Processor Design
–
Arithmetics and Datapath Design

Professor Jari Nurmi
Institute of Digital and Computer Systems
Tampere University of Technology, Finland
email jari.nurmi@tut.fi

Refinement of HW Design Process

- Processor design can start with a very informal description
- It is useful to describe the functionality of the processor instructions formally on high level using e.g.
  - VHDL
  - SystemC
  - Some register transfer notation (like RTN in Heuring – Jordan)
- When proceeding in the design process, more details can be added to refine the description
- We proceed to design the datapath and identify needed control signals
- Pipelining further refines the processor operation
- Control part is implemented to generate the ctrl signals

Arithmetics Revisited

- Digital number systems
- Fixed-point arithmetics

Digital Number Systems

- Digital number systems have a base or radix b
- Using positional notation, an m-digit base b number is written

$$x = x_{m-1}\, x_{m-2} \ldots x_1\, x_0$$

$$0 \le x_i \le b-1,\ 0 \le i < m$$

- The value of this unsigned integer is

$$value(x) = \sum_{i=0}^{m-1} x_i \cdot b^i$$

Fractions and Fixed-Point Numbers

- The value of the base b fraction $.f_{-1}f_{-2}...f_{-m}$ is the value of the integer $f_{-1}f_{-2}...f_{-m}$ divided by b^m
- The value of a mixed fixed point number

  $x_{n-1}x_{n-2}...x_1x_0.x_{-1}x_{-2}...x_{-m}$

  is the value of the n+m digit integer

  $x_{n-1}x_{n-2}...x_1x_0x_{-1}x_{-2}...x_{-m}$

  divided by b^m
- Moving radix point one place left divides by b
  - For fixed radix point position in word, this is a right shift of word
- Moving radix point one place right multiplies by b
  - For fixed radix point position in word, this is a left shift of word

Negative Numbers, Complements, and Complement Representations

- Complement number systems
  - Represent both positive and negative numbers
- One's complement
- Two's complement
  -
- How to compute the complements
- Use of complement number systems in signed addition hardware

Complement Number Systems

- Complement number systems use unsigned numbers to represent both positive and negative numbers
- Recall that the range of an m digit base b unsigned number is $0 \le x \le b^m - 1$
- The first half of the range is used for positive, and the second half for negative, numbers
- Positive numbers are simply represented by the unsigned number corresponding to their absolute value

Use of Complements to Represent Negative Numbers

- The complement of a number in the range from 0 to $b^m/2$ is in the range from $b^m/2$ to $b^m - 1$
- A negative number is represented by the complement of its absolute value
- There are an equal number (±1) of positive and negative number representations
  - The ±1 depends on whether b is odd or even and whether radix complement or diminished radix complement is used
- We will assume the most useful case of b = 2
  - Then radix (2's) complement system has one more negative representation
  - Diminished radix (1's) complement system has equal numbers of positive and negative representations

Reasons to Use Complement Systems for Negative Numbers

- The usual sign-magnitude system introduces extra symbols + and - in addition to the digits
- In binary, it is easy to map 0 〉 + and 1 〉 -
- In base b > 2, using a whole digit for the two values, + and - , is wasteful
- Most important, however, it is easy to do signed addition and subtraction in complement number systems

Negation in Complement Number Systems

- Except for $-b^m/2$ in the 2's complement system, the negative of any m-bit value is also m bits
- The negative of any number x, positive or negative, in the 2's or 1's complement system is obtained by applying the complement operation to x, respectively
- 1's complement is taken by simply inverting each bit
- In 2's complement we need to add 1 to the 1's complement

Complement Fractions

- Since m digit fraction is same as m digit integer divided by b^m, the b^m in complement definitions corresponds to 1 for fractions
- Thus 2's complement of $x = .x_{-1}x_{-2}...x_{-m}$ is
  (1-x) mod 1, where mod 1 means discard integer
- The range of fractions is roughly -1 to +1

Scaling Complement Numbers by Powers of the Base

- Roughly, multiplying by b corresponds to moving radix point one place right or shifting number one place left
- Dividing by b roughly corresponds to a right shift of the number or a radix point move to the left one place
- There are 2 new issues for complement numbers:
  1) What is new left digit on right shift?
  2) When does a left shift overflow?

