CPE300: Digital System Architecture and Design

Fall 2011
MW 17:30-18:45 CBC C316

Register Transfer Notation
09192011

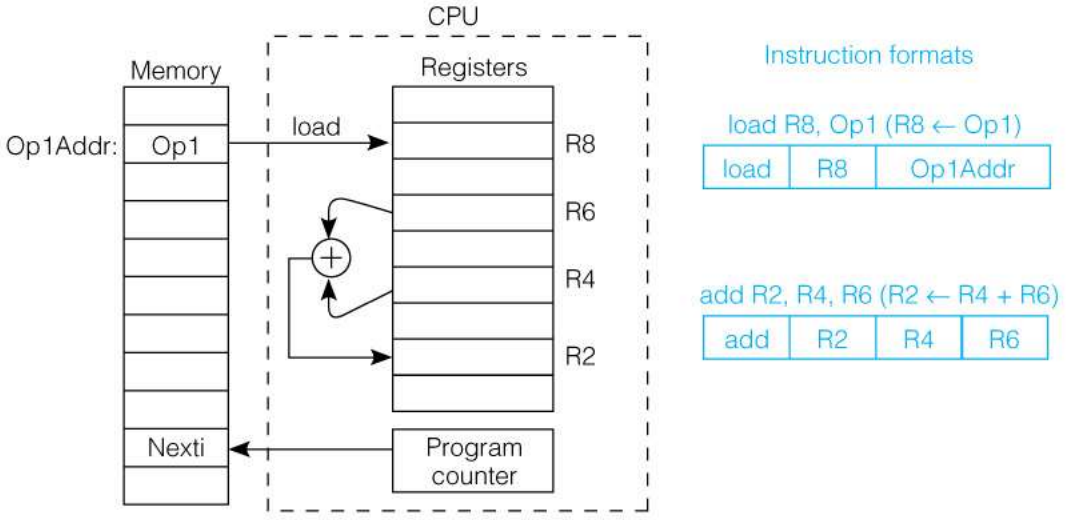
http://www.egr.unlv.edu/~b1morris/cpe300/

Outline

- Recap
- Register Transfer Notation (RTN)
- Logic Circuits for Register Transfer

General Register Machines

- Most common choice for general purpose computers
  - Load-store machines
- Registers specified by “small” address
  - Close to CPU for speed and reuse for complex operations



Instructions/Register Trade-Offs

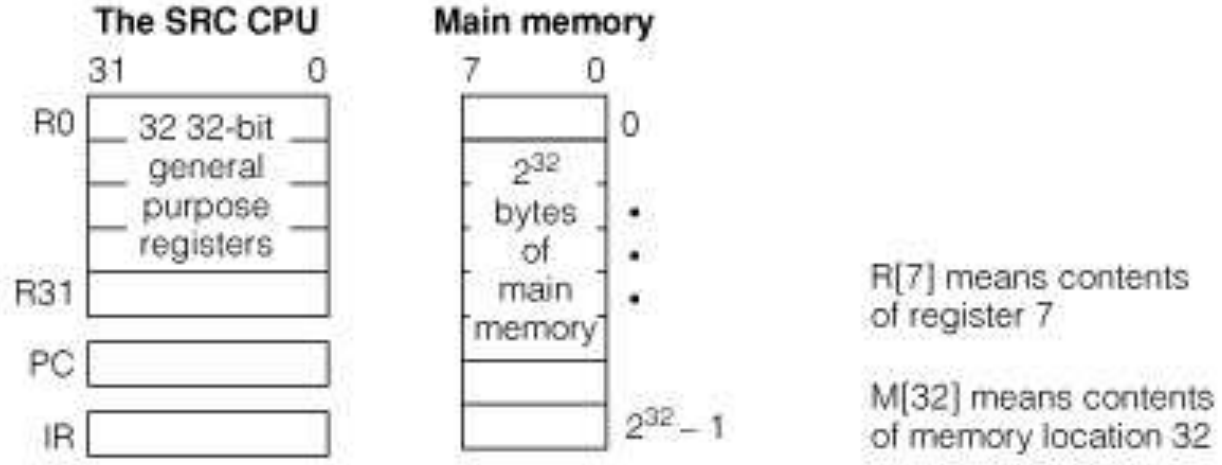
- 3-address machines have shortest code but large number of bits per instruction
- 0-address machines have longest code but small number of bits per instruction
  - Still require 1-address (push, pop) instructions
- General register machines use short internal register addresses in place of long memory addresses
- Load-store machines only allow memory addresses in data movement instructions (load, store)
- Register access is much faster than memory access
- Short instructions are faster

Addressing Modes

- Hardware support for determining access paths to operands (in memory or registers)
  - Some addresses may be known at compile time, e.g. global vars.
  - Others may not be known until run time, e.g. pointers
  - Addresses may have to be computed
    - Record (struct) components:
      - variable base(full address) + const.(small)
    - Array components:
      - const. base(full address) + index var.(small)

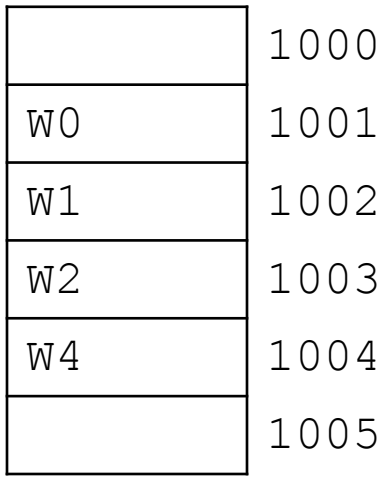
Simple RISC Computer (SRC)

- 32 general purpose registers (32 bits wide)
- 32 bit program counter (PC) and instruction register (IR)
- 2^{32} bytes of memory address space
- Use C-style array referencing for addresses



