CPE300: Digital System Architecture and Design

Fall 2011

MW 17:30-18:45 CBC C316

Number Representation

09212011

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

Outline

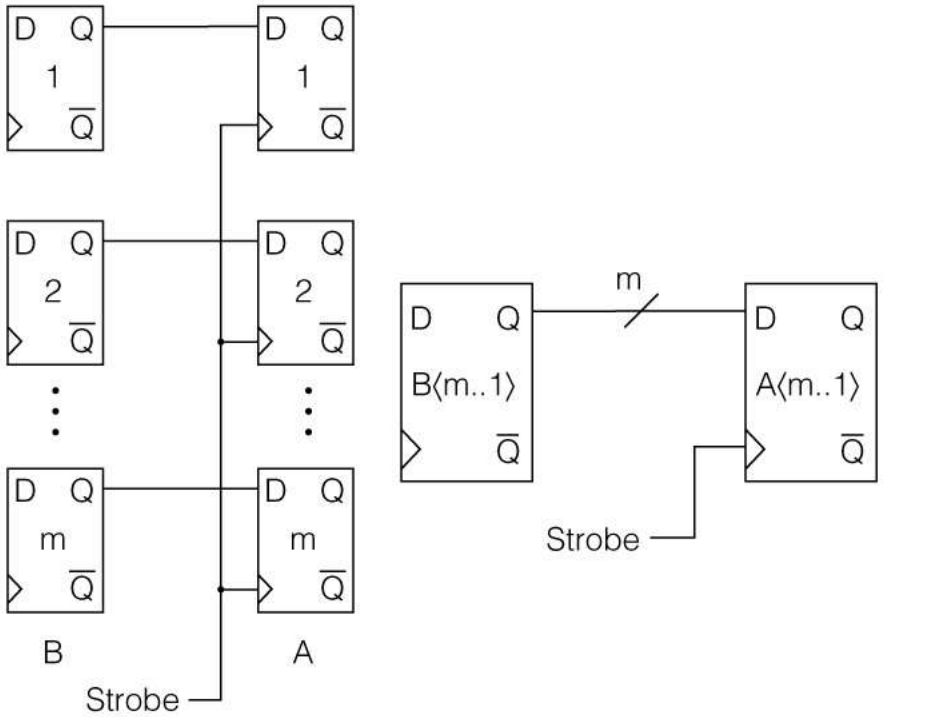
- Recap Logic Circuits for Register Transfer
- Machine Number Representation
- Performance Measurement
- CISC vs. RISC

Logic Circuits in ISA

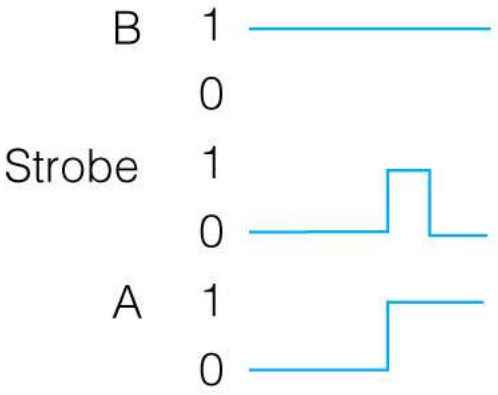
- Circuit components support data transmission and storage as well
  - Flip-flops for registers (machine state)
  - Logic gates for control

Multi-Bit Register Transfer

- Implementing A<m..1> ← B<m..1>
- Strobe signal to store (latch) value in register

