Summary of previous lectures

- computer architecture

  = instruction set architecture + machine organisation

- $\text{CPI} = \sum (\text{CPI}_i \times \text{instr. count}_i) / (\sum \text{instr. count}_i)$

- minimise: $\text{exe. time} = \text{instr. count} \times \text{CPI} \times \text{cycle time}$

- CISC: ↓ instr. count ↓ code size ↑ CPI ↑ cycle time

  RISC: ↑ instr. count ↑ code size ↓ CPI ↓ cycle time

Computer arithmetic

(3rd Ed: p.160-175, Apx. B; 4th Ed: p.224-229, Apx. C.5; 5th Ed: p.178-182, Apx. B)

- two's complement: signed integer representation
- e.g. $1011_{2C} = (1\times -2^3) + (0\times 2^2) + (1\times 2^1)+(1\times 2^0) = -5_{ten}$
- n-bit: range (-2^{n-1}) .. $(2^{n-1}-1)$
- sign extension: $1011_{2C} = 1111011_{2C}$
- overflow: $A, B > 0, \quad A+B \leq 0$
  $A, B < 0, \quad A+B \geq 0$
- in MIPS: slt, slti work with two's complement
  sltu, sltiu work with unsigned representation
  (do not cause exception when overflow)

Logical operations

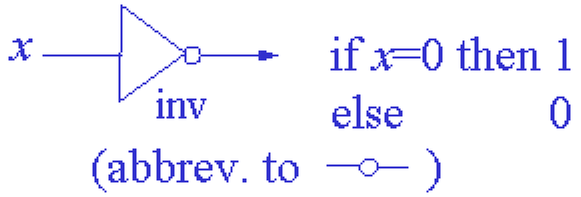
- shift left logical
  - `sll $10, $16, 8`   # reg10 = reg16 « 8 bits
  - reg16 0 .. 0 0000 0000 1101
  - reg10 $\underbrace{\text{0 .. 0}}_{\text{20 bits}}$ 1101 $\underbrace{\text{0000 0000}}_{\text{introduce zeros}}$
  - format

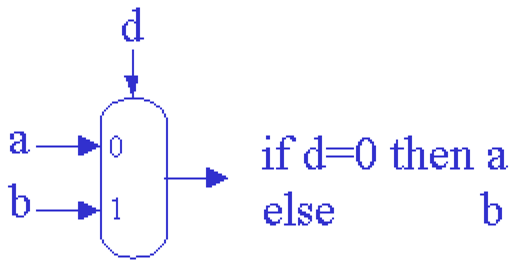
0	0	16	10	8	0
R type	source1	source2	dest.	shamt	funct

- shift left logical variable (sllv): shamt in register source1
- right shifts: srl, srlv, sra (sign-extend high order bits)
- bitwise: or, and, ori, andi