SRC Memory

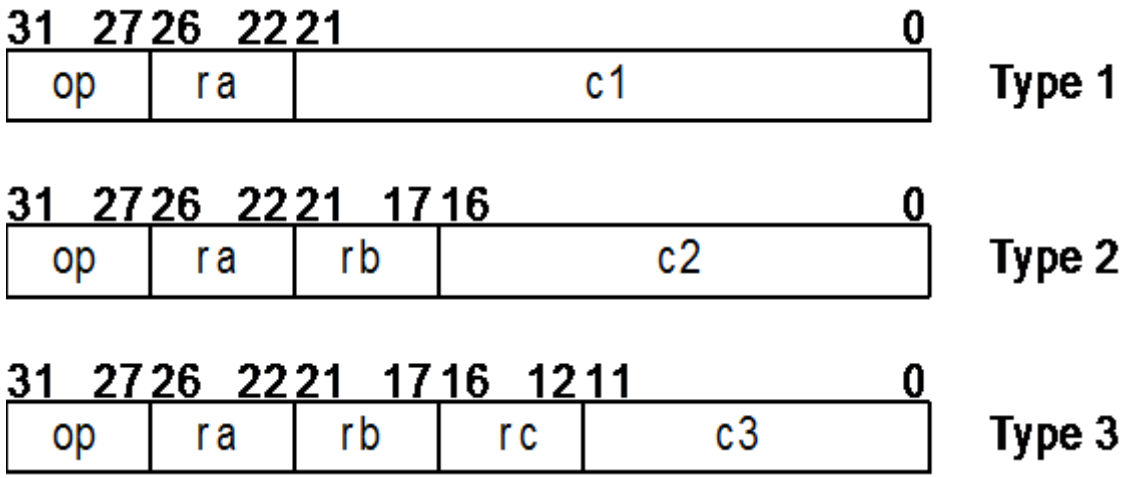
- 2^{32} bytes of memory address space
- Access is 32 bit words
  - 4 bytes make up word, requires 4 addresses
  - Lower address contains most significant bits (msb) – highest least significant bits (lsb)



Bits	31	23	15	7	0
Address	1001	1002	1003	1004	
Value	W0	W1	W2	W3	

SRC Basic Instruction Formats

- There are three basic instruction format types
- The number of register specific fields and length of the constant field vary
- Other formats result from unused fields or parts



SRC Assembly Language

- Full Instruction listing available in Appendix B.5
- Form of line of SRC assembly code

```
Label:      opcode      operands                ;comments
```

- `Label:` = assembly defined symbol
  - Could be constant, label, etc. – very useful but not always present
- `Opcode` = machine instruction or pseudo-op
- `Operands` = registers and constants
  - Comma separated
  - Values assumed to be decimal unless indicated (B, 0x)

Register Transfer Notation (RTN)

- Provides a formal means of describing machine structure and function
  - Mix natural language and mathematical expressions
- Does not replace hardware description languages.
  - Formal description and design of electronic circuits (digital logic) – operation, organization, etc.
- Abstract RTN
  - Describes what a machine does without the how
  - Overall effect on visible registers (ignores temporary)
- Concrete RTN
  - Describe a particular hardware implementation (how it is done)
  - Detailed register transfer, specified by clock cycle
- Meta-language = language to describe machine language

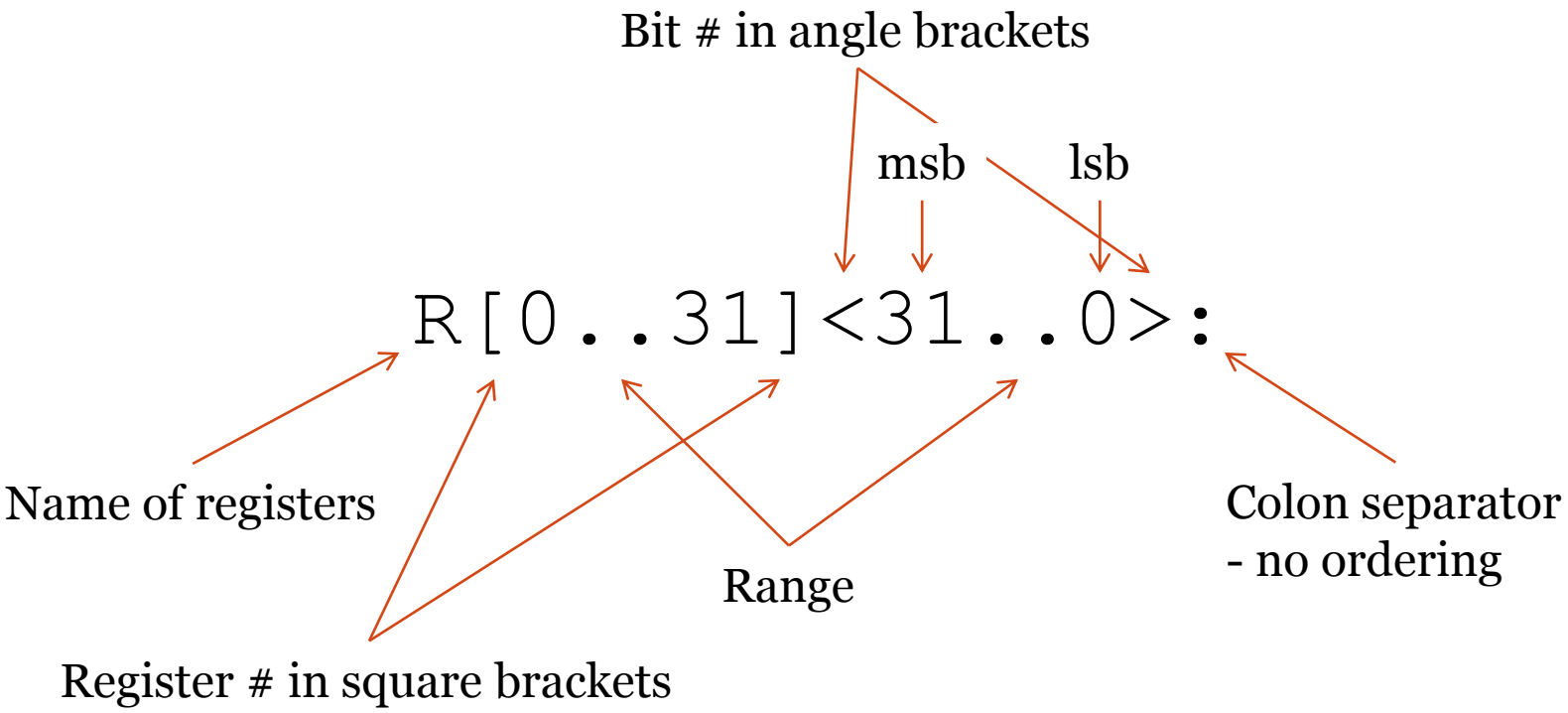
RTN Symbol Definitions (Appendix B.4)