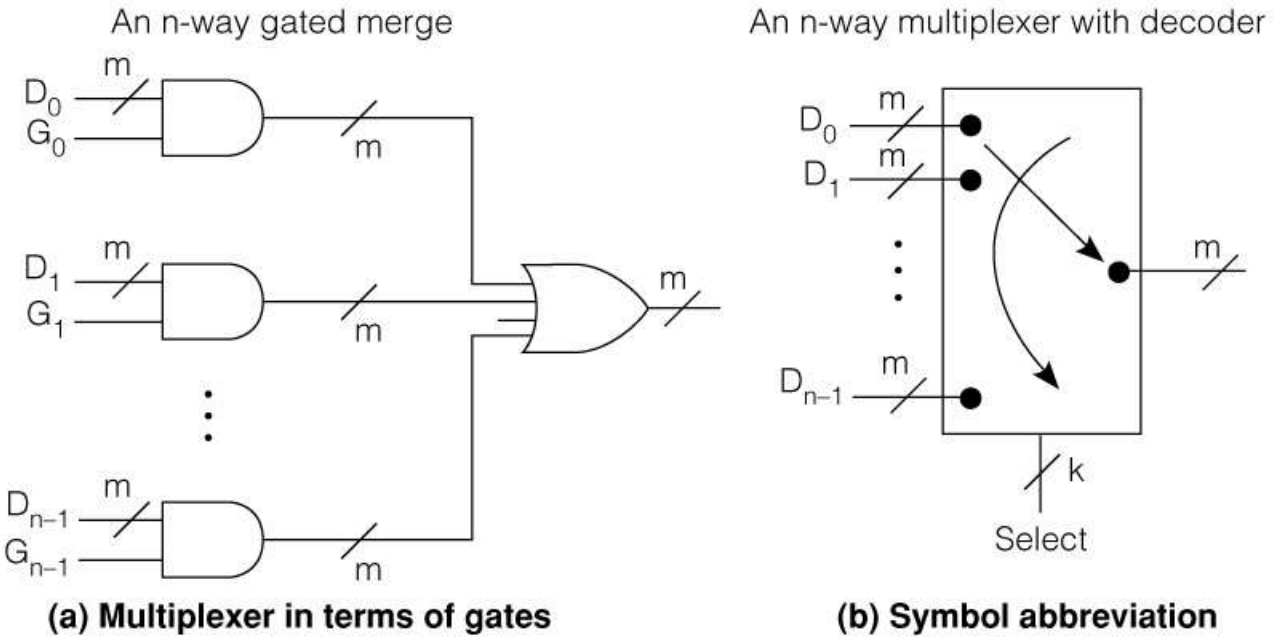


(a) Individual flip-flops (b) Abbreviated notation



(b) Timing

m-Bit Multiplexer



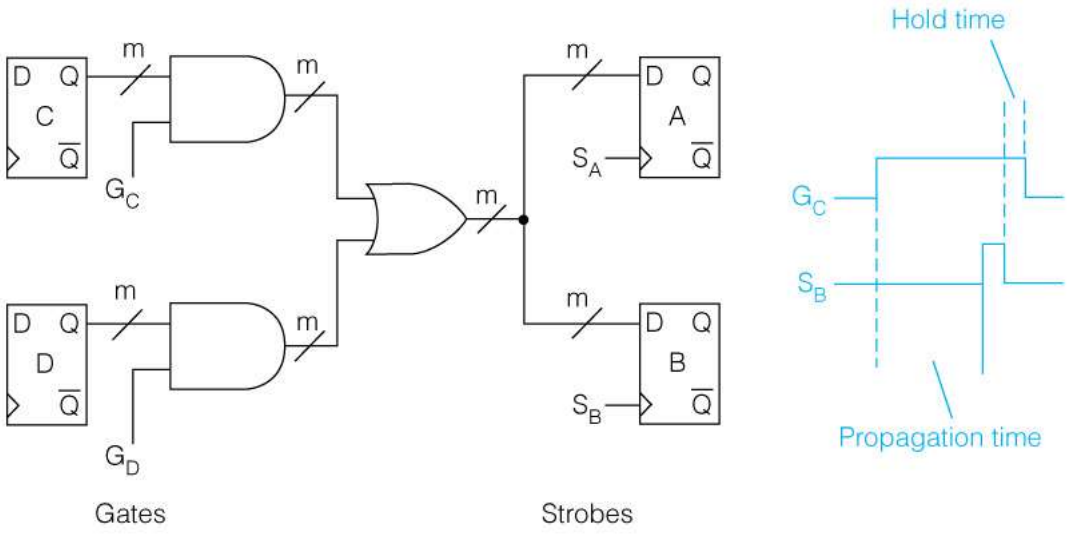
(a) Multiplexer in terms of gates

(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?