Right Shifting a Complement Number to Divide by 2

- For positive $x_{m-1}x_{m-2}...x_1x_0$, dividing by 2 corresponds to right shift with zero fill

  $$0x_{m-1}x_{m-2}...x_1$$

- For negative $x_{m-1}x_{m-2}...x_1x_0$, dividing by 2 corresponds to right shift with 1-fill

  $$(1)x_{m-1}x_{m-2}...x_1$$

- So, we feed in the sign bit!

- This holds for both 2's and 1's complement systems

Complement Number Overflow on Left Shift to Multiply by 2

- For positive numbers, overflow occurs if any bit 1 shifts off left end
- Positive numbers also overflow if the digit shifted into left position makes number look negative
- For negative numbers, overflow occurs if bit 0 shifts off left end
- Negative numbers also overflow if new left digit makes number look positive

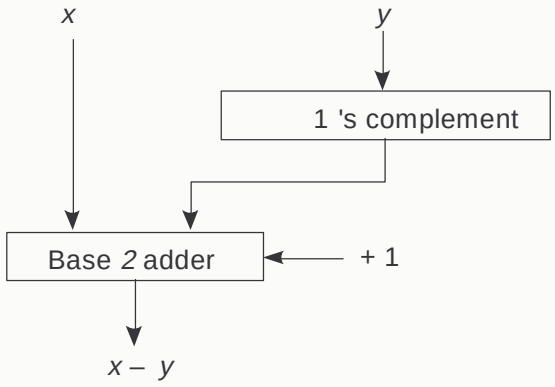
Fixed-Point Addition and Subtraction

- If the radix point is in the same position in both operands, addition or subtraction act as if the numbers were integers
- Addition of signed numbers in radix complement system needs only an unsigned adder
- So we only need to concentrate on the structure of an m-bit binary unsigned adder
- To see this let x be a signed integer and rep(x) be its 2's complement representation
- The following theorem summarizes the result

  Operands are rep(x) & rep(y). Then s = rep(x+y), except for overflow
- So, all we need is a binary m-bit adder

Base b Radix Complement Subtracter

- To do subtraction in the radix complement system, it is only necessary to negate (radix complement) the 2nd operand
- It is easy to take the diminished radix complement, and the adder has a carry-in for the +1

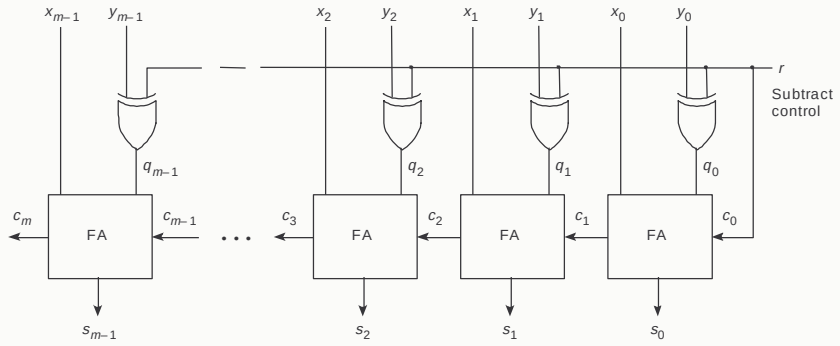


Overflow Detection in Complement Add and Subtract

- We saw that all cases of overflow in complement addition came when adding numbers of like signs, and the result seemed to have the opposite sign
- For even b=2, the sign can be determined from the left digit of the representation
- Thus an overflow detector only needs x_{m-1}, y_{m-1}, s_{m-1}, and an add/subtract control