←	Register transfer: register on LHS stores value from RHS
[]	Word index: selects word or range from named memory
<>	Bit index: selects bit or bit range from named memory
n..m	Index range: from left index n to right index m; can be decreasing
→	If-then: true condition of left yields value and/or action on right
:=	Definition: text substitution with dummy variables
#	Concatenation: bits on right appended to bits on left
:	Parallel separator: actions or evaluations carried out simultaneously
;	Sequential separator: RHS evaluated and/or performed after LHS
@	Replication: LHS repetitions of RHS are concatenated
{}	Operation modifier: information about preceding operation, e.g., arithmetic type
()	Operation or value grouping
= ≠ < ≤ ≥ >	Comparison operators: produce binary logical values
+ - ÷ ×	Arithmetic operators
∧ ∨ ¬ ⊕ ≡	Logical operators: and, or, not, xor, equivalence

Machine Static Properties

- Processor state items
  - `IR<31..0>`
    - 32 bit register named IR
  - `R[0..31]<31..0>`
    - 32 32-bit general purpose registers
- Create alias (:=)
  - `op<4..0> := IR<31..27>`
  - 5 most significant bits of IR are is called (defined) as op
  - Does not create new register

RTN Register Declaration



RTN Memory Declaration

- Define word memory (big endian)
- Main memory state

$Mem[0..2^{32} - 1]\langle 7..0\rangle$: 2^{32} addressable bytes of memory

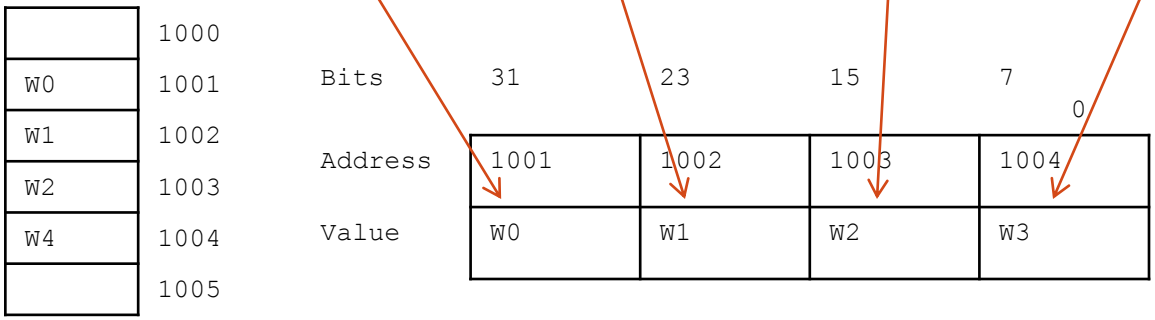
Dummy parameter

Naming operator

Concatenation operator

All bits in register if no bit index given

$M[x]\langle 31..0\rangle := Mem[x]\#Mem[x+1]\#Mem[x+2]\#Mem[x+3]$:



Machine Dynamic Properties

- Calculated at run-time
- If-then conditions
- Displacement address

```
disp<31..0> := ((rb=0) → c2<16..0> {sign extend}:
               (rb≠0) → R[rb] + c2<16..0> {sign extend, 2's comp.}):
```

  - if `(rb=0)` and if `(rb≠0)` occur at same time (:)
    – no else statement
  - Register `R[0]` used in calculation
- Relative address

```
rel<31..0> := PC<31..0> + c1<21..0> {sign extend, 2's complement}:
```

Range of Addresses

- Direct addressing (`rb=0`)
  - `c2<16..0>=0` (positive displacement)
    - `0x00000000 – 0x0000FFFF`
  - `c2<16..0>=1` (negative displacement)
    - `0xFFFF0000 – 0xFFFFFFFF`
- Relative addressing (`c1<21..0>`)
  - Max = $2^{21}-1$
  - Min = -2^{21}
  - $-2^{21} + PC - PC + 2^{21}-1$
- Note the difference between rb and R[rb]

RTN Fetch-Execute Cycle

- ii := instruction_interpretation:
- ie := instruction_execution :

- ii := (
  ¬Run∧Strt → Run ← 1:
  Run → (IR ← M[PC]: PC ← PC + 4;
  ie));
- ie := (
  ld (:= op= 1) → R[ra] ← M[disp]:
  ldr (:= op= 2) → R[ra] ← M[rel]:
  . . .
  stop (:= op= 31) → Run ← 0:
  ); ii

Big switch statement on opcode

- Thus ii and ie invoke each other, as co-routines

RTN Described Addressing Modes

Common Name	Assembler Syntax	Meaning	Typical Usage
Register	Ra	R[t] ← R[a]	Temporary variable
Register indirect	(Ra)	R[t] ← M[R[a]]	Pointers to structures
Immediate	#x	R[t] ← x	Constant operand
Direct, absolute	x	R[t] ← M[x]	Global variable
Indirect	(x)	R[t] ←M[M[x]]	Accessing value through its pointer
Indexed, based, displacement	x(Ra)	R[t] ← M[x + R[a]]	Arrays and structures
Relative	x(PC)	R[t] ← M[x + PC]	Instructions or values stored in program
Autoincrement	(Ra)+	R[t] ← M[R[a]]; R[a] ← R[a] + 1;	Sequential access or stack pop
Autodecrement	-(Ra)	R[a] ← R[a]-1; R[t] ← M[R[a]];	Sequential access or stack push

Addressing Mode Example 2.4

- Give contents of register R1 for different addressing modes

Machine State			
Registers		Memory	
PC	4000	Addr	Data
R2	3000	1000	2000
		2000	3000
		3000	4000
		4000	5000
		5000	6000