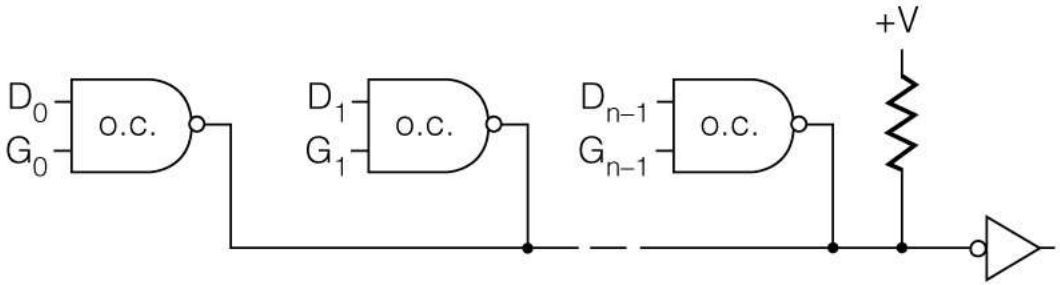
Multiplexed Transfers using Gates and Strobes

- 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



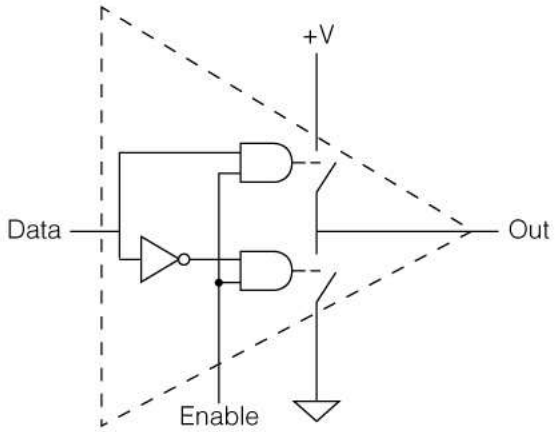
Wired-OR Bus

- Bus is a shared datapath
  - Open collector gates driving bus
  - OR distributed over the entire connection
  - Single pull-up resistor for whole bus

