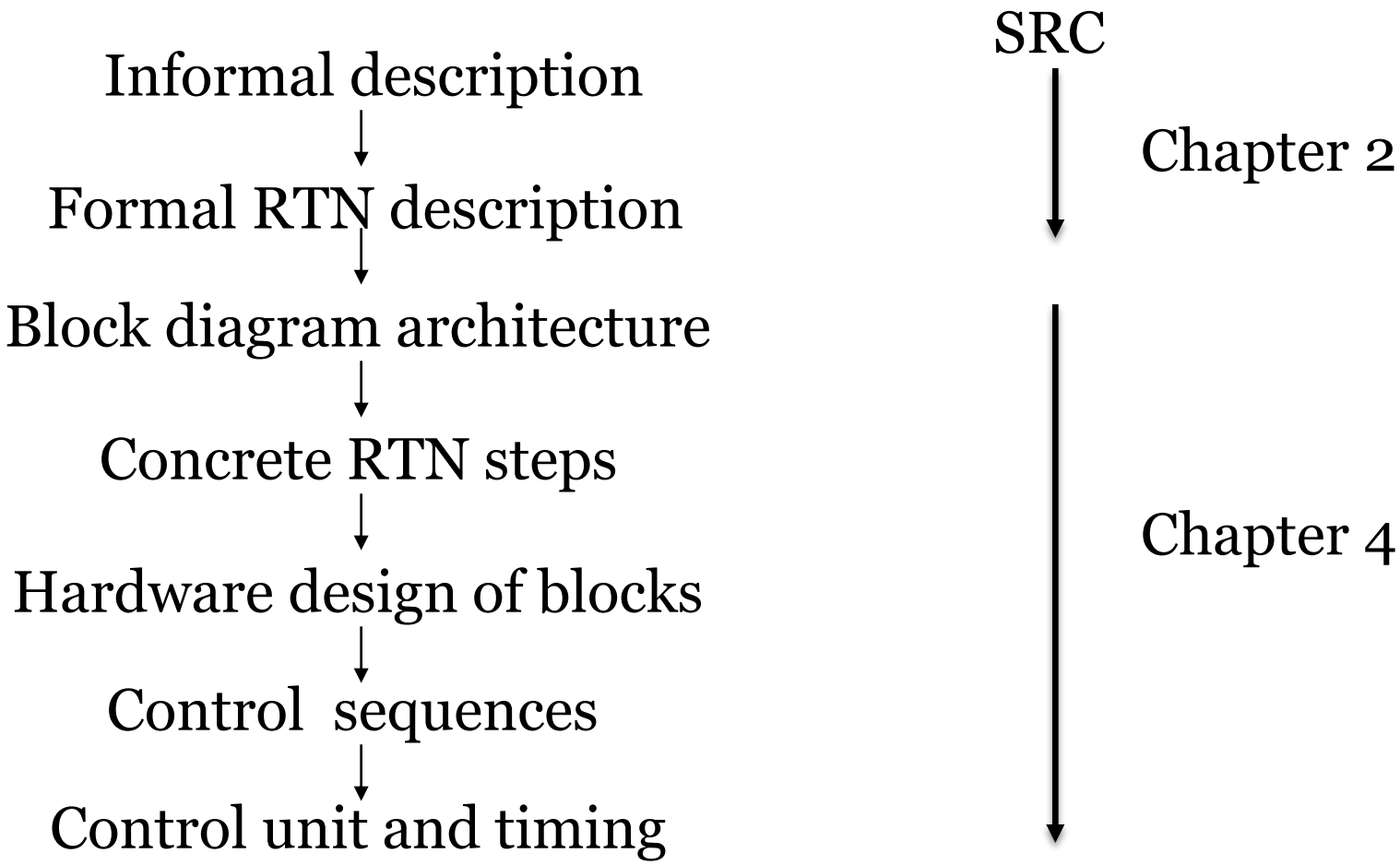
Step	Concrete RTN	
	¬(ireq∧IE)	(ireq∧IE)
T0	¬(ireq∧IE)→(MA←PC: C←PC+4):	(ireq∧IE)→(IPC←PC: II←Isrc_info: IE←0: PC←20@#Isrc_vect<7..0>#00: iack←1; iack←0; End);
T1	MD←M[MA]: PC←C;	
T2	IR←MD;	
T3	instruction_execution	

- 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 (string move)
  - Some exception conditions prevent instruction completion (uninstalled memory)
- 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 Design Process



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