Addressing Mode	Instruction	Contents of R1
Immediate	`MOV R1, #1000`	
Direct	`MOV R1, 1000`	
Indirect	`MOV R1, (1000)`	
Register Indirect	`MOV R1, (R2)`	
Indexed	`MOV R1, 1000(R2)`	
Relative	`MOV R1, 1000(PC)`	

Example 2.4

Addressing Mode	Instruction	Meaning	Contents of R1
Immediate	MOV R1, #1000	R1 ← 1000	1000
Direct	MOV R1, 1000	R1 ← M[1000]	2000
Indirect	MOV R1, (1000)	R1 ← M[M[1000]]	3000
Register Indirect	MOV R1, (R2)	R1 ← M[R[2]]	4000
Indexed	MOV R1, 1000(R2)	R1 ← M[1000 + R[2]]	5000
Relative	MOV R1, 1000(PC)	R1 ← M[1000 + PC]	6000

Specification Language Notes

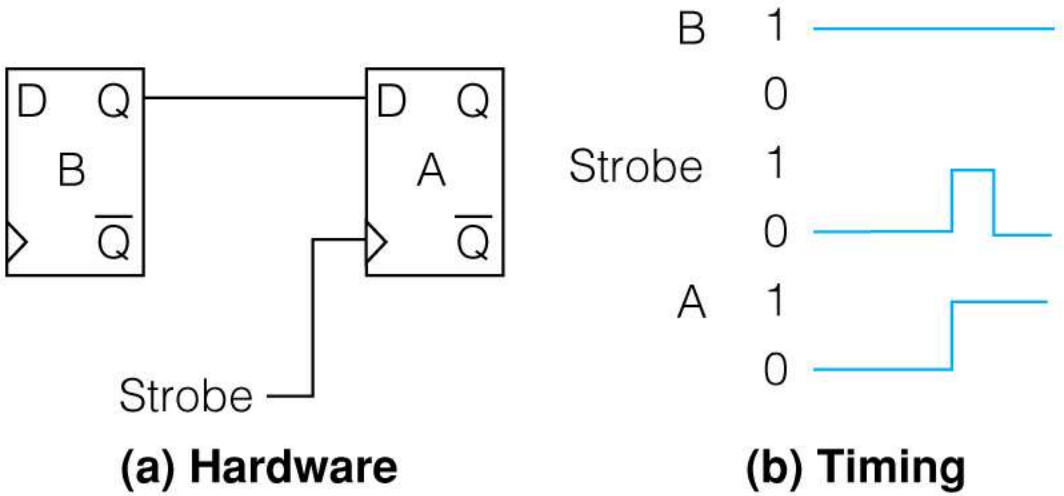
- They allow the description of *what* without having to specify *how*.
- They allow precise and unambiguous specifications, unlike natural language.
- They reduce errors:
  - errors due to misinterpretation of imprecise specifications written in natural language
  - errors due to confusion in design and implementation - “human error.”
- Now the designer must debug the specification!
- Specifications can be automatically checked and processed by tools.
  - An RTN specification could be input to a simulator generator that would produce a simulator for the specified machine.
  - An RTN specification could be input to a compiler generator that would generate a compiler for the language, whose output could be run on the simulator.

Logic Circuits in ISA

- Logic circuits
  - Gates (AND, OR, NOT) for Boolean expressions
  - Flip-flops for state variables
- Computer design
  - Circuit components support data transmission and storage as well