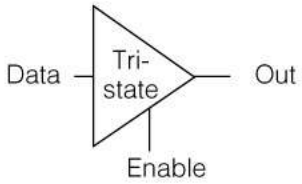


Tri-State Gate

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



(a) Tri-state gate structure

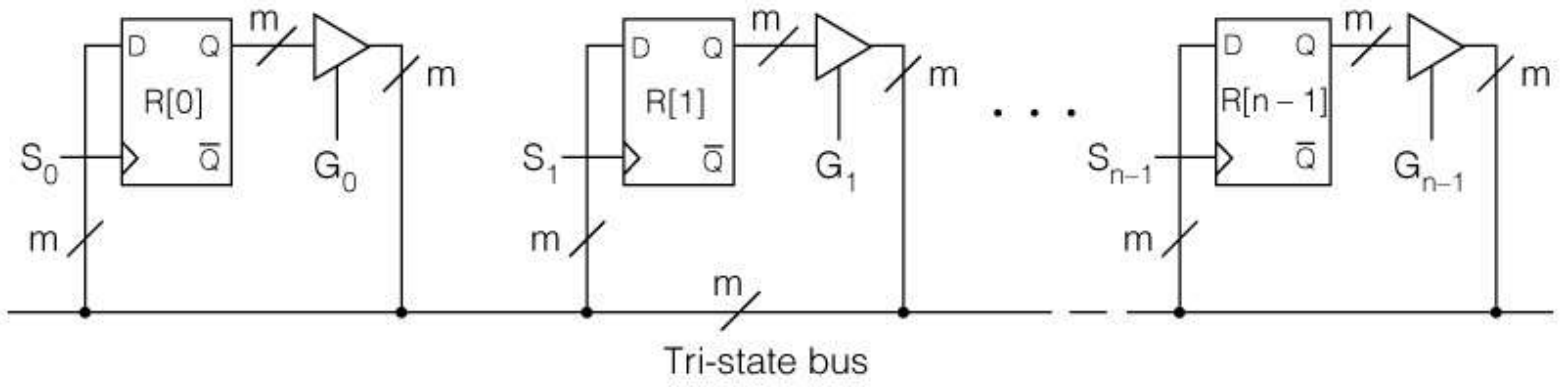


(b) Tri-state gate symbol

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
  - $G_i \neq G_j = 1$

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

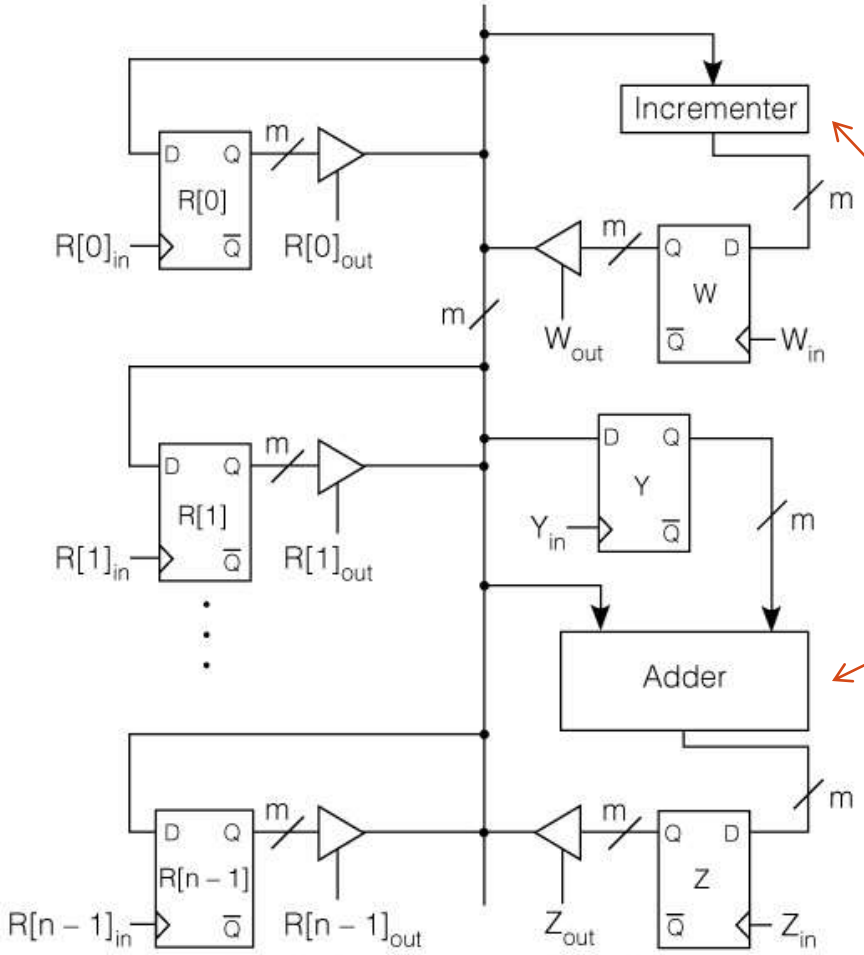
```
R[3] ← R[1]+R[2];
```

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}, $R[3]_{in}$;