2's Complement Adder/Subtracter

- A multiplexer to select y or its complement becomes an exclusive OR gate



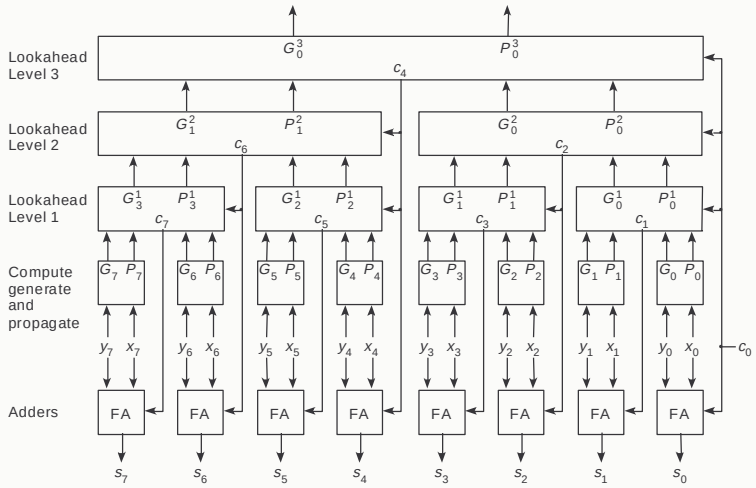
Speeding up Addition

- One way or another speed up the carry chain
- Carry look-ahead
  - calculate the propagate/generate information
  - decision of the carry-out from each bit can be then done locally
- Carry-skip
  - calculate the propagate/generate information
  - use it to effectively cut the carry chain by skipping sections
- Carry-select
  - pre-calculate both with carry-in = 0 and =1
  - choose the right when carry known (block-wise)

Binary Propagate and Generate Signals

- In binary, the generate for digit j is $G_j = x_j \cdot y_j$
- Propagate for digit j is $P_j = x_j + y_j$
  - Of course $x_j + y_j$ covers $x_j \cdot y_j$ but it still corresponds to a carry out for a carry in
- Carries can then be written: $c_1 = G_0 + P_0 \cdot c_0$
- $c_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0$
- $c_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0$
- $c_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0$
- In words, the c_2 logic is: c_2 is one if digit 1 generates a carry, or if digit 0 generates one and digit 1 propagates it, or if digits 0 and 1 both propagate a carry-in

Carry Lookahead Adder for Group Size k = 2

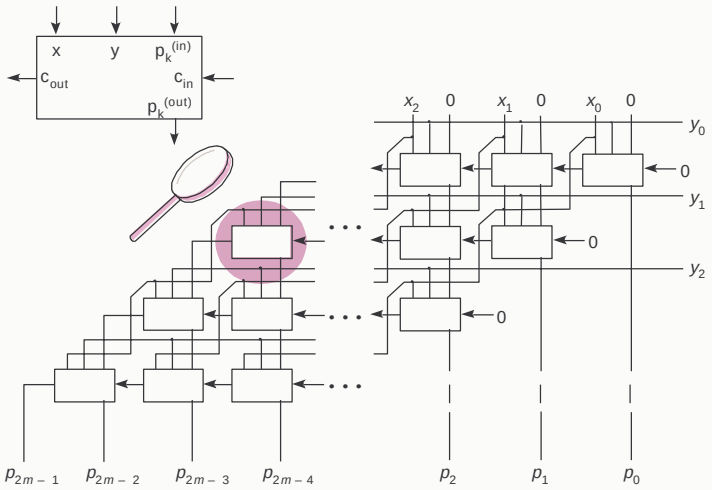


Digital Multiplication Schema

				x_3	x_2	x_1	x_0	multiplicand
				y_3	y_2	y_1	y_0	multiplier
			$(xy_0)_4$	$(xy_0)_3$	$(xy_0)_2$	$(xy_0)_1$	$(xy_0)_0$	pp_0
		$(xy_1)_4$	$(xy_1)_3$	$(xy_1)_2$	$(xy_1)_1$	$(xy_1)_0$		pp_1
	$(xy_2)_4$	$(xy_2)_3$	$(xy_2)_2$	$(xy_2)_1$	$(xy_2)_0$			pp_2
$(xy_3)_4$	$(xy_3)_3$	$(xy_3)_2$	$(xy_3)_1$	$(xy_3)_0$				pp_3
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	

