Intro. to Computer Architecture

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨csdsp@bristol.ac.uk⟩

January 9, 2018

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

- Agenda:

  1. comments, questions, recap, then
  2. some special-purpose implementation techniques and related trade-offs, namely
     - improving the functionality of a ripple-carry adder, and
     - reducing the latency of a ripple-carry adder (via **carry look-ahead** addition).

Notes:

- Recap: we produced the following algorithm for (ripple-carry) addition

Algorithm

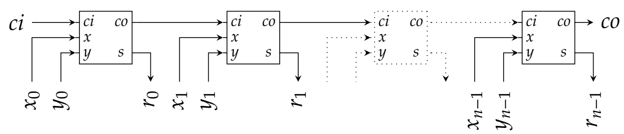
Input: Two unsigned, n-digit, base-b integers x and y, and a 1-digit carry-in $ci \in \{0, 1\}$
Output: An unsigned, n-digit, base-b integer $r = x + y$, and a 1-digit carry-out $co \in \{0, 1\}$

```
r ← 0, c_0 ← ci
for i = 0 upto n − 1 step +1 do
  | r_i ← (x_i + y_i + c_i) mod b
  | if (x_i + y_i + c_i) < b then c_{i+1} ← 0 else c_{i+1} ← 1
end
co ← c_n
return r, co
```

Notes:

- Recap: we produced the following design for (ripple-carry) addition

Circuit



based on existence of a full-adder cell.

Notes:

Boolean algebra + computer arithmetic ⇝ design trade-offs (1)

Area vs. functionality

- Question: how can we support subtraction?

Notes:

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (1)

Area vs. functionality

- Question: how can we support subtraction?
- Solution: the algorithm is basically the same

Algorithm

Input: Two unsigned, n-digit, base-b integers x and y, and a 1-digit borrow-in $bi \in \{0, 1\}$
Output: An unsigned, n-digit, base-b integer $r = x - y$, and a 1-digit borrow-out $bo \in \{0, 1\}$

```
r ← 0, c_0 ← bi
for i = 0 upto n − 1 step +1 do
  | r_i ← (x_i − y_i − c_i) mod b
  | if (x_i − y_i − c_i) ≥ 0 then c_{i+1} ← 0 else c_{i+1} ← 1
end
bo ← c_n
return r, bo
```

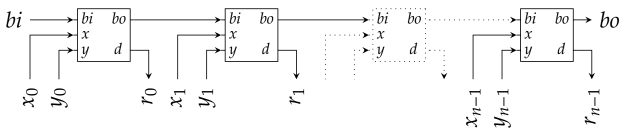

Notes:

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (1)

Area vs. functionality

- Question: how can we support subtraction?
- Solution: the design is basically the same we produced the following design

Circuit



albeit now based on existence of an alternative, full-subtractor cell.

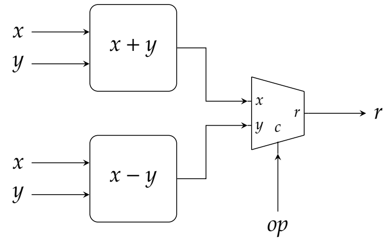
© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Intro. to Computer Architecture
git # 3b6f641 @ 2018-01-09
University of BRISTOL

Notes:

Boolean algebra + computer arithmetic ⇝ design trade-offs (2)
Area vs. functionality

- **Question**: how can we support subtraction *and* addition?
- **Solution #1**: compute *both*, then select which we want, i.e.,

Circuit



st.

$$r = \begin{cases} x + y & \text{if } op = 0 \\ x - y & \text{if } op = 1 \end{cases},$$

ignoring the carry/borrow in and out for the moment.

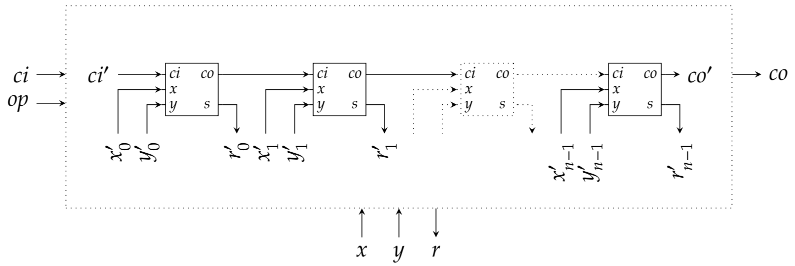
Notes:

Boolean algebra + computer arithmetic ⇝ design trade-offs (2)
Area vs. functionality

- **Question**: how can we support subtraction *and* addition?