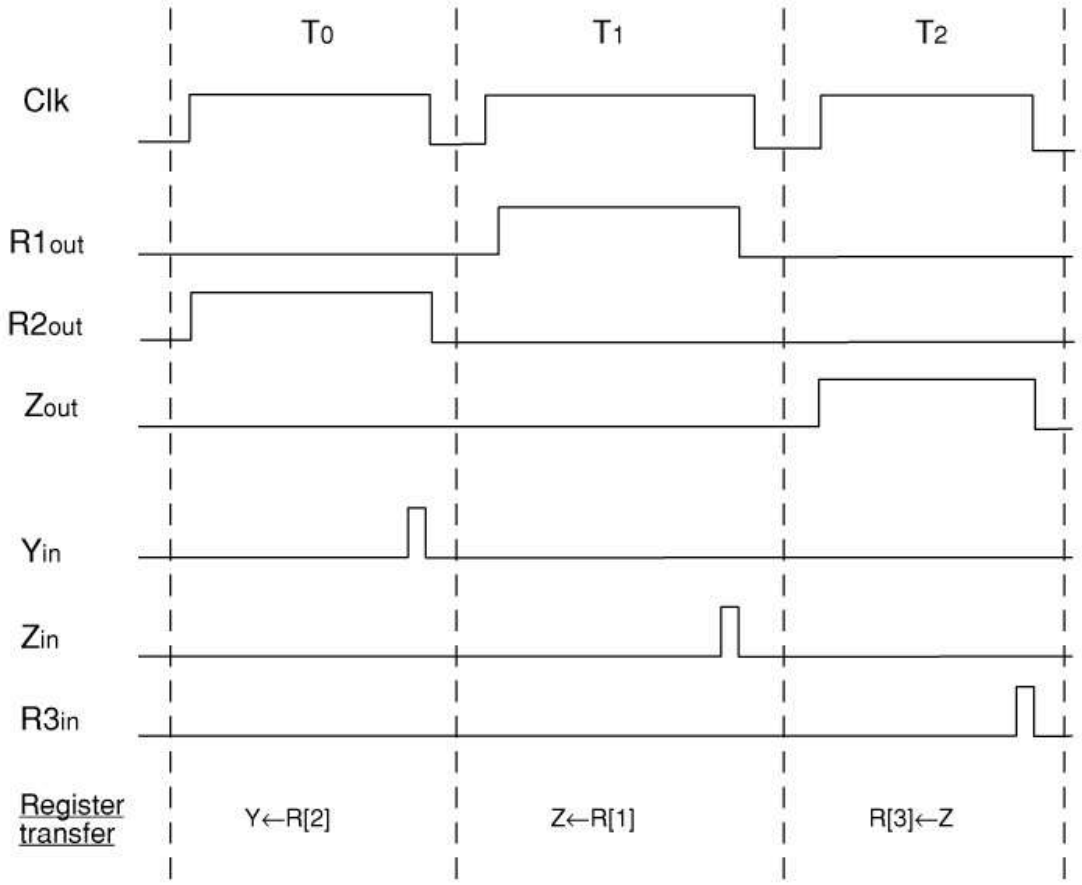


ALU-type units are combinational logic – have no memory

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

Machine Representation

- Computers manipulate bits
  - Bits must represent “things”
    - Instructions, numbers, characters, etc.
    - Must tell machine what the bits mean
- Given N bits
  - 2^N different things can be represented

Positional Notation for Numbers

- Base (radix) B number → B symbols per digit
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6 , 7, 8, 9
  - Base 2 (binary) 0, 1
- Number representation
  - $d_{31}d_{30}...d_2d_1d_0$ is 32 digit number
  - Value = $d_{31}\times B^{31} + d_{30}\times B^{30} + \ldots + d_1\times B^1 + d_0\times B^0$
- Examples
  - (Decimal): 90
    - $= 9\times 10^1 + 0\times 10^0$
  - (Binary): 1011010
    - $= 1\times 2^6 + 0\times 2^5 + 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0$
    - $= 64 + 16 + 8 + 2$
    - $= 90$
  - 7 binary digits needed for 2 digit decimal number

Hexadecimal Number: Base 16

- More human readable than binary
- Base with easy conversion to binary
  - Any multiple of 2 base could work (e.g. octal)
- Hexadecimal digits

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

  - 1 hex digit represents 16 decimal values or 4 binary digits
  - Will use `0x` to indicate hex digit

Hex/Binary Conversion

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

- Examples
  - 1010 1100 0101 (binary)
    - = 0xAC5
  - 10111 (binary)
    - = 0001 0111 (binary)
    - = 0x17
  - 0x3F9
    - = 0011 1111 1001 (binary)
    - = 11 1111 1001 (binary)

Signed Numbers

- N bits represents 2^N values
- Unsigned integers
  - Range `[0, 2^32-1]`
- How can negative values be indicated?
  - Use a sign-bit
  - Boolean indicator bit (flag)

Sign and Magnitude

- 16-bit numbers
  - +1 (decimal) = 0000 0000 0000 0001 = 0x0001
  - -1 (decimal) = 1000 0000 0000 0000 = 0x8001
- Problems
  - Two zeros
    - 0x0000
    - 0x8000
  - Complicated arithmetic
    - Special steps needed to handle when signs are same or different (must check sign bit)