p: product pp: partial product

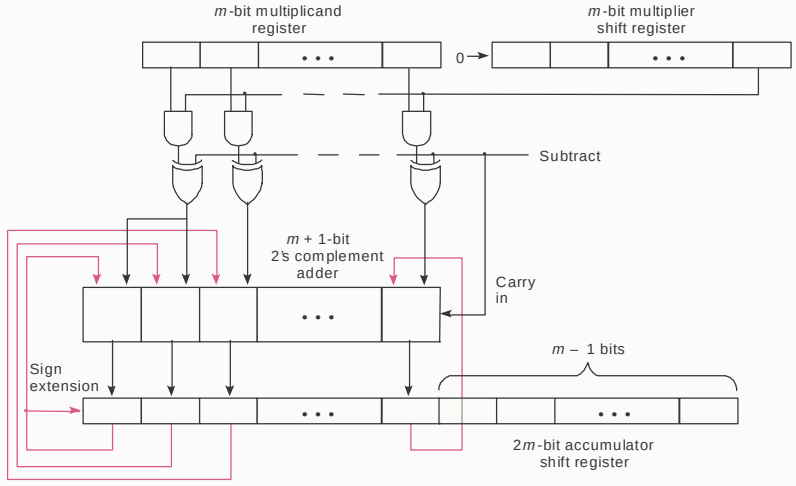
Parallel Array Multiplier for Unsigned Numbers



Signed Multiplication

- The sign of the product can be computed immediately from the signs of the operands
- For complement numbers, negative operands can be complemented, their magnitudes multiplied, and the product recomplemented if necessary
- A complement representation multiplicand can be handled by a 2's complement adder for partial products and sign extension for the shifts

2's Complement Multiplier Hardware



A 2's Complement Integer's Value Can Be Represented as:

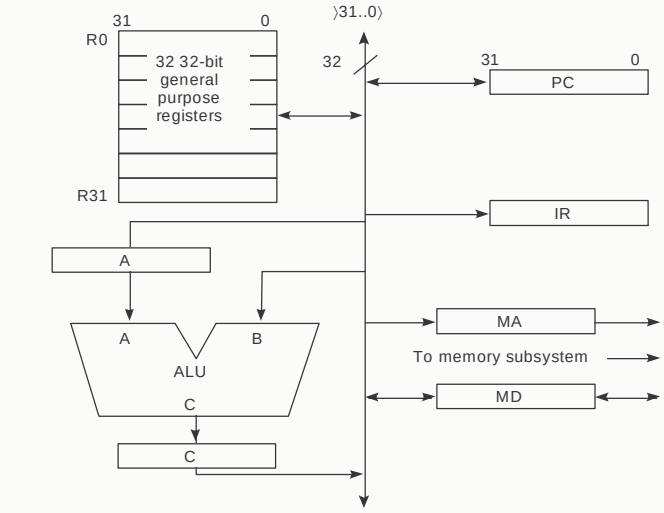
$$value(y) = -y_{m-1}\,2^{m-1} + \sum_{i=0}^{m-2} Y_i\,2^i$$

This means that the value can be computed by *adding* the weighted values of all the digits except the most significant, and *subtracting* that digit.