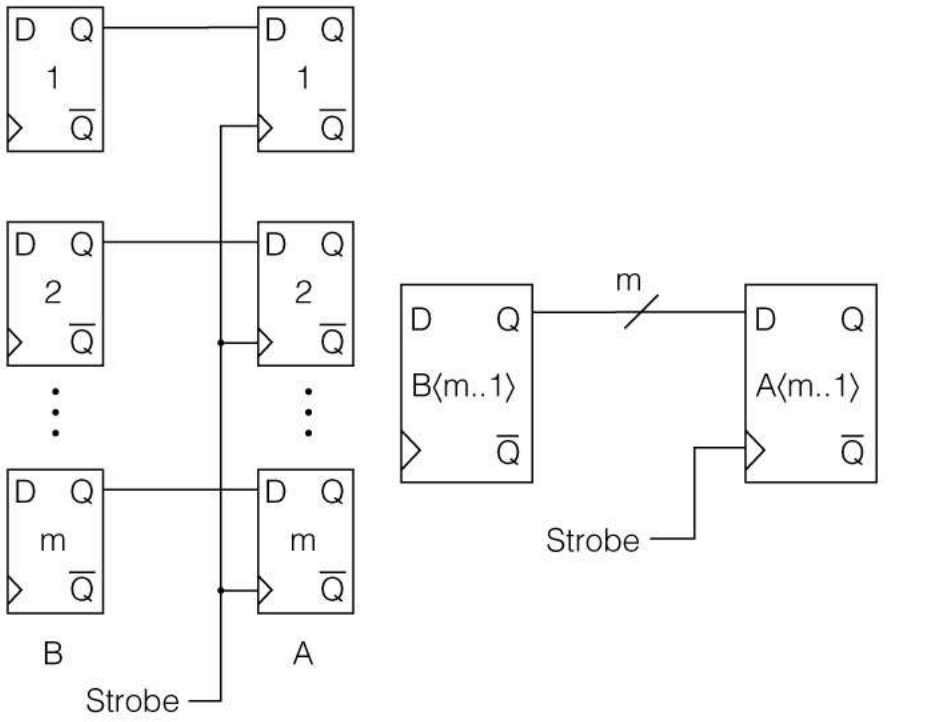
Logic Circuits for Register Transfer

- RTN statement `A` ← `B`



Multi-Bit Register Transfer

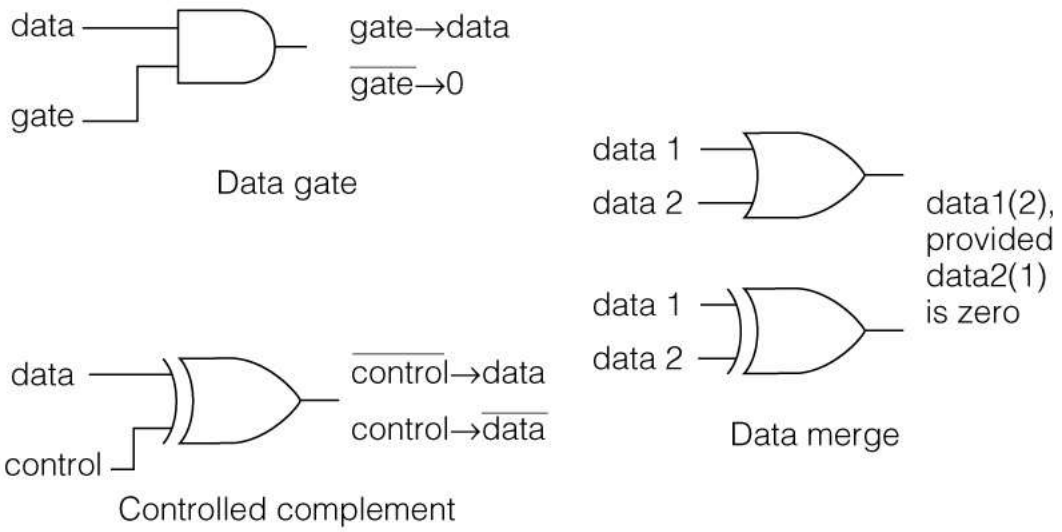
- Implementing `A<m..1>` ← `B<m..1>`



(a) Individual flip-flops **(b) Abbreviated notation**

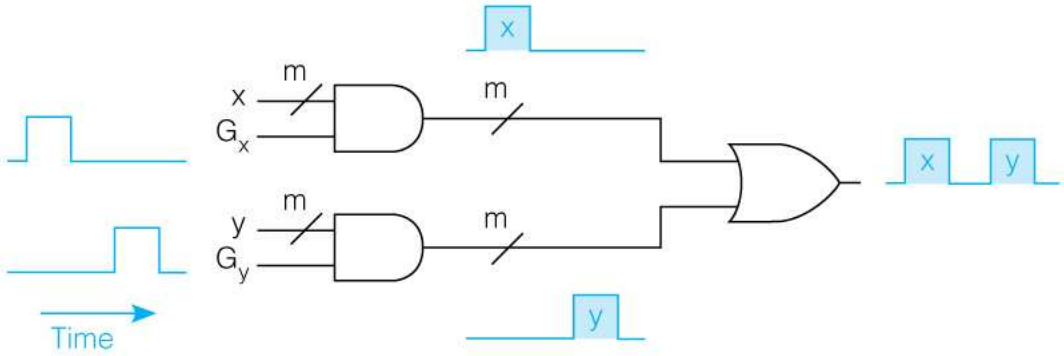
Logic Gates and Data Transmission

- Logic gates can control transmission of data

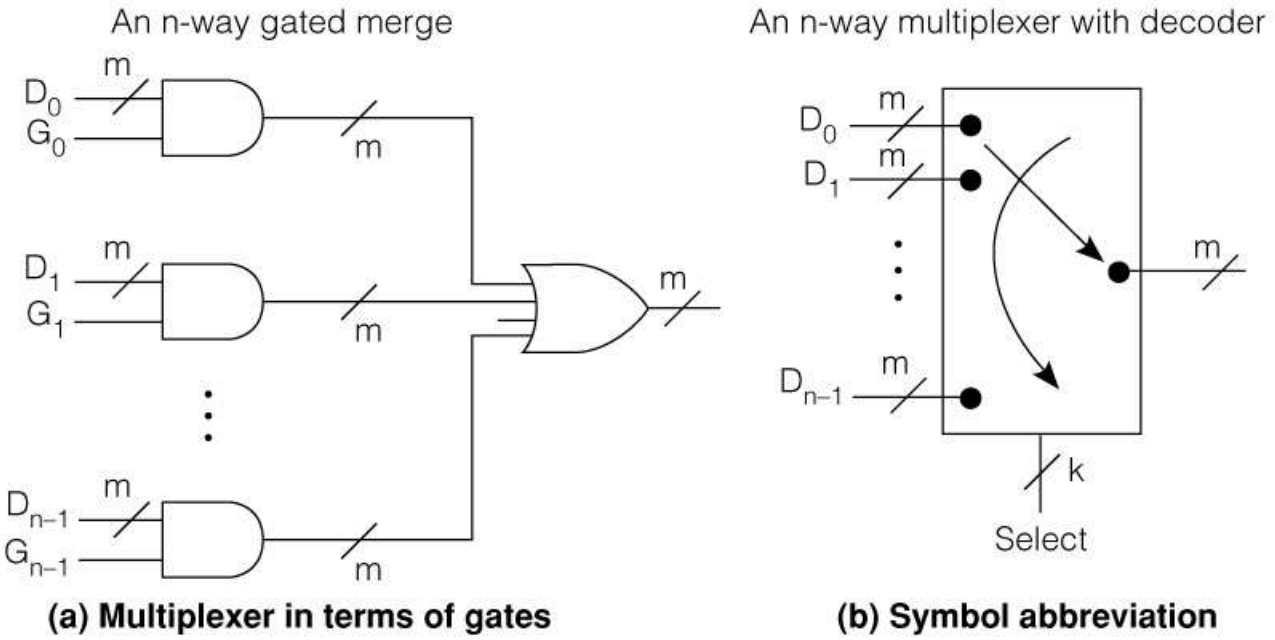


2-Way Multiplexer

- Data from multiple sources can be selected for transmission



m-Bit Multiplexer



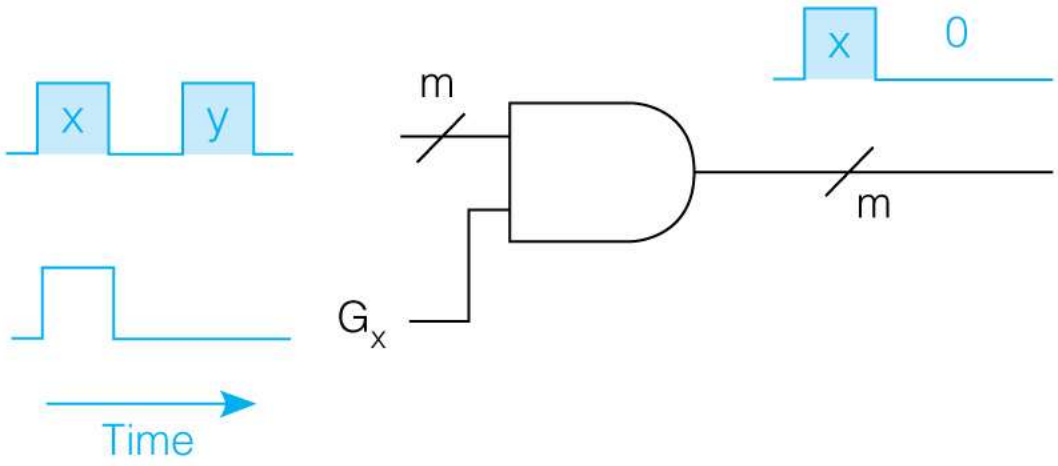
(a) Multiplexer in terms of gates

(b) Symbol abbreviation

- Multiplexer gate signals Gi may be produced by a binary to one-out-of n decoder
  - How many gates with how many inputs?
  - What is relationship between k and n?

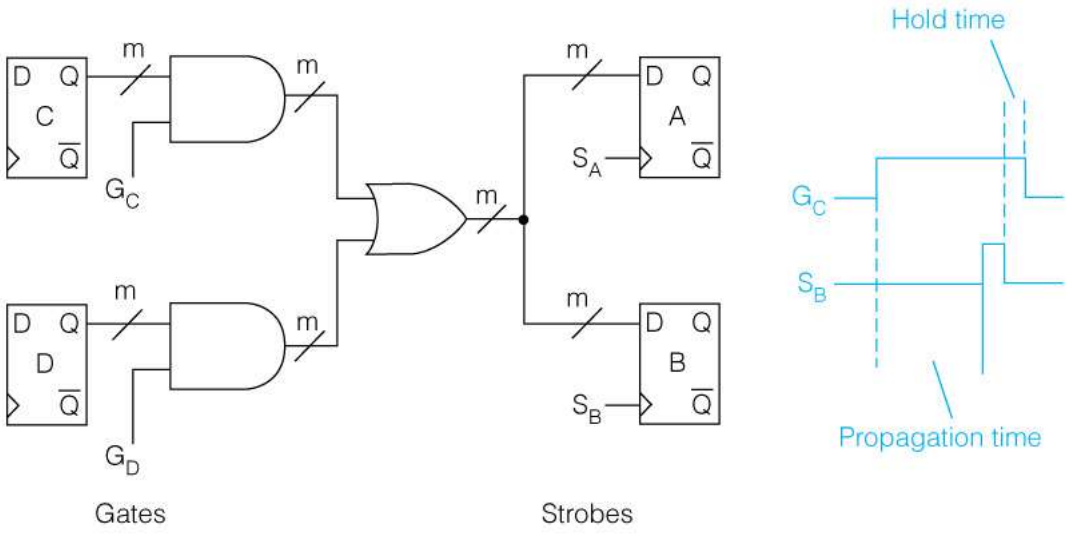
Separating Merged Data

- Merged data can be separated by gating at appropriate time