Ones Complement

- Complement the bits of a number
  - +1 (decimal) = `0000 0000 0000 0001 = 0x0001`
  - -1 (decimal) = `1111 1111 1111 1110 = 0xFFFE`
- Positive number have leading zeros
- Negative number have leading ones
- Arithmetic not too difficult
- Still have two zeros

Two's Complement

- Subtract large number from a smaller one
  - Borrow from leading zeros
  - Result has leading ones
- Unbalanced representation
  - Leading zeros for positive
    - 2^{N-1} non-negatives
  - Leading ones for negative number
    - 2^{N-1} negative number
  - One zero representation
- First bit is sign-bit (must indicate width)
  - Value = $d_{31} \times -2^{31} + d_{30} \times 2^{30} + \ldots + d_1 \times 2^1 + d_0 \times 2^0$

Binary	Decimal
… 0011	3
… 0100	4
… 1111	-1

Negative value for sign bit

Two's Complement Negation

- Shortcut = invert bits and add 1
  - Number + complement = 0xF..F = -1
    - $x + \bar{x} = -1$
    - $\bar{x} + 1 = -x$
- Example

x	1111 1110
$\bar{x}$	0000 0001
$\bar{x} + 1$	0000 0010

Two's Complement Sign Extension

- Machine's have fixed width (e.g. 32-bits)
  - Real numbers have infinite width (invisible extension)
    - Positive has infinite 0's
    - Negative has infinite 1's
- Replicate sign bit (msb) of smaller container to fill new bits in larger container
- Example
  - 1111 1111 1111 1111 → 1111 1111 1111 1111 1111 1111 1111 1110

Overflow

- Fixed bit width limits number representation
- Occurs if result of arithmetic operation cannot be represented by hardware bits
- Example
  - 8-bit: 127 + 127

Binary	Decimal
0111 1111	127
0111 1111	127
1111 1110	**-2 (254)**

- Sometimes called the V flag in condition code

Chapter 3

- 3.1 Machine characteristics and performance
- 3.2 RISC vs. CISC
- 3.3 A CISC microprocessor
  - The Motorola MC68000
- 3.4 A RISC architecture
  - The SPARC

Machine Performance

- What is machine performance?
- How can performance be measured?

- Response time
  - How long to complete a task
- Throughput
  - Total work completed per unit time
  - E.g. task/per hour

Some Performance Metrics

- MIPS: Millions of Instructions Per Second
  - $\text{MIPS} = \frac{\text{Instruction Count}}{\text{Execution Time}}$
  - Pitfalls
    - Differences in ISA between machines (different instruction counts on different machines)
    - Differences in complexity between instructions
    - Different values for a single computer (two different programs)
- MFLOPS: Million Floating Point OPs Per Second
  - Other instructions counted as overhead for the floating point
  - Used by supercomputing community
- Whetstones: Synthetic benchmark
  - A program made-up to test specific performance features
- Dhrystones: Synthetic competitor for Whetstone
  - Made up to "correct" Whetstone's emphasis on floating point
- System Performance Evaluation Cooperative (SPEC)
  - Selection of "real" programs for benchmark
  - Taken from the C/Unix world

Relative Performance

- $\text{Performance}_x = \dfrac{1}{\text{Execution time}_x}$
- $\text{Speedup} = n = \dfrac{\text{Performance}_x}{\text{Performance}_y} = \dfrac{\text{Execution time}_y}{\text{Execution time}_x}$
- Example
  - Compare driving speeds. 34 mph old route and 46 mph on new
    - $n = \dfrac{\text{speed}_{new}}{\text{speed}_{old}} = \dfrac{46}{34} = 1.35$
  - Compare based on driving time. 96 minutes old route and 71 minutes on new
    - $n = \dfrac{\text{time}_{old}}{\text{time}_{new}} = \dfrac{96}{71} = 1.35$

Measuring Performance

- Program execution time is best measure of performance
- Wall clock time/response time/elapsed time
  - Total time to complete a task (including disk access, memory access, I/O, etc.)
- CPU (execution) time
  - Time spent just on CPU computation
  - User CPU time – time spent on program
  - System CPU time – time spent in OS

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock
  - Clock cycles/ticks/periods
- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12} sec
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9} Hz