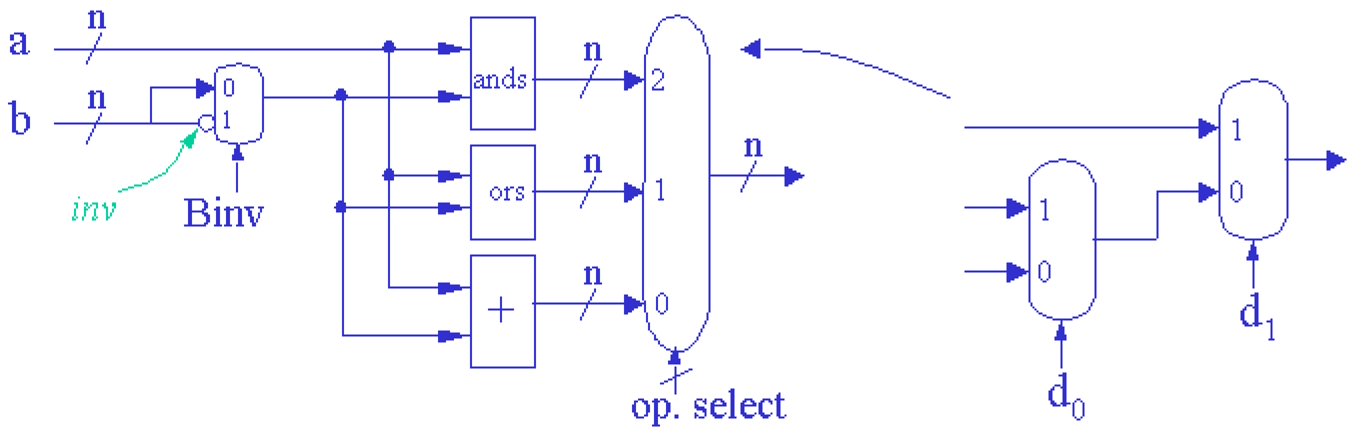
ALU building blocks

- and, or, inv, mux





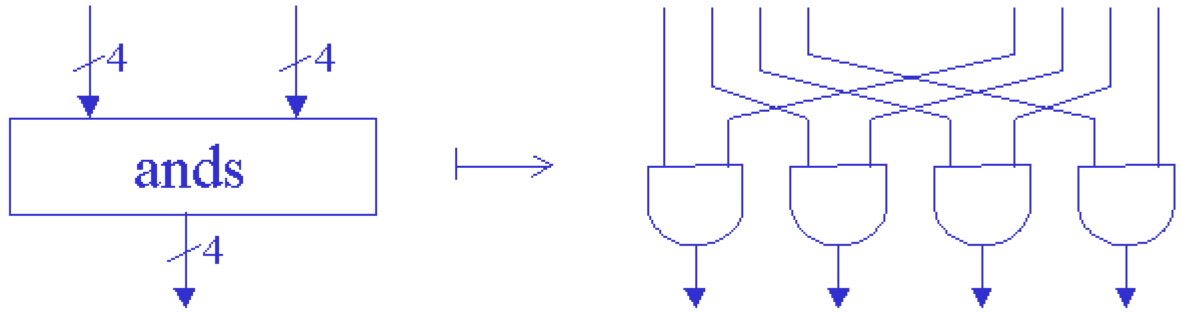
- ALU: Arithmetic Logic Unit n=32 for MIPS



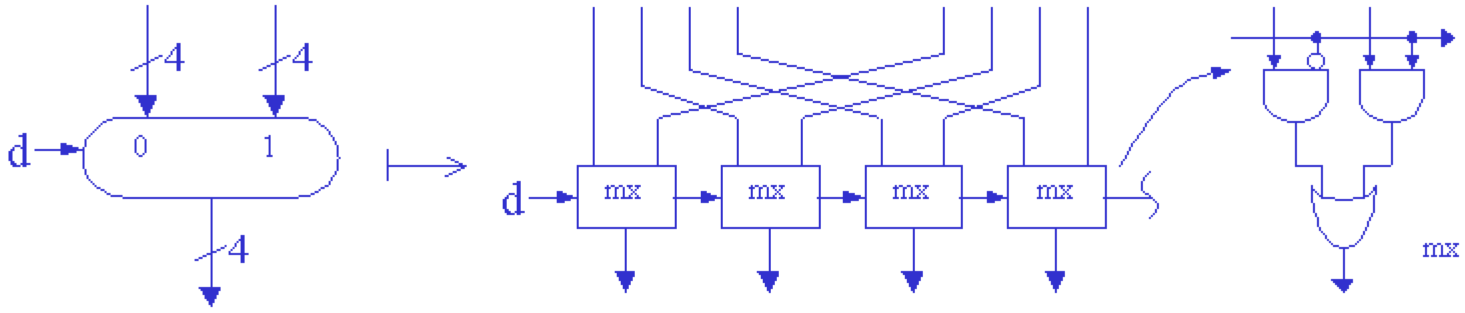
- bit-level realisation: hierarchical, regular structure