- **Solution #2**: we know $x - y \equiv x + (-y)$ and can *already* compute $x + y$, so

Circuit



given we want

$$r = \begin{cases} x + y + ci & \text{if } op = 0 \\ x - y - ci & \text{if } op = 1 \end{cases},$$

we just need to consider how x', y' and ci' are controlled?

Notes:

Definition

The term two's-complement can be used as a noun (i.e., to describe the *representation*) or a verb (i.e., to describe an *operation*): the latter case defines "taking the two's-complement of x" to mean negating x and thus computing the representation of $-x$. To do so, we compute $-x \mapsto \neg x + 1$.

- So,

op	ci	r
0	0	$x + y + ci$
0	1	$x + y + ci$
1	0	$x - y - ci$
1	1	$x - y - ci$

Notes:

Boolean algebra + computer arithmetic $\leadsto$ design trade-offs (3)
Area vs. functionality

Definition

The term two's-complement can be used as a noun (i.e., to describe the *representation*) or a verb (i.e., to describe an *operation*): the latter case defines "taking the two's-complement of x" to mean negating x and thus computing the representation of $-x$. To do so, we compute $-x \mapsto \neg x + 1$.

- So,

op	ci	r
0	0	$x + y + 0$
0	1	$x + y + 1$
1	0	$x - y - 0$
1	1	$x - y - 1$

Notes:

Definition

The term two's-complement can be used as a noun (i.e., to describe the *representation*) or a verb (i.e., to describe an *operation*): the latter case defines "taking the two's-complement of x" to mean negating x and thus computing the representation of $-x$. To do so, we compute $-x \mapsto \neg x + 1$.

- So,

op	ci	r
0	0	$x + y + 0$
0	1	$x + y + 1$
1	0	$x + \neg y + 1 - 0$
1	1	$x + \neg y + 1 - 1$

Notes:

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (3)
Area vs. functionality

Definition

The term two's-complement can be used as a noun (i.e., to describe the *representation*) or a verb (i.e., to describe an *operation*): the latter case defines "taking the two's-complement of x" to mean negating x and thus computing the representation of $-x$. To do so, we compute $-x \mapsto \neg x + 1$.

- So,

op	ci	r
0	0	$x + y + 0$
0	1	$x + y + 1$
1	0	$x + \neg y + 1$
1	1	$x + \neg y + 0$

Notes:

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (3)

Area vs. functionality

Definition

The term two's-complement can be used as a noun (i.e., to describe the *representation*) or a verb (i.e., to describe an *operation*): the latter case defines "taking the two's-complement of x" to mean negating x and thus computing the representation of $-x$. To do so, we compute $-x \mapsto \neg x + 1$.

- So,

op	ci	r
0	0	$x + y + 0$
0	1	$x + y + 1$
1	0	$x + \neg y + 1$
1	1	$x + \neg y + 0$

and then just translate via

op	ci	x_i	y_i	ci'	x'_i	y'_i
0	0	0	0	0	0	0
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	1	1	1	0	1	0

i.e., $ci' = ci \oplus op$, $x'_i = x_i$, and $y'_i = y_i \oplus op$ ...

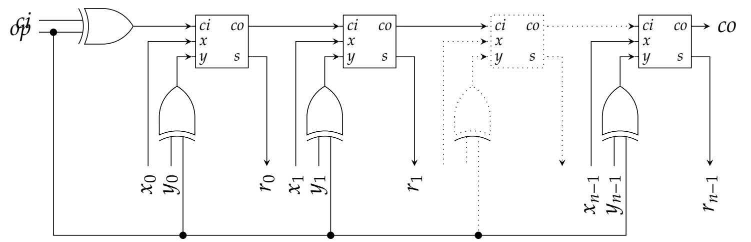
Notes:

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (4)

Area vs. functionality

- ... to yield

Circuit



Notes:

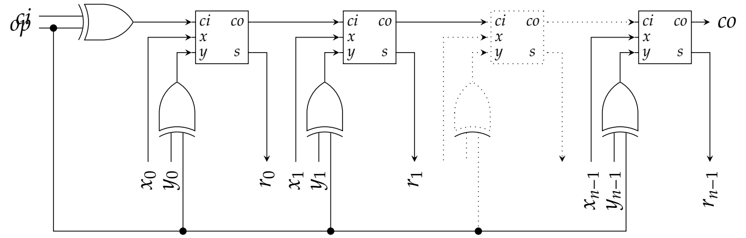
- The comparison with a single ripple-carry adder is easy to see: the combined design includes $n + 1$ additional XORs, the area related to which is traded-off against support for subtraction. In contrast, the comparison with the separate design (i.e., a ripple-carry adder plus a subtractor) is a more subtle wrt. the functionality metric. The point is that although both support subtraction, the former is clearly unable to simultaneously perform addition and subtraction, whereas the latter *might* be.

