Chapter8. Architecture & Components

2019年12月10日 14:07

NOTE TAKING AREA

Computer basic

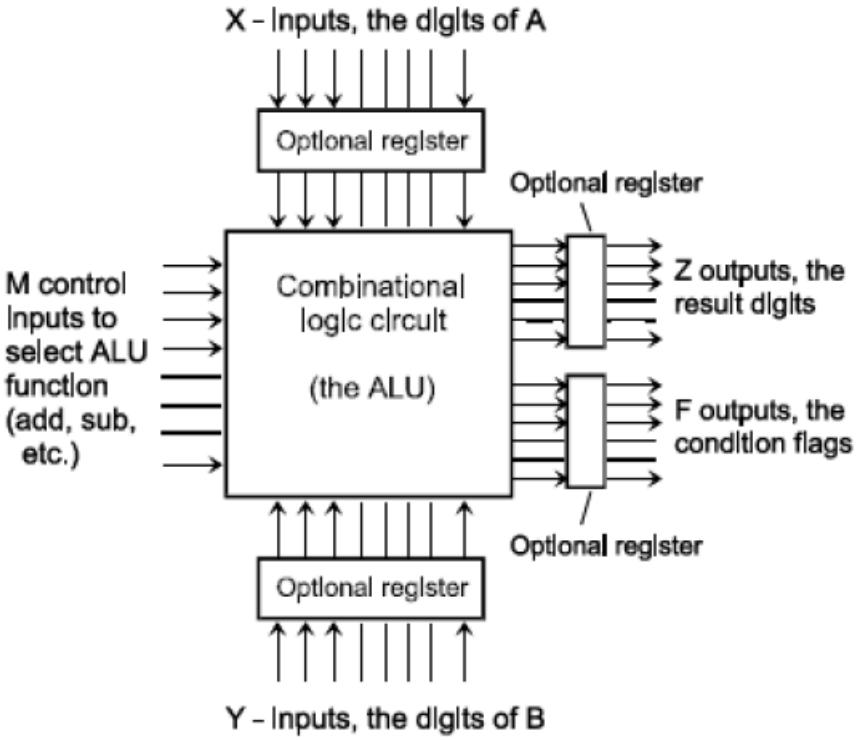
Components of a computer: I/O device, memory, CPU (control circuits, registers, instruction decoder, arithmetic and local circuits (ALU), clock).

- Computer architecture: user's view. Instruction set, visible registers, memory management table structure, exception handling model.
- Computer organization: user-invisible implement of architecture. Pipeline structure, transparent cache, table-walking hardware, translation look-aside buffer.

Computer architecture

I/O can be integrated within the memory system.
Von Neumann system: has a von Neumann architecture, follow the von Neumann operating sequence.
CPU components: ALU, registers, datapaths, multiplexers & demultiplexers, sequential circuits, clock.

ALU circuit design



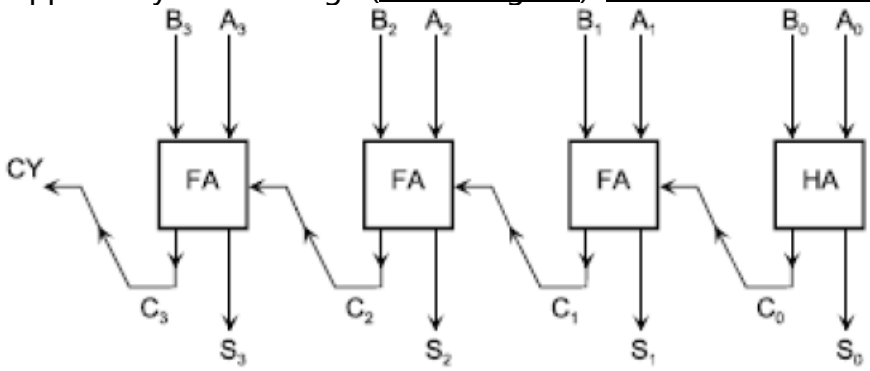
Design of large combinational circuits: modular methods (cell technique, used by ARM7), convert to a sequential circuit.