Bit-wise logical / selection operations

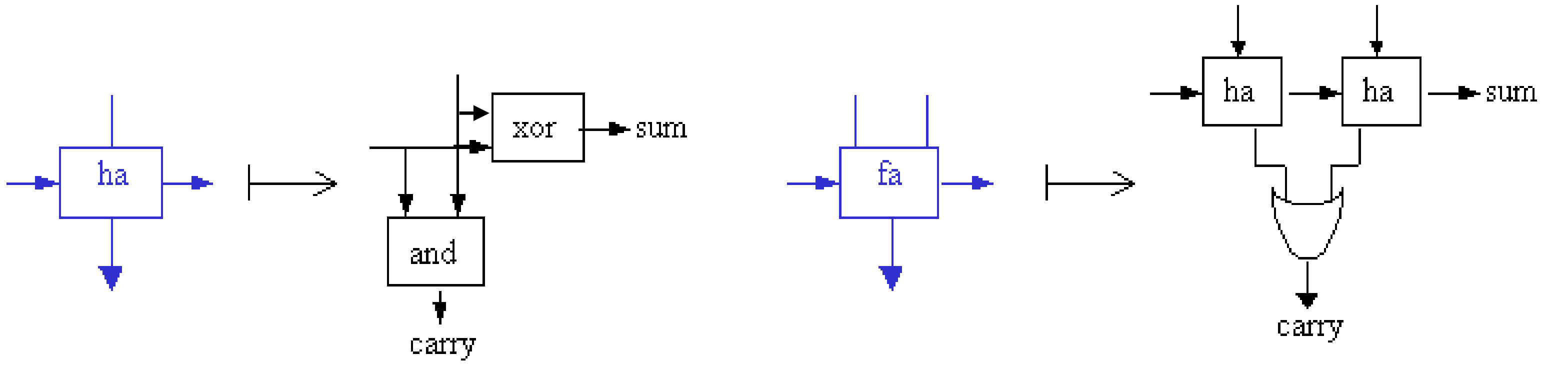
- and, or, …

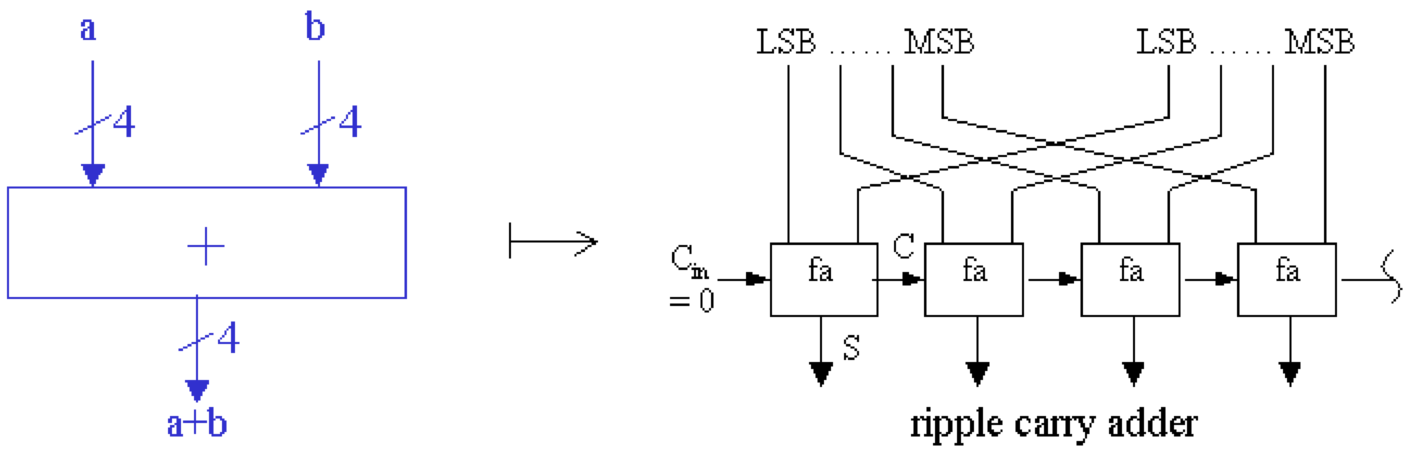


- selector / multiplexor



Add / subtract

- 

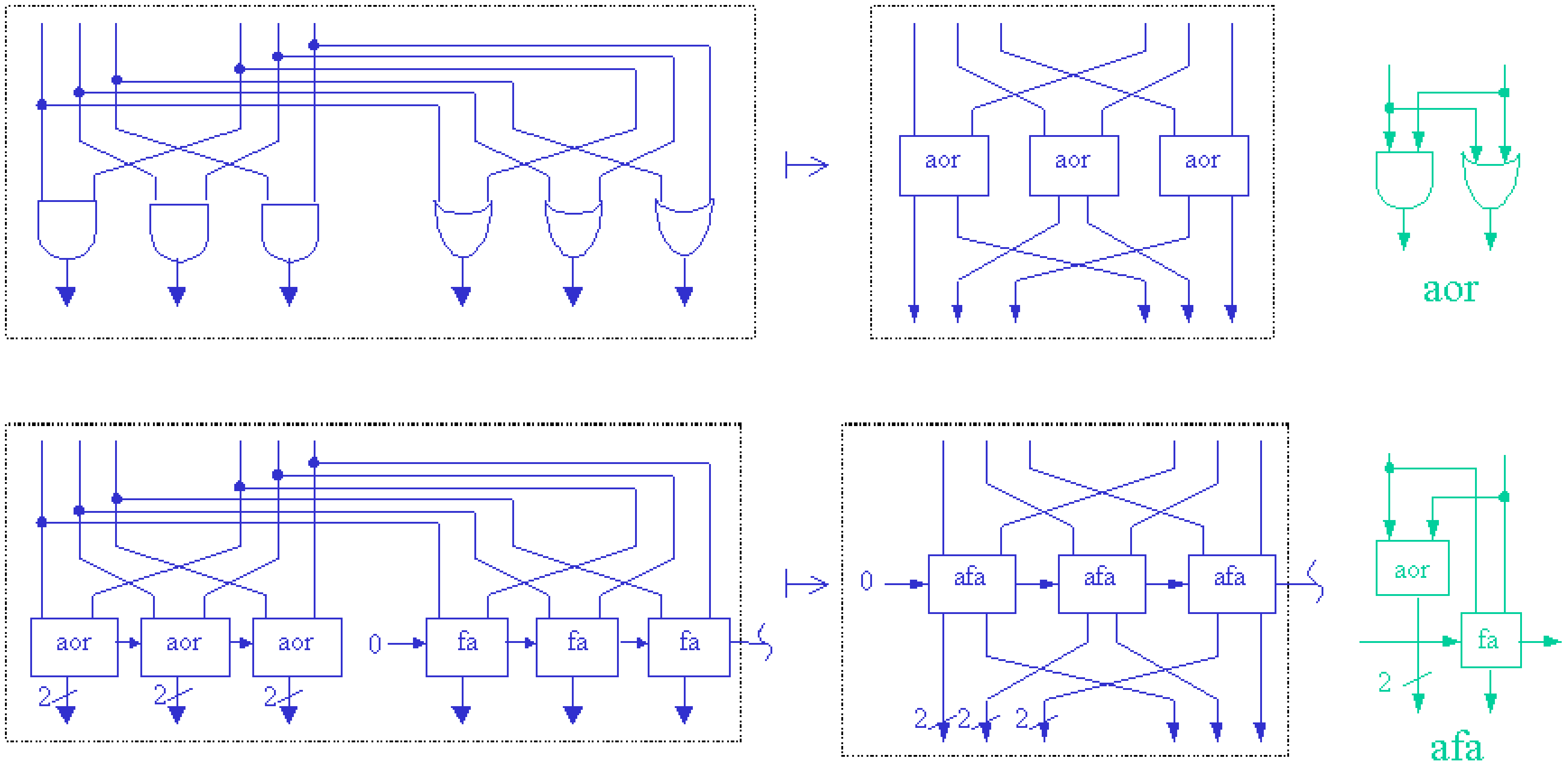
- 

- subtractor: a-b
  - $(\sum 2^i \times b_i) + (\sum 2^i \times \bar{b}_i) = -1$ e.g. $0101_2 \oplus 1010_2 = 1111_2 = -1_{10}$