13

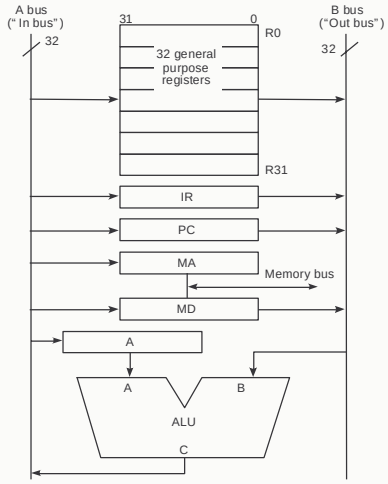
Exploration of Architecture Alternatives

1-bus SRC (Simple RISC)



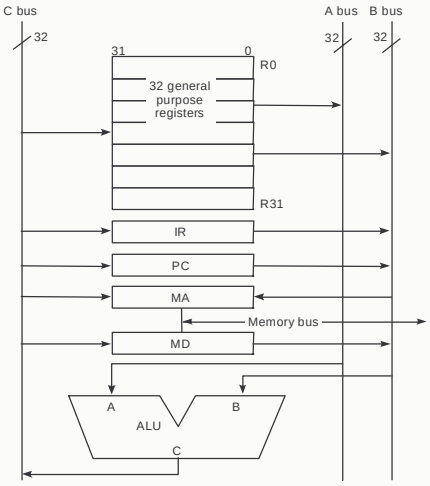
Source: Heuring – Jordan: Computer Systems Architecture and Design

2-bus SRC (Simple RISC)



Source: Heuring – Jordan: Computer Systems Architecture and Design

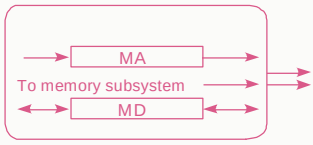
3-bus SRC (Simple RISC)

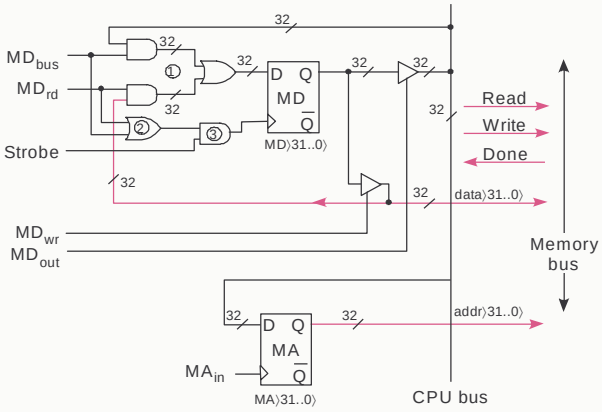


Source: Heuring – Jordan: Computer Systems Architecture and Design

The CPU–Memory Interface: Memory Address and Memory Data Registers, MA<31...0> and MD<31...0>

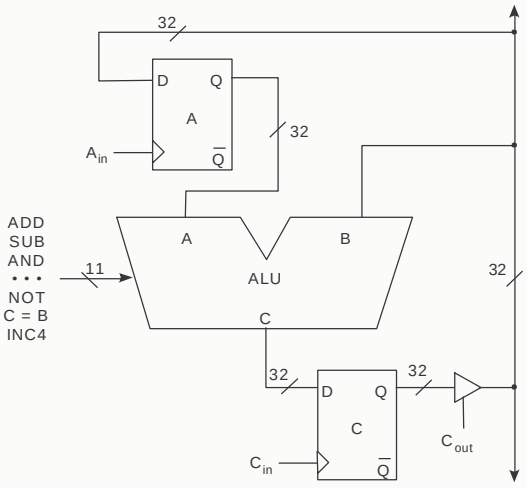
- MD is loaded from memory or from CPU bus