ALU adder design

Example of addition and carry flag:

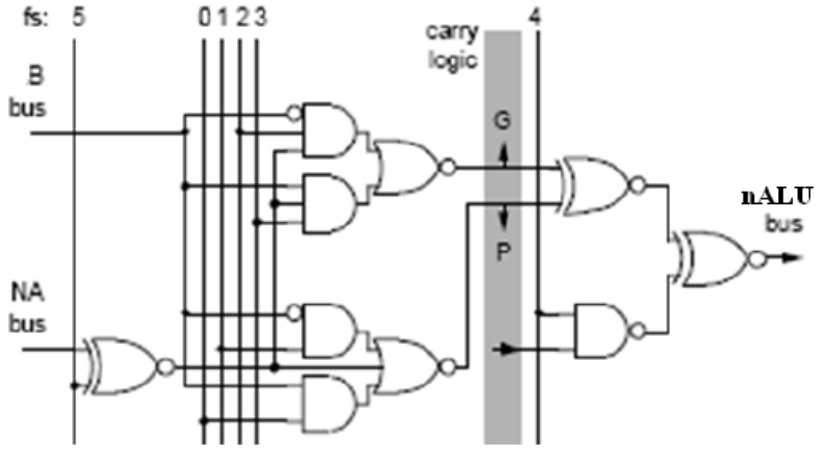
3 2 1 0	← Bit number
0 1 1 0	Addend (digits of A)
0 0 1 1	Augend (digits of B)
0 1 1 0	Carry to next column
1 0 0 1	Sum

The carry flag would be set if MSB column is carry.
Taking each column separately: LSB column, next column (carry out from previous column & carry in from this column).
Ripple carry adder design (block diagram): half adder & full adder.

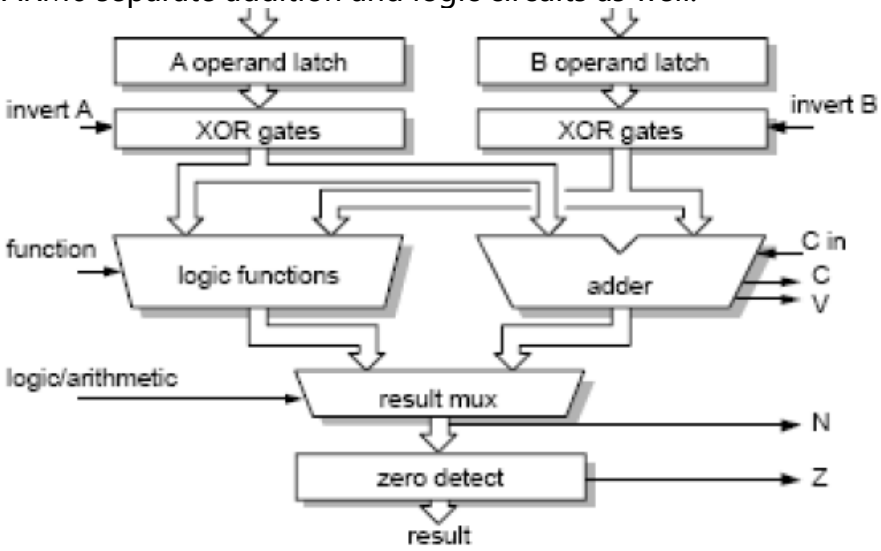


Disadvantage of ripple carry adder design is delay.
Carry out logic: use G and P, $G = 1\ if\ A = 1\ \&\ B = 1$, $P = 1\ if\ A = 1 \mid B = 1$.
ARM1 uses ripple carry, ARM2 uses look-ahead, ARM6 uses carry out.

ALU logic function design
Logic functions can be combined in the same circuit as **ripple carry adder**.
ARM2 needs a separate logic function block:



Function selection (fs) decides the operation.
ARM6 separate addition and logic circuits as well:



XOR gates can give 1's complement of input (invert signal).

Other arithmetic functions
Subtraction uses 2's complement addition, or uses 1's complement addition with carry in to 1.

Multiplication circuits: add each partial product into a total as it is formed.

			A_3	A_2	A_1	A_0			Multiplicand
			B_3	B_2	B_1	B_0			Multiplier
			$A_3.B_0$	$A_2.B_0$	$A_1.B_0$	$A_0.B_0$			First partial product
		$A_3.B_1$	$A_2.B_1$	$A_1.B_1$	$A_0.B_1$				Second partial product
		PS	PS	PS	S_1	$A_0.B_0$			Partial sum
	$A_3.B_2$	$A_2.B_2$	$A_1.B_2$	$A_0.B_2$					Third partial product
	PS	PS	PS	S_2	S_1	$A_0.B_0$			Partial sum
$A_3.B_3$	$A_2.B_3$	$A_1.B_3$	$A_0.B_3$						Fourth partial product
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0		Total of partial products

Matrix multiplier: adding each partial sum generated by adder.

Need 32*32 AND gates and 31 adders for 32 bit numbers.

Sequential implementation of multiplier: shift and add.

Need 32*32 AND gates and 31 adders for 32 bit numbers.

Booth's algorithm: improved rule.