  - $(\sum 2^i \times a_i) - (\sum 2^i \times b_i) = (\sum 2^i \times a_i) + (\sum 2^i \times \bar{b}_i) + 1$

    i.e. invert b bitwise and set C_{in} = 1 for adder

Deriving ALU cell by interleaving components

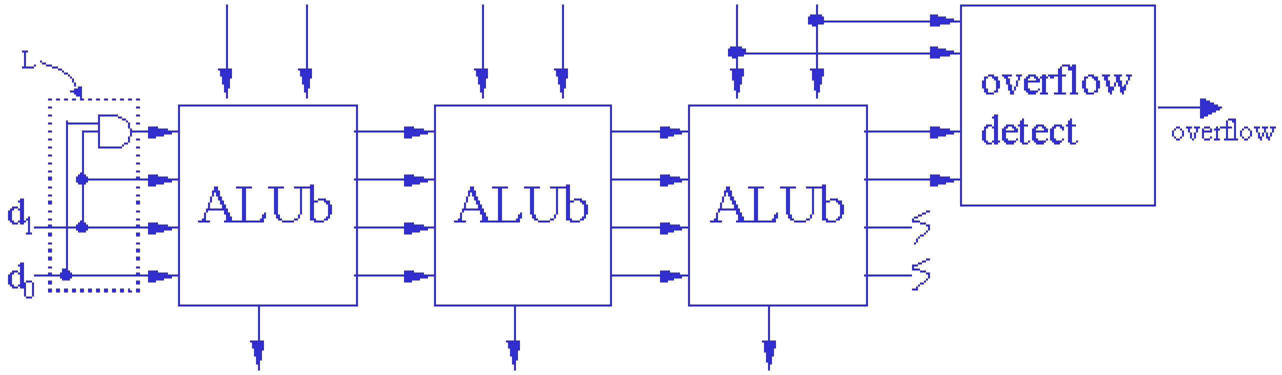
- group components together to form larger repeated unit