CPU Time

- For a given program
  - $\text{CPU Time} = \text{CPU clock cycles} \times \text{clock cycle time}$
  - $\text{CPU Time} = \dfrac{\text{CPU clock cycles}}{\text{Clock Rate}}$
- Performance improvements
  - Reduce number of clock cycles
  - Increase clock rate
  - Hardware designer must trade off between clock cycle count and clock rate

CPU Clock Cycles

- Cycles related to number of instructions
  - CPU clock cycles = # Instructions × CPI
  - CPU Time = #Instructions × CPI × clock cycle time
  - $$\text{CPU Time} = \frac{\#\,\text{Instructions} \times \text{CPI}}{\text{Clock Rate}}$$
- Clock cycles per instruction (CPI)
  - Average number of clock cycles per instruction (given a program or program fragment)
  - Determined by CPU hardware
  - Different CPI for different instructions
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
  - Same ISA
- Which is faster, and by how much?
- Computer A:
  - CPU time = I × 2.0 × 250 ps = I × 500 ps
- Computer B:
  - CPU time = I × 1.2 × 500 ps = I × 600 ps
- Speedup
  - $n = \frac{\text{time}_{slow}}{\text{time}_{fast}} = \frac{I \times 600}{I \times 500} = 1.2$
  - Computer A is 1.2 times faster, 20% speedup

A faster

CPI Details

- Different instruction classes may have different cycle time

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Relative frequency of instruction class

Performance Summary

- CPU Time = #Instructions × CPI × clock cycle time

$$\text{Execution time} = T = IC \times CPI \times \tau$$

- T := CPU time
- IC := instruction count
- CPI := clock cycles/instruction
- τ := duration of clock period

Example 3.1

- System clock of computer increased in frequency from 700 MHz to 1.2 GHz.
- What is speedup? (assume no other factors)
- Speedup
  - $n = \frac{\text{time}_{old}}{\text{time}_{new}} = \frac{(IC \times CPI \times \tau)_{old}}{(IC \times CPI \times \tau)_{new}} = \frac{1/700}{1/1200} = 1.71$
  - IC and CPI do not change because only clock was adjusted

RISC vs. CISC Designs

- CISC: Complex Instruction Set Computer
  - Many complex instructions and addressing modes
  - Some instructions take many steps to execute
  - Not always easy to find best instruction for a task
- RISC: Reduced Instruction Set Computer
  - Few, simple instructions, addressing modes
  - Usually one word per instruction
  - May take several instructions to accomplish what CISC can do in one
  - Complex address calculations may take several instructions
  - Usually has load-store, general register ISA

Memory Bottleneck

- Memory no longer expensive
- Design for speed

Parameter	1981 (8086)	2004 (Pentium P4)	Improvement factor
Clock Frequency	4.7 MHz	4 GHz	~1000
Clock Period	212 ns	200 ps	~1000
Memory Cycle Time	100 ns	70 ns	1.4
Clocks per Memory Cycle	.47	280	~ -500

Dealing with Memory Bottleneck

- Employ one or more levels of cache memory.
  - Prefetch instructions and data into I-cache and D-cache.
  - Out of order execution.
  - Speculative execution.
- One word per instruction (RISC)
- Simple addressing modes (RISC)
- Load-Store architecture (RISC)
- Lots of general purpose registers (RISC)

RISC Design Characteristics

- Simple instructions can be done in few clocks
  - Simplicity may even allow a shorter clock period
- A pipelined design can allow an instruction to complete in every clock period
- Fixed length instructions simplify fetch & decode
- The rules may allow starting next instruction without necessary results of the previous
  - Unconditionally executing the instruction after a branch
  - Starting next instruction before register load is complete

More on RISC

- Prefetch instructions
  - Get instruction/data/location before needed in pipeline
- Pipelining
  - Beginning execution of an instruction before the previous instruction(s) have completed. (Chapter 5.)
- Superscalar operation
  - Issuing more than one instruction simultaneously.
  - Instruction-level parallelism (Chapter 5.)
- Out-of-order execution
- Delayed loads, stores, and branches
  - Operands may not be available when an instruction attempts to access them.
- Register Windows
  - ability to switch to a different set of CPU registers with a single command. Alleviates procedure call/return overhead. Discussed with SPARC (Chapter 3)