  - Can be strobed into a flip-flop when valid



Multiplexed Transfers using Gates and Strobes

- Selected gate and strobe determine which Register is transferred to where.
  - A←C, and B←C can occur together, but not A←C, and B←D

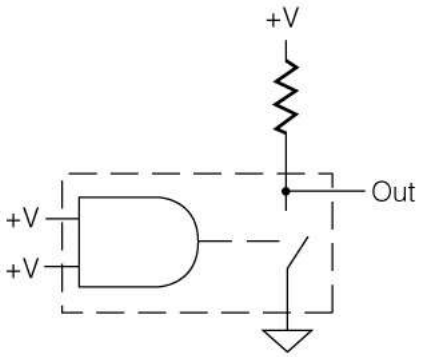


Open-Collector Bus

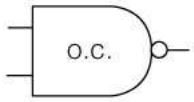
- Bus is a shared datapath (as in previous slides)
- Multiplexer is difficult to wire
  - Or-gate has large number of inputs (m x #gated inputs)
- Open-collector NAND gate to the rescue

Inputs		Output	
0v	0v	Open	(Out = +V)
0v	+V	Open	(Out = +V)
+V	0v	Open	(Out = +V)
+V	+V	Closed	(Out = 0v)

(a) Open-collector NAND truth table



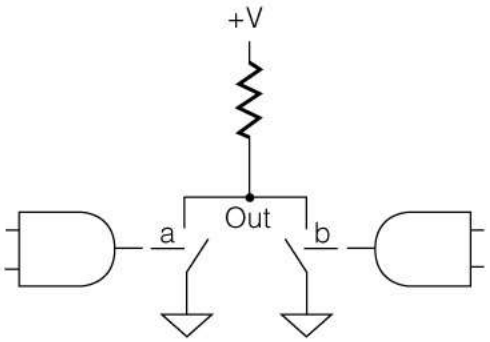
(b) Open-collector NAND



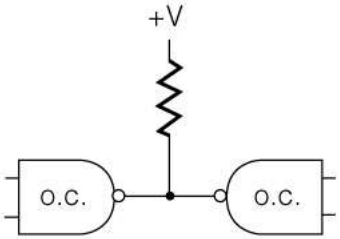
(c) Symbol

Wired AND Connection

- Connect outputs of 2 OC NAND gates
  - Only get high value when both gates are open