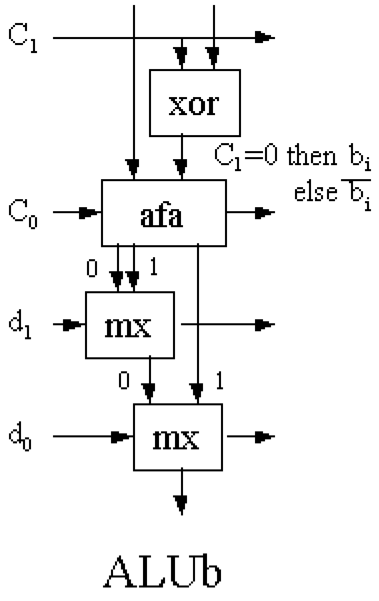


- the dotted boxes have the same function and interface

Selecting ALU operation

- programmable inverter for b_i (using xor)
- connecting mux in series

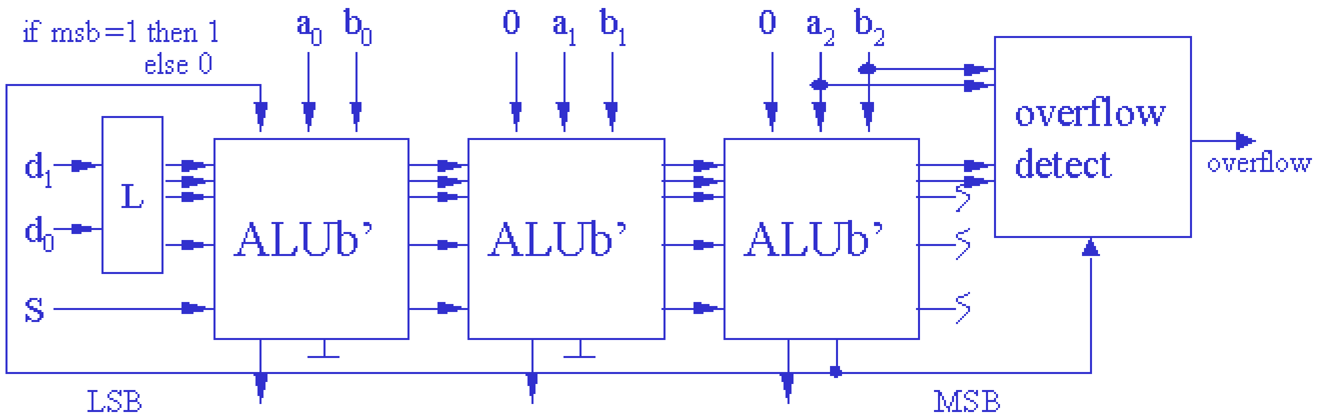


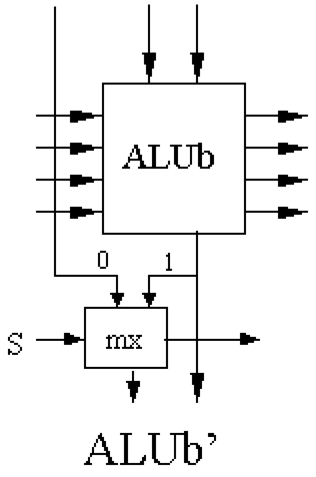


- d_0d_1: 00 and, 01 or,
  10 add, 11 subtract
- detecting overflow: exercise

Comparison operations

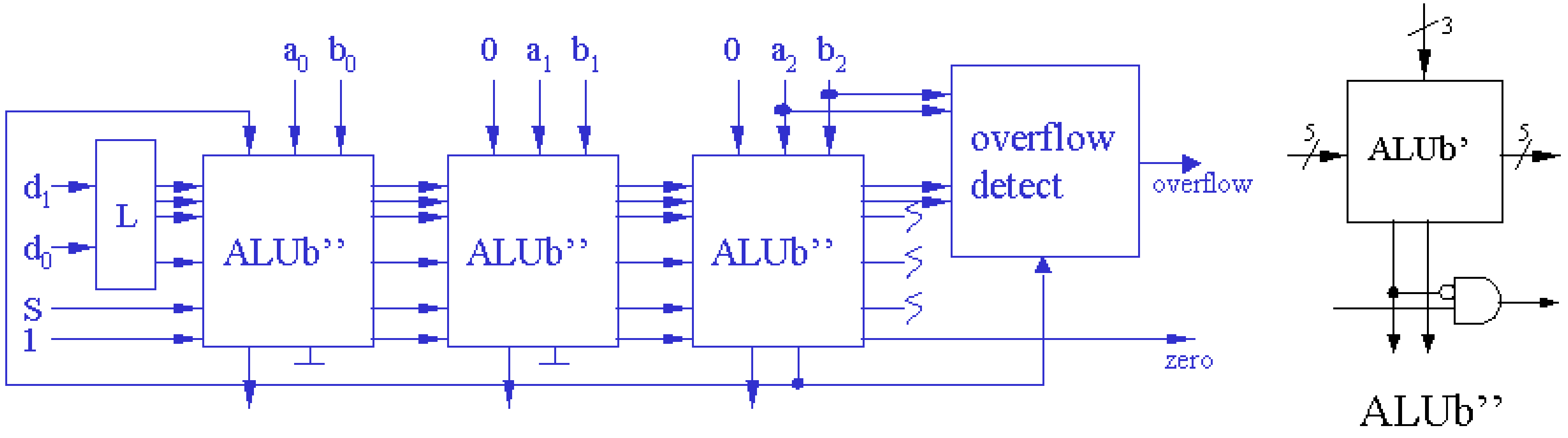
- slt: set on less than, if a < b then 1 else 0
- if a < b, a-b < 0, so MSB of (a-b) is 1
- implementation
  - provide additional input to each cell
  - LSB input from MSB ALUb output, other inputs set to 0
  - include additional mux in cell for selection
  - to select slt, s=0, d_0=1, d_1=1





Zero detector

- beq, bne: test a=b or a-b=0