Initialization

1. Set a running total, T, to zero
2. Put the multiplier in a special register M
3. Set a carry bit, C_{in}, to 0
4. Set a cycle counter, N, to 0

THIS is the LOOP ENTRY point

5. Take the two LSBs of M, call this value B and add value of C_{in} to B
6. **Next action is one of**
   i. if C_{in} + B = 000_2, do nothing and set C_{out} = 0.
   ii. if C_{in} + B = 001_2, left shift the multiplicand 2N places & add this to running total T, set C_{out} = 0
   iii. if C_{in} + B = 010_2, left shift the multiplicand 2N+1 places & add this to running total T, set C_{out} = 0
   iv. if C_{in} + B = 011_2, left shift the multiplicand 2N places & **subtract** this from running total T, set C_{out} = 1
   v. if C_{in} + B =100_2, do nothing and set C_{out} = 1.
7. Add 1 to N and copy C_{out} to C_{in}.
8. Right shift M two places (discarding bits shifted out)
9. If not dealt with all bits of multiplier go back to step 5.

Replacing *3 by *(4-1) and can be completed in N/2 cycle.

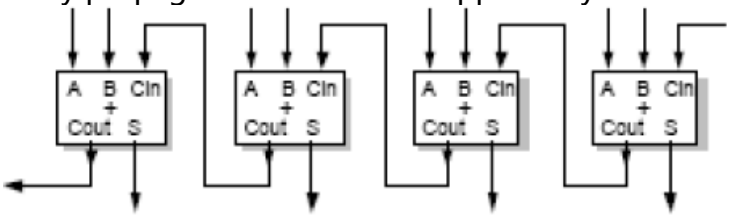
The action in N-th cycle table:

C_{in}	Multiplier	LSL #	ALU	C_{out}
0	$\times 00_2$	-	A+0	0
0	$\times 01_2$	2N	A+B	0
0	$\times 10_2$	(2N+1)	A+B	0
0	$\times 11_2$	2N	A–B	1
1	$\times 00_2$	2N	A+B	0
1	$\times 01_2$	(2N+1)	A+B	0
1	$\times 10_2$	2N	A–B	1
1	$\times 11_2$	-	A+0	1

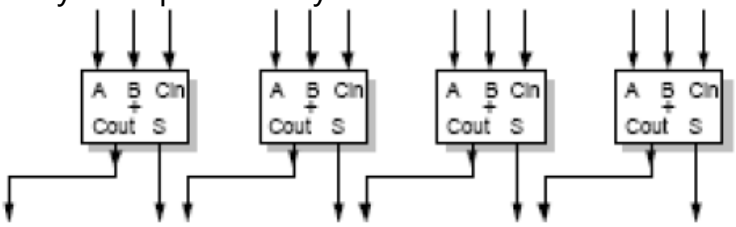
Booth's algorithm also support negative numbers.

Further performance improvements: adds 5 32 bit numbers together (carry save).

- Carry propagate: identical to a ripple carry adder.



- Carry save: passes carry value onto the next adder stage.



Production length: $\boldsymbol{n\ bit \times m\ bit = (n+m-1)\ bit\ or\ (n+m)\ bit.}$

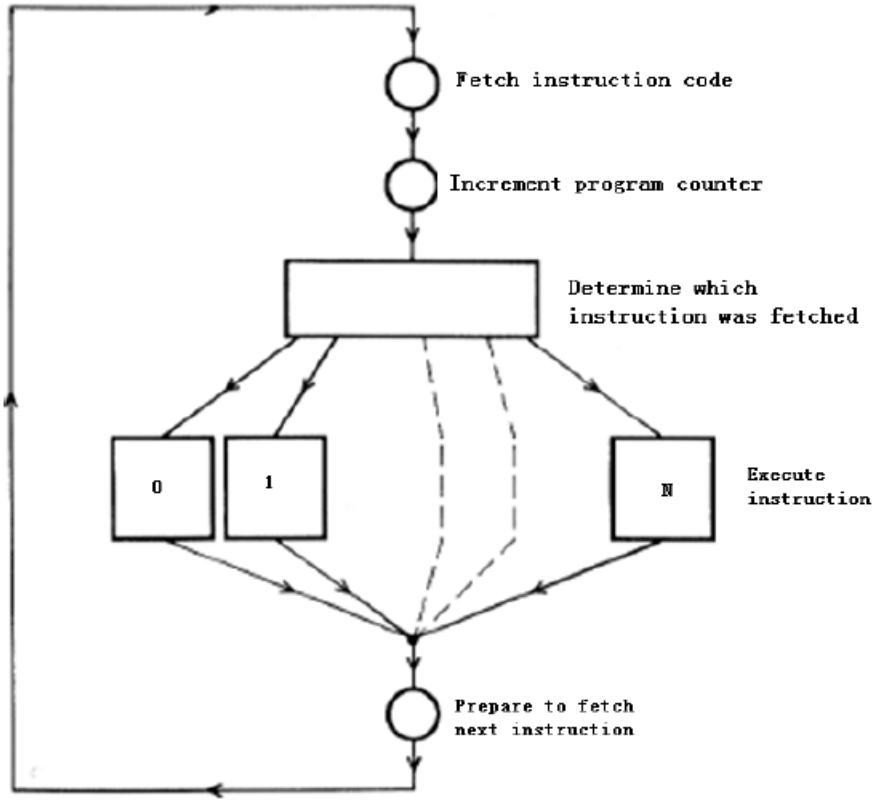
ARM support long unsigned production $UMULL$ and signed $SMULL$.