(a) Wired AND connection



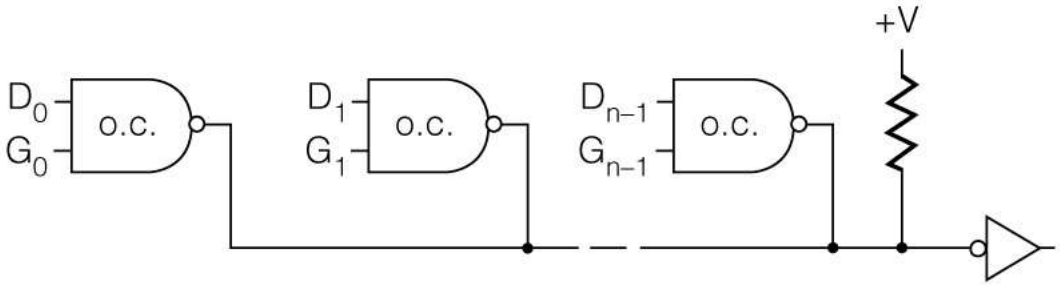
(b) With symbols

Switch		Wired AND output
a	b	
Closed(0)	Closed(0)	0v (0)
Closed(0)	Open (1)	0v (0)
Open (1)	Closed(0)	0v (0)
Open (1)	Open (1)	+V (1)

(c) Truth table

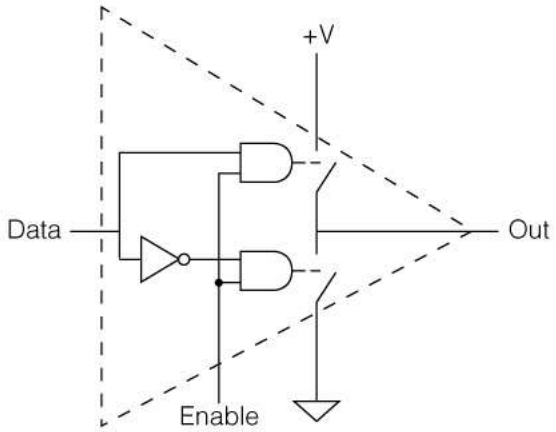
Wired-OR Bus

- Convert AND to OR using DeMorgan's Law
- Single pull-up resistor for whole bus
- OR distributed over the entire connection

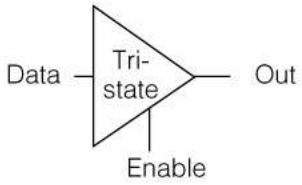


Tri-State Gate

- Controlled gating
  - Only one gate active at a time
  - Undefined output when not active



(a) Tri-state gate structure

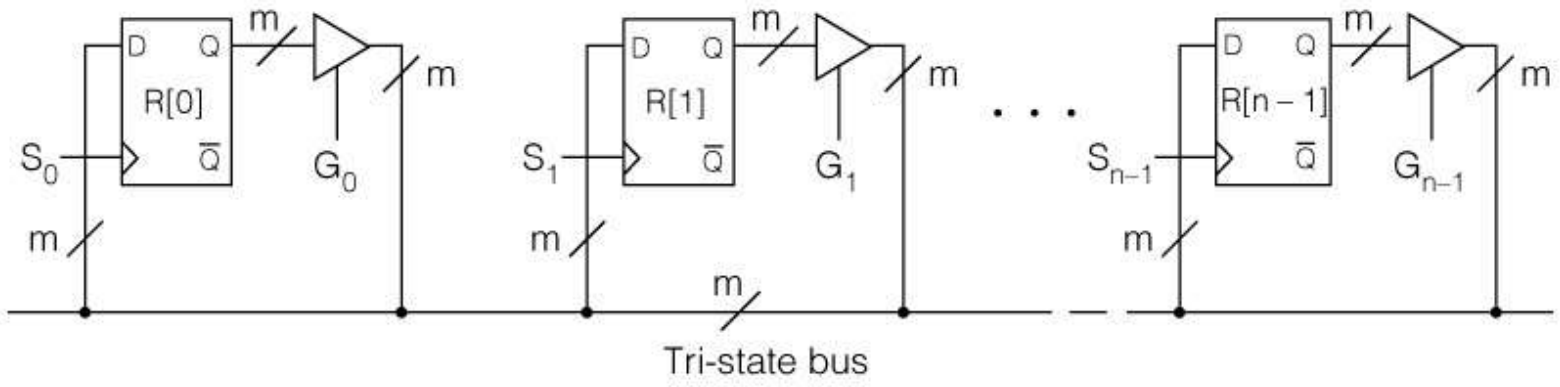


(b) Tri-state gate symbol

Enable	Data	Output
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

(c) Tri-state gate truth table

Tri-State Bus



- Can make any register transfer `R[i]` ← `R[j]`
- Only single gate may be active at a time
  - $G_i \neq G_j$

Heuring's Rules of Buses

- Only one thing on bus during a clock cycle
  - Gate-strobe paradigm
- Bus contents disappear at end of clock cycle
  - Bus items are not stored unless strobed into a register
- Clock period must be long enough to ensure valid signals everywhere along bus

- What are contents of tri-state bus when enable signal is low?
  - Hi-Z – in disconnected "floating" state