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (4)

Area vs. functionality

- ... to yield

Circuit



- Question(s):
  1. what are the design trade-offs here, and/or
  2. how do we evaluate this design vs. alternatives?
- Answer(s):

  −ve: : higher area vs. ripple-carry adder
  +ve: : greater functionality vs. ripple-carry adder

Notes:

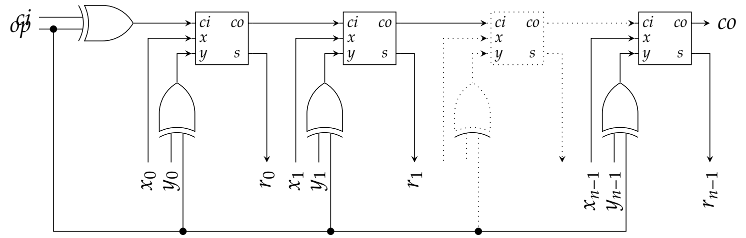
- The comparison with a single ripple-carry adder is easy to see: the combined design includes $n + 1$ additional XORs, the area related to which is traded-off against support for subtraction. In contrast, the comparison with the separate design (i.e., a ripple-carry adder plus a subtractor) is a more subtle wrt. the functionality metric. The point is that although both support subtraction, the former is clearly unable to simultaneously perform addition and subtraction, whereas the latter *might* be.

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (4)

Area vs. functionality

- ... to yield

Circuit



- Question(s):
  1. what are the design trade-offs here, and/or
  2. how do we evaluate this design vs. alternatives?
- Answer(s):

  +ve: : lower area vs. ripple-carry adder plus subtractor
  −ve: : lesser functionality vs. ripple-carry adder plus subtractor

© Daniel Page (csdsp@bristol.ac.uk)
Intro. to Computer Architecture git # 3b6f641 @ 2018-01-09
University of BRISTOL

Notes:

- The comparison with a single ripple-carry adder is easy to see: the combined design includes $n + 1$ additional XORs, the area related to which is traded-off against support for subtraction. In contrast, the comparison with the separate design (i.e., a ripple-carry adder plus a subtractor) is a more subtle wrt. the functionality metric. The point is that although both support subtraction, the former is clearly unable to simultaneously perform addition and subtraction, whereas the latter *might* be.

Definition

latency, *n. the interval between the reception of a stimulus and the response to that stimulus.*

– OED (`http://www.oed.com`)

Notes:

Definition

latency, *n. the interval between the reception of a stimulus and the response to that stimulus.*

– OED (`http://www.oed.com`)

- Question: what is the latency of ripple-carry addition?

Notes:

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (5)

Area vs. latency

Definition

latency, *n. the interval between the reception of a stimulus and the response to that stimulus.*

– OED (http://www.oed.com)

- Question: what is the latency of ripple-carry addition?
- Solution: $O(n)$.

Notes:

Boolean algebra + computer arithmetic $\rightsquigarrow$ design trade-offs (5)

Area vs. latency

Definition

latency, *n. the interval between the reception of a stimulus and the response to that stimulus.*

– OED (http://www.oed.com)

- Question: what is the latency of ripple-carry addition?
- Solution: $O(n)$.
- Question: can we *improve* this somehow?

Notes:

Definition

latency, *n. the interval between the reception of a stimulus and the response to that stimulus.*

– OED (http://www.oed.com)

- Question: what is the latency of ripple-carry addition?
- Solution: $O(n)$.
- Question: can we *improve* this somehow?
- Solution: yes ...

Notes:

- Observation: the carry-chain between cells *dictates* latency.
- Idea: *decouple* computation of the carry from the sum.
  - We know *something* about how the i-th cell behaves in isolation, i.e.,

$$\begin{array}{lllll} 1. & x_i + y_i > b - 1 & \Rightarrow & \text{it } \textit{generates} & \text{a carry} \\ 2. & x_i + y_i = b - 1 & \Rightarrow & \text{it } \textit{propagates} & \text{a carry} \\ 3. & x_i + y_i < b - 1 & \Rightarrow & \text{it } \textit{absorbs} & \text{a carry} \end{array}$$

  - For $b = 2$, imagine g_i and p_i tell us whether the stage generate or propagate a carry; then

$$\begin{array}{lcl} g_i & = & x_i \wedge y_i \\ p_i & = & x_i \oplus y_i \end{array}$$

and we have that

$$c_{i+1} = g_i \vee (c_i \wedge p_i)$$

where, again, $c_0 = ci$ and we produce a carry-out $c_n = co$.