ARM multiplier uses Booth's algorithm with a dedicated carry save adder to complete a 32 bit multiplication in 5 cycles.

If there are leading 0, the computation can complete less than 5 cycles.

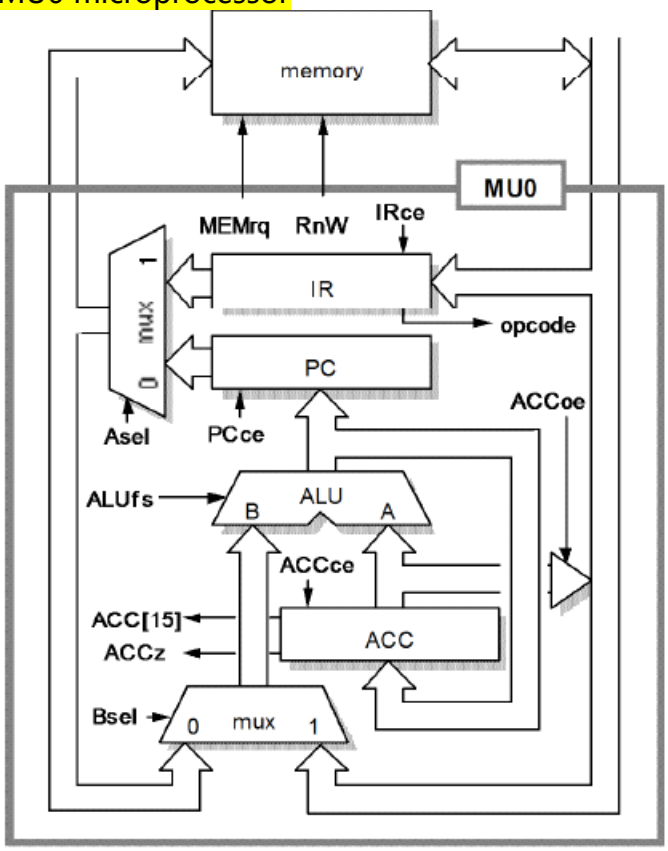
CUE COLUMN

Finite state machine of von Neumann system



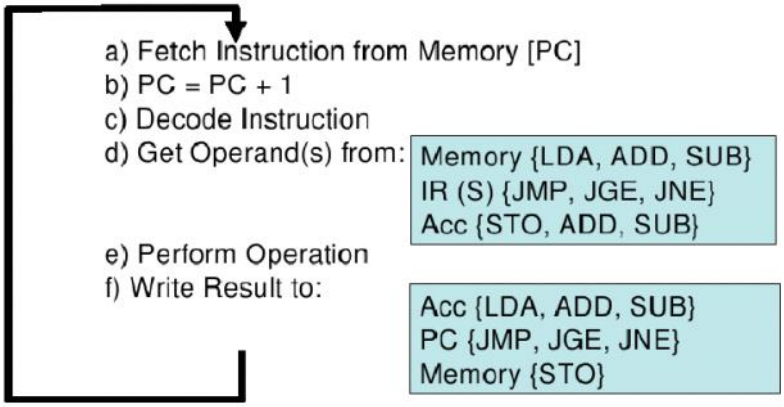
The most complex part is execution stage.
Fetch -> decode -> execute.

MU0 microprocessor



With 3 internal registers (ACC, IR, PC), 12 bit address space, instructions are 16 bits long with 4 bit opcode and 12 bit address word.
The control logic: everything else (decode, control, etc.) use FSM approach.

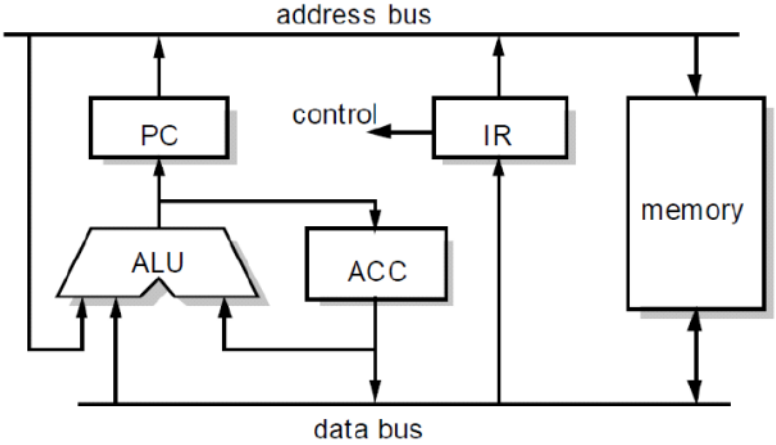
Execution sequence of MU0:



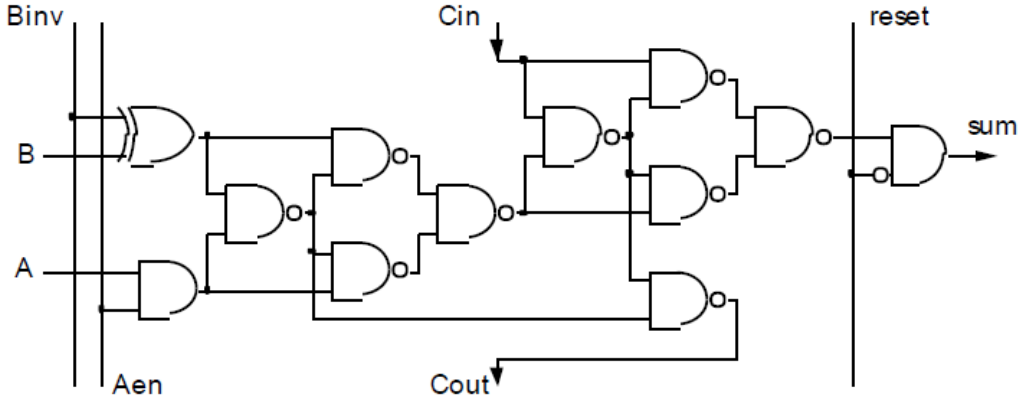
The MU0 instruction set:

Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

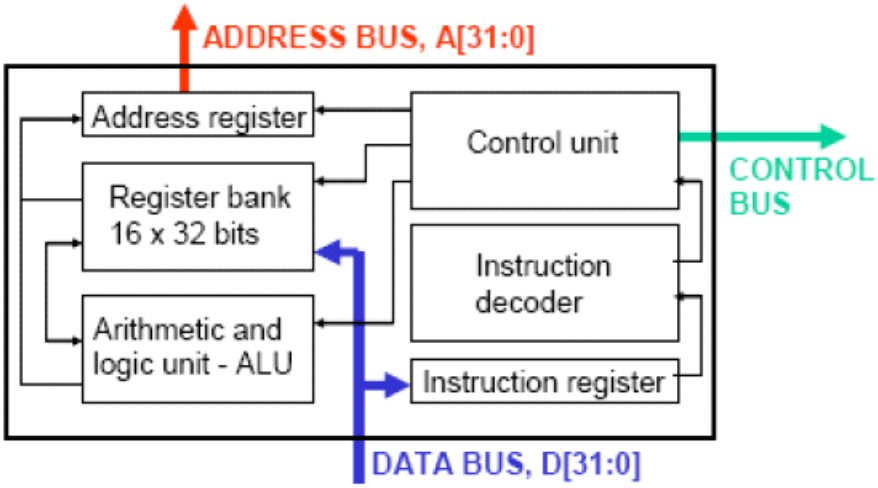
MU0 datapath example:



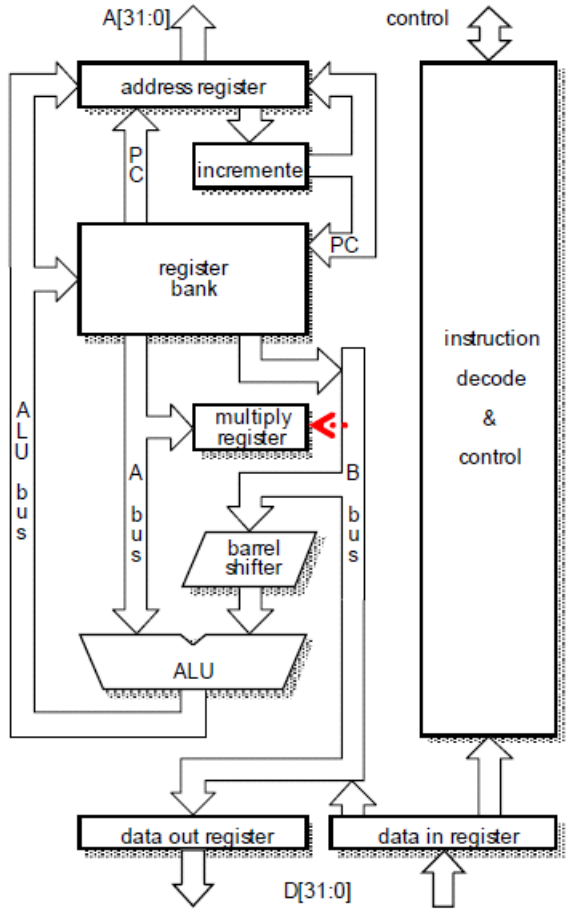
MU0 ALU logic for 1 bit:



ARM7 CPU architecture



The architecture details:



With internal registers, address register and register bank.
3 internal datapaths: A bus, B bus, ALU bus.
Additional ALU functionality: multiplier, barrel shifter, adder, logic functions.

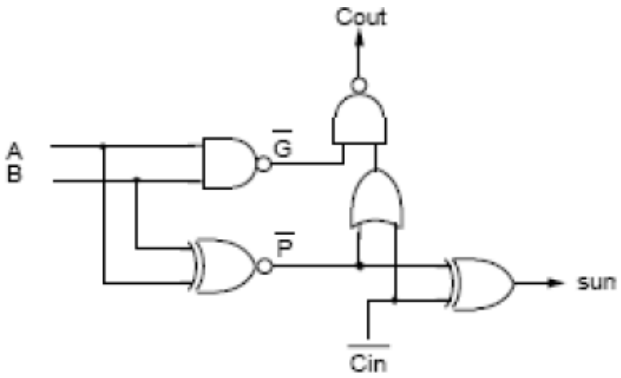
One bit full adder circuit

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$Sum = (A \oplus B) \oplus Cin$$

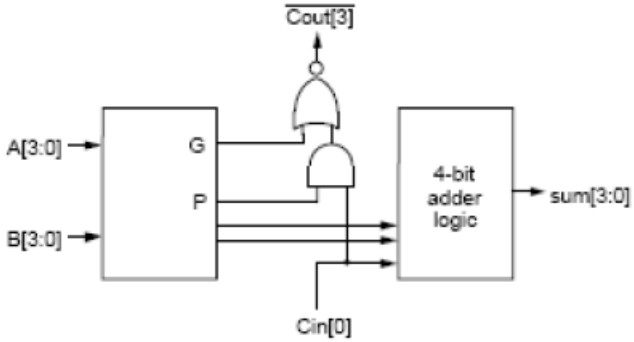
$$Cout = A.B + A.Cin + B.Cin$$

ARM1 ripple-carry adder circuit:



$$Cout = \overline{\overline{G}.(\overline{P} + \overline{Cin})}$$

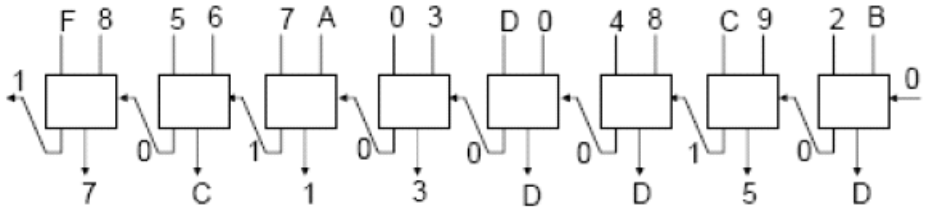
ARM2 4-bit carry look-ahead:



$G = 1\ if\ (A + B) > 15, P = 1\ if\ (A + B) = 15$

Carry look-ahead example:

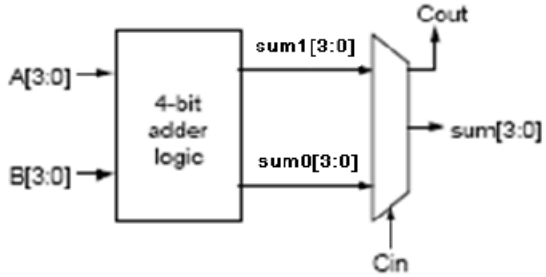
For example adding 0xF570D4C2 to 0x86A3089B (with Cin = 0):



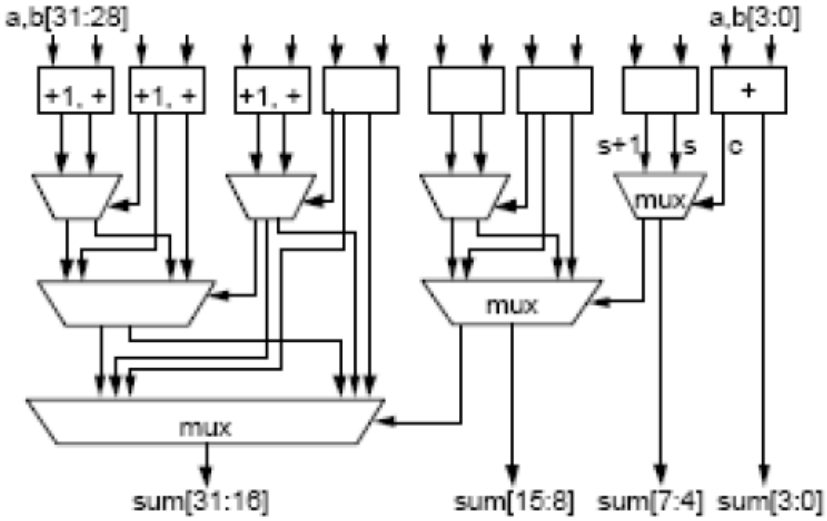
The sum is 0x7C13DD5D with Cout = 1.

ARM6 carry select adder:

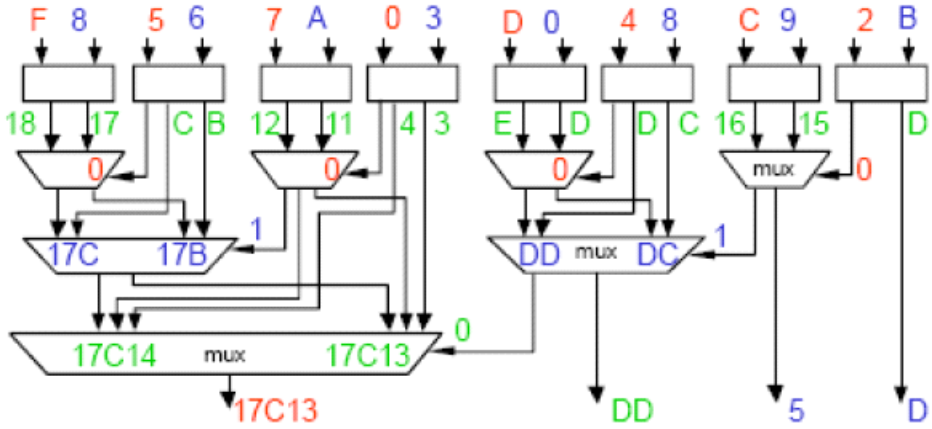
Further reduce using carry select scheme, produces 2 result: A+B and A+B+1.



The 32 bit adder carry select adder diagram (with max 3 propagation delay):



A further example, use 32 bit carry select adder:
For example adding 0xF570D4C2 to 0x86A3089B:



The sum is 0x7C13DD5D with Cout = 1.

The performance comparison of 3 adders:

Size of adder	Ripple carry	Look ahead	Carry select
4 bits	4	1	1
8 bits	8	2	1
16 bits	16	4	2
32 bits	32	8	3
64 bits	64	16	4

ARM2 ALU function codes

Note that the table given in Furber (Table 4.1) is incorrect.

Also in Furber Fig. 4.12, the NB (not B) bus should be B bus and the ALU output is inverted (nALU).

fs5	fs4	fs3	fs2	fs1	fs0	ALU output
0	0	0	1	0	0	A and B
0	0	1	0	0	0	A and not B
0	0	1	0	0	1	A xor B
0	1	0	1	1	0	A plus not B plus carry
0	1	1	0	0	1	A plus B plus carry
1	1	1	0	0	1	not A plus B plus carry
0	0	0	0	0	0	A
0	0	0	0	0	1	A or B
0	0	0	1	0	1	B
0	0	1	0	1	0	not B
0	0	1	1	0	0	zero

Example of sequential multiplier

$200_{10} \times 100_{10} = 1100\ 1000_2 \times 0110\ 0100_2$

In each step, one bit of the multiplier $0110\ 0100_2$ is selected.

If the bit is logic 1 the multiplicand, $1100\ 1000_2$ is shifted left and added to the running total.

LSL	Multiplier	ALU	Running total
0	×0	A+0	0000 0000 0000 0000
1	×0	A+0	0000 0000 0000 0000
2	×1	A+B	0000 0011 0010 0000
3	×0	A+0	0000 0011 0010 0000
4	×0	A+0	0000 0011 0010 0000
5	×1	A+B	0001 1100 0010 0000
6	×1	A+B	0100 1110 0010 0000
7	×0	A+0	0100 1110 0010 0000

Example of Booth's algorithm

$200_{10} \times 100_{10} = \ldots0000\ 1100\ 1000_2 \times \ldots0000\ 0110\ 0100_2$

The multiplier, $0110\ 0100_2$ is broken into 2 bit blocks and depending upon the value of the bits, one of four actions are implemented.

The multiplicand, $1100\ 1000_2$, is shifted and added onto the running total.

N	C_{in}	Multiplier	LSL	ALU	C_{out}	Running total
0	0	$\times 00_2$	-	A+0	0	0000 0000 0000 0000 0000 0000 0000 0000
1	0	$\times 01_2$	#2	A+B	0	0000 0000 0000 0000 0000 0011 0010 0000
2	0	$\times 10_2$	#5	A+B	0	0000 0000 0000 0000 0001 1100 0010 0000
3	0	$\times 01_2$	#6	A+B	0	0000 0000 0000 0000 0100 1110 0010 0000

In which A is the running total, initially to be 0, B is multiplicand instead of LSB 2 bits of multiplier.

Another complex example:

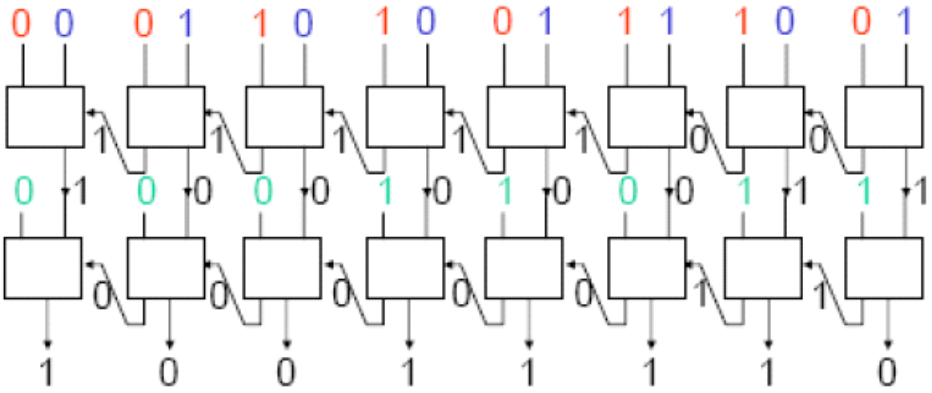
$100_{10} \times 743_{10} = \ldots0000\ 0110\ 0100_2 \times \ldots0010\ 1110\ 0111_2$

N	C_{in}	Multiplier	LSL	ALU	C_{out}	Running total
0	0	$\times11_2$	#0	A–B	1	1111 1111 1111 1111 1111 1111 1001 1100
1	1	$\times01_2$	#3	A+B	0	0000 0000 0000 0000 0000 0010 1011 1100
2	0	$\times10_2$	#5	A+B	0	0000 0000 0000 0000 0000 1111 0011 1100
3	0	$\times11_2$	#6	A–B	1	1111 1111 1111 1111 1111 0110 0011 1100
4	1	$\times10_2$	#8	A–B	1	1111 1111 1111 1111 1001 0010 0011 1100
5	1	$\times00_2$	#10	A+B	0	0000 0000 0000 0001 0010 0010 0011 1100

Carry save and carry propagation example

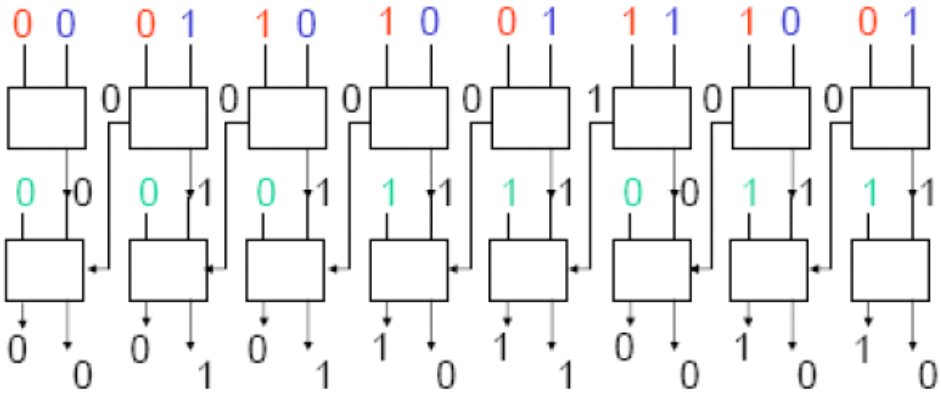
Carry propagation adder example:

In the following example X = 00110110, Y = 01001101 and Z = 00011011. This structure suffers the same problem as the ripple carry adder; multiple propagation delays.



Carry save adder example:

If X = 00110110, Y = 01001101 and Z = 00011011 then the partial sum S = 01101000 & the partial carry C = 00011011. C is left shifted once and added to S; (S + 2C) = 10011110.



SUMMARIES

1. Computer basic: computer architecture and computer organization;
2. Computer architecture and components;
3. ALU circuit design for adder, logic, subtraction, and multiplication.

- Adder: ripple carry, look-ahead, carry out;
- Logic: same as ripple carry adder or separate;
- Subtraction: use complements;
- Multiplication: matrix multiplier, sequential multiplier (Booth's algorithm), and product length.