Example: Registers + ALU with Single Bus

Example

Abstract RTN

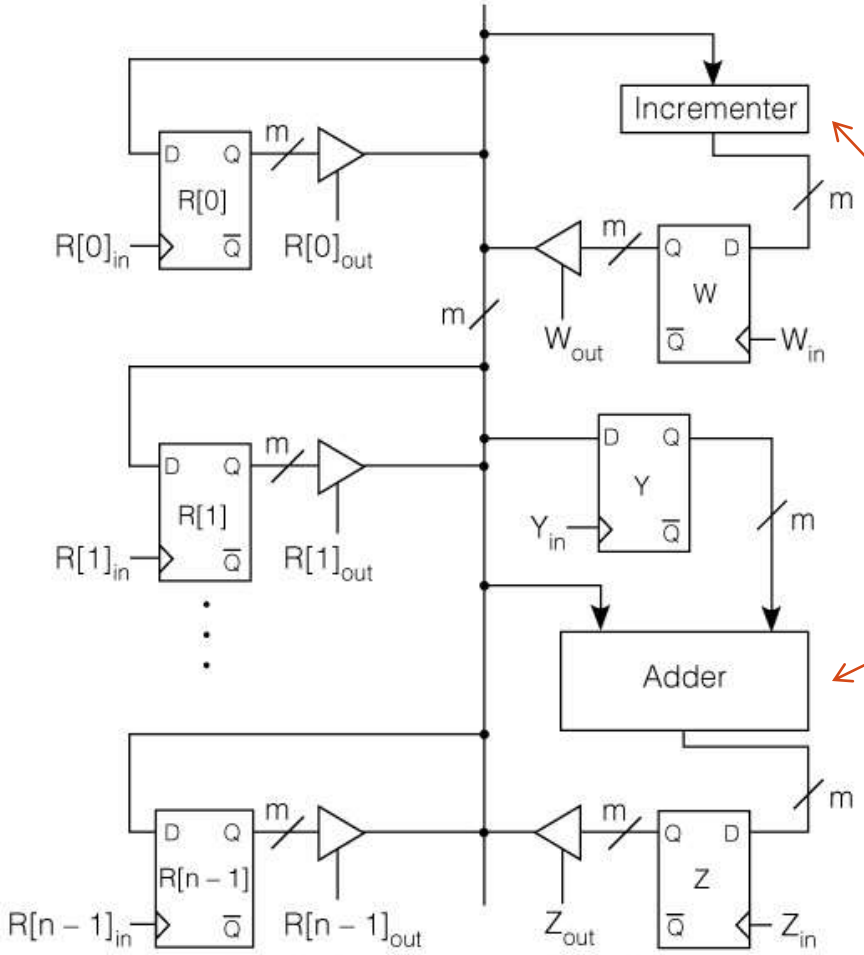
```
R[3] ← R[1]+R[2];
```

Concrete RTN

```
Y ← R[2];
Z ← R[1] + Y;
R[3] ← A;
```

Control Sequence

$R[2]_{out}$, Y_{in};
$R[1]_{out}$, Z_{in};
Z_{out}, $R[3]_{in}$;

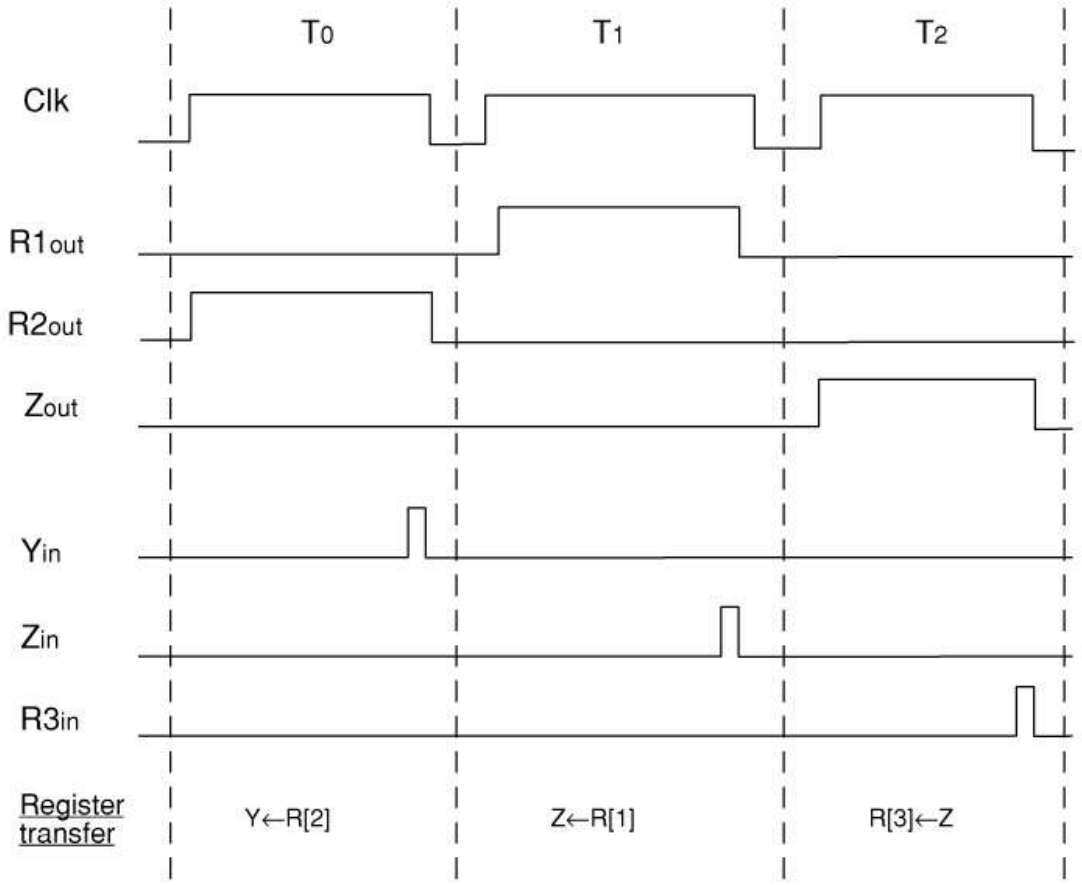


ALU-type units are combinational logic – have no memory

Note: 3 concrete steps to describe single abstract RTN step

Signal Timing

- Distinction between gating and strobing signal
- How is minimum clock period determined?



Example notes

- `R[i]` or `Y` can get the contents of anything but `Y`
- Result cannot be on bus containing operand
  - Arithmetic units have result registers
- Only one of two operands can be on the bus at a time
  - Adder has register for one operand

RTN and Implementation

- Abstract RTN
  - Describes what machine does
  - R[3] ← R[1] + R[2];
- Concrete RTN
  - Describes how it is accomplished given particular hardware implementation
  - Y← R[2]; Z ← R[1] + Y; R[3] ← Z;
- Control Sequence
  - Control signal assertion sequence to produce result
  - $R[2]_{out}$, Y_{in}; $R[1]_{out}$, Z_{in}; Z_{out}, $R[3]_{in}$

Chapter 2 Summary

- Classes of computer ISAs
- Memory addressing modes
- SRC: a complete example ISA
- RTN as a description method for ISAs
- RTN description of addressing modes
- Implementation of RTN operations with digital logic circuits
- Gates, strobes, and multiplexers