- include another gate to test if output zero



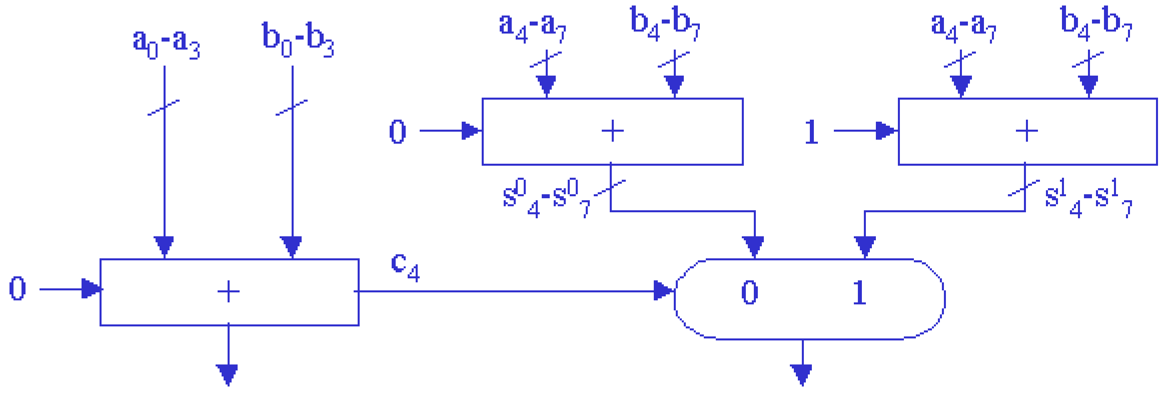
summary:	s d_0 d_1	011	100	101	110	111
	function:	set on less than	and	or	add	subtract

Performance estimation

- clocked circuit: no combinational loops
- speed limited by propagation delay through the slowest combinational path
- slowest path: usually carry path
- clock rate: approx. 1/(delay of slowest path) assuming
  - edge-triggered design
  - flip-flop propagation delay, set-up time, clock skew etc. negligible (see 3rd Ed: B.7, B.11; 4th Ed: C.7, C.11)

Fast addition

- carry select
  - compute both zero-carry-in and one-carry-in after *n* stages
  - e.g. 8 bits: use three 4-bit ripple carry adders



- other possibilities
  - carry-lookahead adder
  - conditional-sum adder