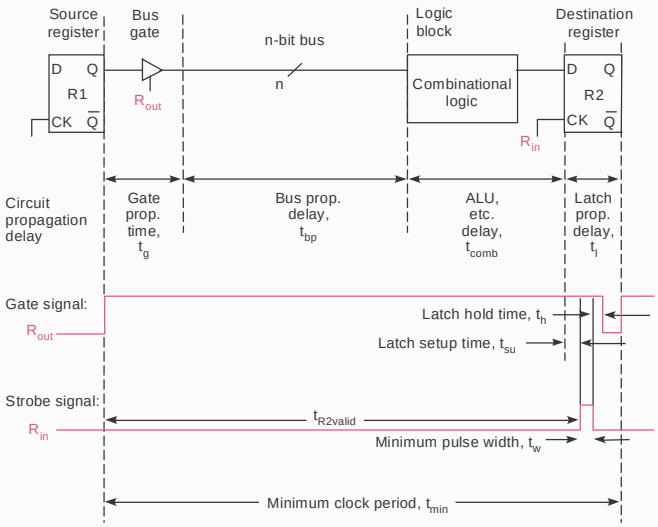


- MD can drive CPU bus or memory bus

Construction of Datapath



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

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

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

Implementation Issues 1

- Registers vs. Latches
  - Latch is only about ½ of the FF complexity
  - Transparency may cause malfunctions in the presence of feedback
  - Testability or technology selections may restrict latch usage
  - However, master-slave FF can be made up from two latches
  - Requires two-phase **non-overlapping** clocking
  - Useful in retiming critical paths (separation of M and S latches!)
  - Other clocking schemes may be used with edge-triggered FFs
  - D-type of FF is most useful from complexity point of view
  - SR, JK, T flip-flops can be easily constructed from DFF if needed

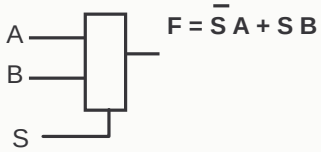
Implementation Issues 2

- Tri-states vs. multiplexers

E	A	F
0	0	Z
0	1	Z
1	0	0
1	1	1

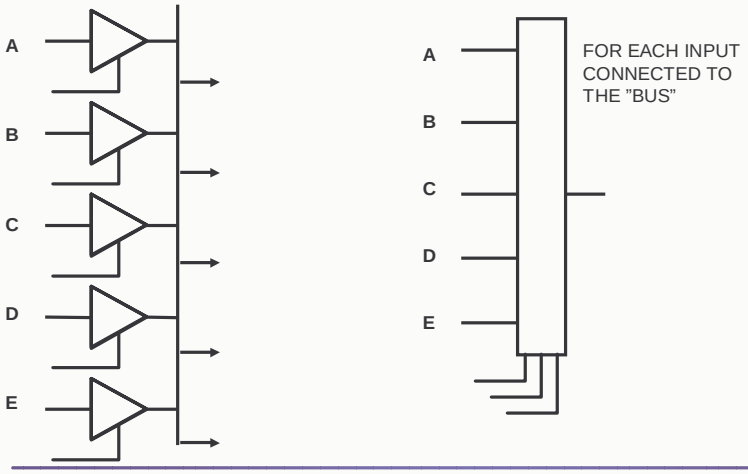
Tri-state buffer

S	A	B	F
0	0	x	0
0	1	x	1
1	x	0	0
1	x	1	1



2-input multiplexer

- Bus implementation



Implementation Issues 3

- Target technology impacts
  - FPGA restrictions
    - Tri-states may not be available
    - Latches should be avoided
    - Asynchronous operation may fail because of routing delays
  - CMOS ASIC technologies
    - Tri-state is the default for electrical bus interface
    - Latches may be allowed
      - Input latching for low power
      - Division of MS-flip-flops to two latch distributed stages
      - "Latch-file"
      - Testability issues may still restrict the use of latches
    - Asynchronous operation better under control
    - Proportion of routing delays vs. gate delays a function of line width
    - Floorplan, routing plan, layer allocation well under designer control

Implementation Issues 4

- Bus signalling
  - Conventional voltage-level based bus
  - Current-mode bus
  - DS-CDMA bus

End of Datapath Design

next we will look at pipelining
and other forms of parallelism