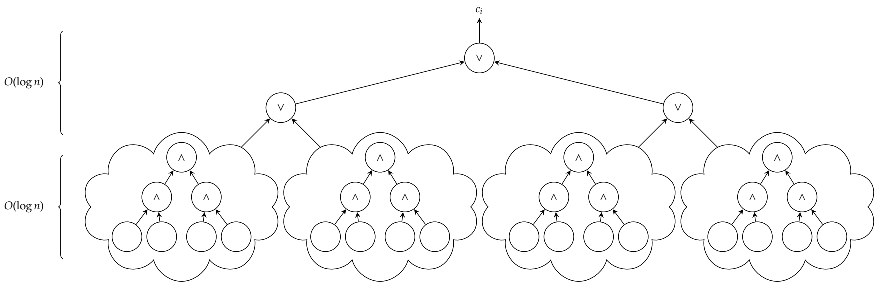
Notes:

Example

Unwinding the recursion, the carry into each cell is described by

$$
\begin{array}{lcl}
c_0 & = & ci \\
c_1 & = & g_0 \vee (ci \wedge p_0) \\
c_2 & = & g_1 \vee (g_0 \wedge p_1) \vee (ci \wedge p_0 \wedge p_1) \\
c_3 & = & g_2 \vee (g_1 \wedge p_2) \vee (g_0 \wedge p_1 \wedge p_2) \vee (ci \wedge p_0 \wedge p_1 \wedge p_2) \\
 & \vdots &
\end{array}
$$

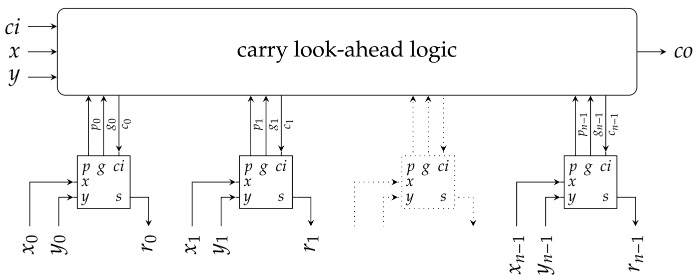
which *looks* horrendous, but notice that the general structure is something like



Notes:

- Solution: a **carry look-ahead** adder design

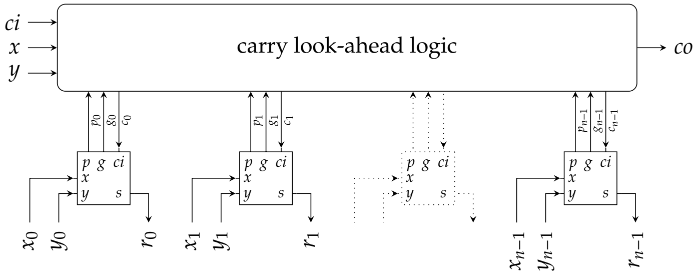
Circuit



Notes:

- Solution: a **carry look-ahead** adder design

Circuit



- Question(s):
  1. what are the design trade-offs here, and/or
  2. how do we evaluate this design vs. alternatives?
- Answer(s):

  −ve: : higher area vs. ripple-carry adder
  +ve: : lower latency vs. ripple-carry adder

Notes:

Conclusions

- Take away points:
  1. We've seen two concrete design trade-offs, namely

     $$\begin{array}{lcl} \text{higher area} & \rightsquigarrow & \text{greater functionality} \\ \text{higher area} & \rightsquigarrow & \text{lower latency} \end{array}$$

     noting that each design was still *correct*.
  2. To produce the associated designs, we've had to understand
     - the problem domain (e.g., computer arithmetic),
     - the problem specification (e.g., design constaints, such as a preference for low latency),
     - the resources and techniques available (e.g., Boolean operators)
     - ...

     which is subtle, but critical: the best results are produced by application of richer understanding (of this unit, CS in general, and beyond).

Notes:

Additional Reading

- *Wikipedia: Computer Arithmetic*. URL: `http://en.wikipedia.org/wiki/Category:Computer_arithmetic`.
- D. Page. "Chapter 7: Arithmetic and logic". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer-Verlag, 2009.
- B. Parhami. "Part 2: Addition/subtraction". In: *Computer Arithmetic: Algorithms and Hardware Designs*. 1st ed. Oxford University Press, 2000.

Notes:

References

[1] *Wikipedia: Computer Arithmetic*. URL: `http://en.wikipedia.org/wiki/Category:Computer_arithmetic` (see p. 57).

[2] D. Page. "Chapter 7: Arithmetic and logic". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer-Verlag, 2009 (see p. 57).

[3] B. Parhami. "Part 2: Addition/subtraction". In: *Computer Arithmetic: Algorithms and Hardware Designs*. 1st ed. Oxford University Press, 2000 (see p. 57).

Notes: