CSSE2010/CSSE7201
Lecture 9

From Digital Logic to Processors - ALU

School of Information Technology and Electrical Engineering
The University of Queensland

Outline

- Recap of what's covered so far
- Part of a CPU – mainly ALU

What We've Covered So Far

Binary number representation

- Unsigned/signed
- Two's complement
- Binary addition

Combinational logic circuits

- Adder/subtractors
- MUX/DEMUX
- Encoders/decoders

Basic logic gates

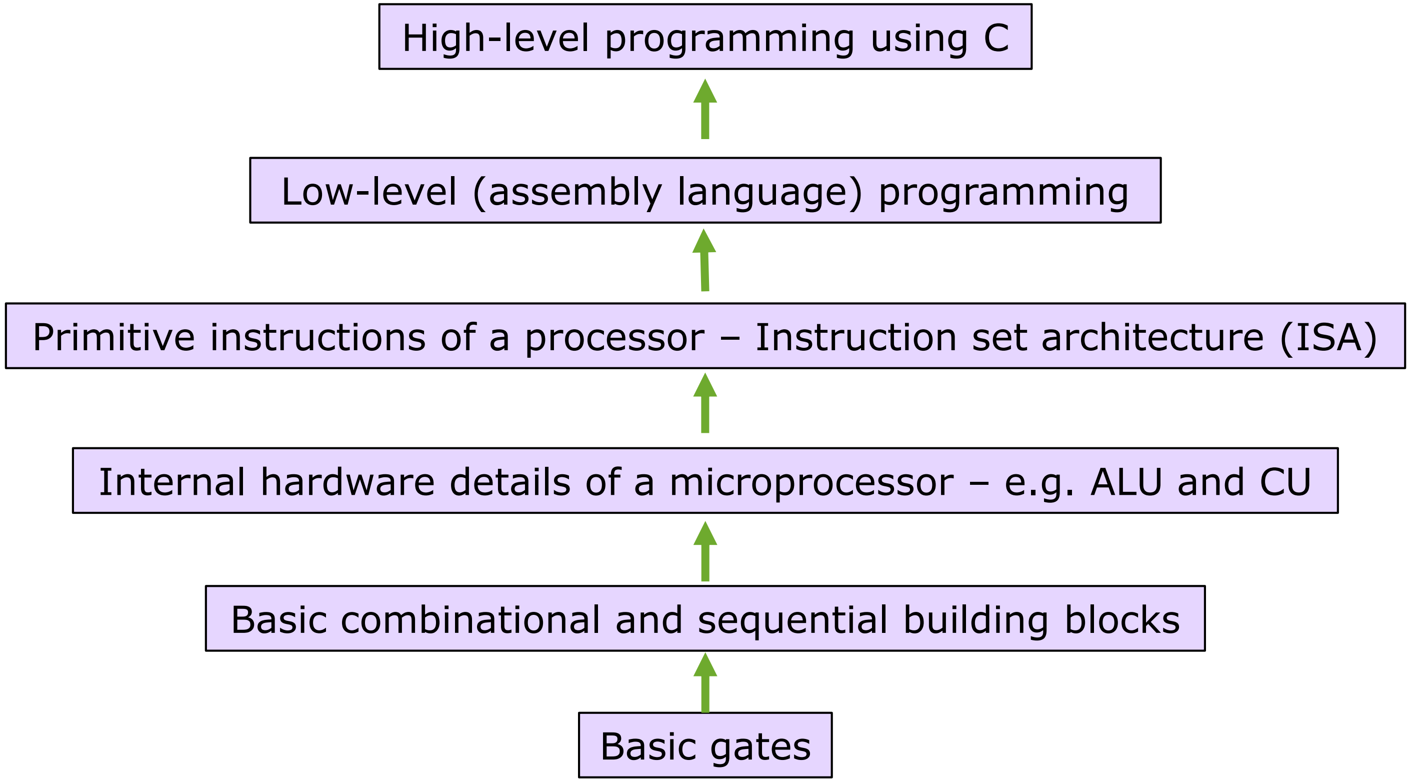
- NOT/AND/OR/NAND/NOR/XOR/XNOR
- Boolean expressions
- Logic diagrams and schematic diagrams
- Truth tables, timing diagrams

Sequential logic circuits

- Latches and Flip-flops
- Shift registers
- Counters
- State machines

We are now in a position to look at internal hardware details of a microprocessor

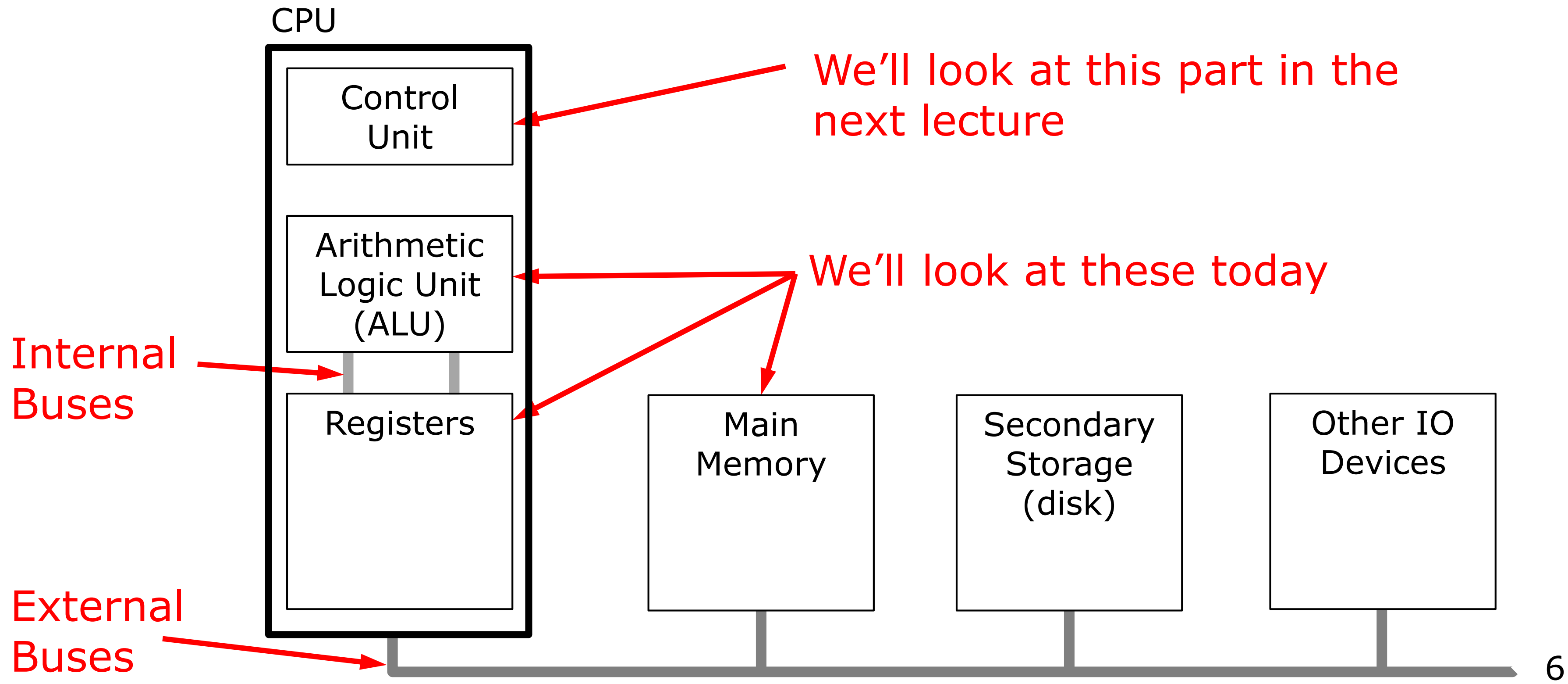
Where Are We Heading To?


High-level programming using C
Low-level (assembly language) programming
Primitive instructions of a processor – Instruction set architecture (ISA)
Internal hardware details of a microprocessor – e.g. ALU and CU
Basic combinational and sequential building blocks
Basic gates

Wk	Day	Date	Lecture (date)		Learning Lab		Assessment due
1	Mon-Tue	26-27 Jul	1	Course Introduction and Binary numbers (26 Jul)		No labs (P2 sessions) on Mon-Tue on week 1	
	Wed-Fri	28-30 Jul	2	Logic gates (28 Jul)	1	Signed number representations	
2	Mon-Tue	2-3 Aug	3	Binary Arithmetic (2 Aug)	2	Logic Gates	Quiz 1 (4pm, 6 Aug)
	Wed-Fri	4-6 Aug	4	Combinational Logic (4 Aug)	3	Binary Arithmetic	
3	Mon-Tue	9-10 Aug	5	Flip flops (9 Aug)	4	Combinational Logic	Quiz 2 (4pm, 13 Aug)
	Wed-Fri	12-13 Aug	6	Sequential Circuits 1 - Shift Registers (11 Aug) Pre-recorded lecture due to public holiday on 11/08/21	5	Flip flops	
4	Mon-Tue	16-17 Aug	7	Sequential Circuits 2 - Counters (16 Aug)	6	Sequential Circuits 1	Quiz 3 (4pm, 20 Aug)
	Wed-Fri	18-20 Aug	8	Sequential Circuits 3 - State Machines (18 Aug)	7	Sequential Circuits 2	
5	Mon-Tue	23-24 Aug	9	From Digital Logic to Processors, ALU (23 Aug)	8	Sequential Circuits 3	Quiz 4 (4pm, 27 Aug)
	Wed-Fri	25-27 Aug	10A	Memory and Control Unit (25 Aug)		Cath-up labs, Assignement 1 release end of week 5	
6	Mon-Tue	30-31 Aug	10B	Introduction to C language (pre-recorded) Review session for Assignment 1 part 1 (30 Aug)		Assignment 1 working time Part 1: 1-hour timed quiz Part 2: Digital logic design	Assignment 1 - part 1 due date (4pm, 3 Sep)
	Wed-Fri	01-03 Sep	11	Introduction to AVR and Assembly Language (1 Sep)			
7	Mon-Tue	06-07 Sep	12	Instruction Set Architecture 1 (06 Sep)	9	C Programming	Assignment 1 - part 2 submission (4pm, 6 Sep) Quiz 5 (4pm, 10 Sep)
	Wed-Fri	08-10 Sep	13	Instruction Set Architecture 2 (08 Sep)	10	Atmel Assembly	
8	Mon-Tue	13-14 Sep	14	C for AVR Microcontroller (13 Sep)	11	AVR Hardware Introduction	Quiz 6 (4pm, 17 Sep)
	Wed-Fri	15-17 Sep	15	Flow Control and AVR Timers (15 Sep)	12	AVR C Programming	
9	Mon-Tue	20-21 Sep	16	Pulse Width Modulation (PWM) (20 Sep)	13	AVR Timers	Quiz 7 (4pm, 24 Sep)
	Wed-Fri	22-24 Sep	17	Interrupts (22 Sep)	14	AVR PWM	

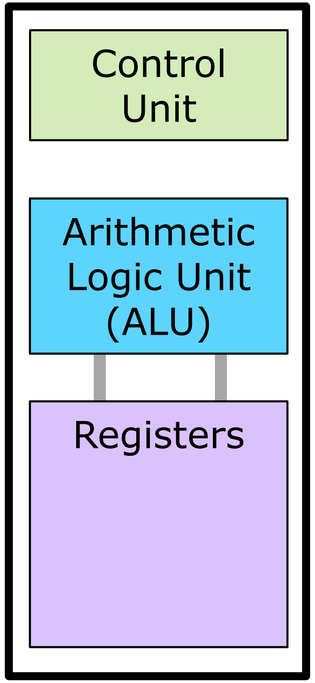
Computer Organisation

- **Simplified** view:



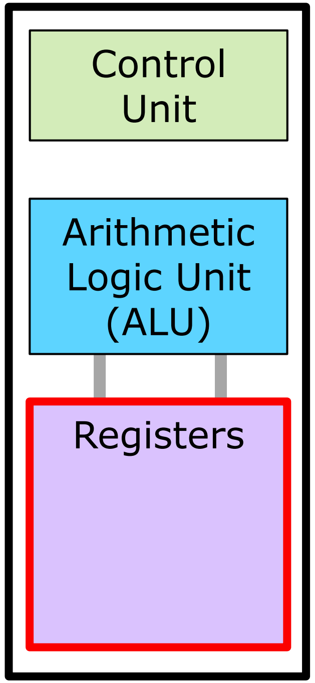
Parts of a CPU

- **Control Unit**
  - Fetches instructions from memory, makes the ALU and the registers perform the instruction
  - →The topic of next lecture
- **ALU**
  - Performs arithmetic and logical operations
- **Registers**
  - High speed memory – stores temporary results and control information
  - e.g. small CPU may have 8 x 8-bit registers
  - large CPU may have 64 x 32-bit registers



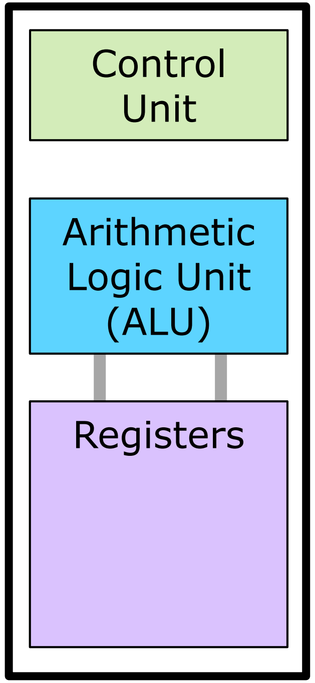
Registers

- Registers are temporary storage locations inside the CPU.
- Two types of registers
  - **General Purpose Registers**
    - Contains data to be operated on (e.g. data read from memory), results of operations,
    - Also known as register file. Width is the CPU word size
  - **Special Purpose Registers**
    - Used by the system
    - Program Counter (PC) – stores the address of the next instruction to be fetched from the memory
    - Instruction Register (IR) – stores the current instruction



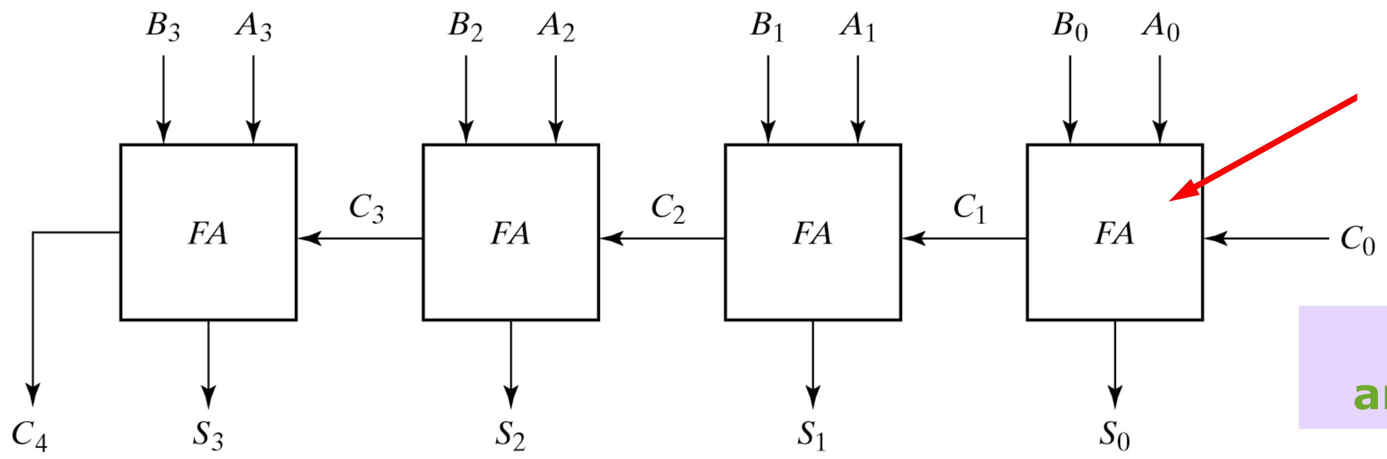
Buses

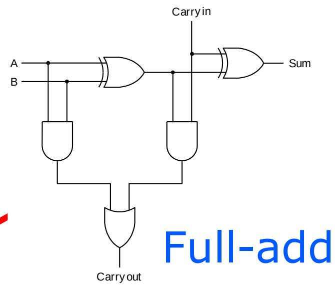
- **Bus** = Common pathway (collection of "wires") connecting parts of a computer
- Characteristics
  - Can be **internal** to CPU (e.g. ALU to registers)
  - Can be **external** to CPU (e.g. CPU to memory)
  - Buses have a **width** – number of bits that can be transferred together over a bus
    - e.g. 1-bit, 8-bits or 32-bits
    - May not always be the same as the word size of the computer



Binary Adder (From Three Weeks Ago)

- Can cascade full adders to make binary adder
  - Example: for 4 bits…





Full-adders

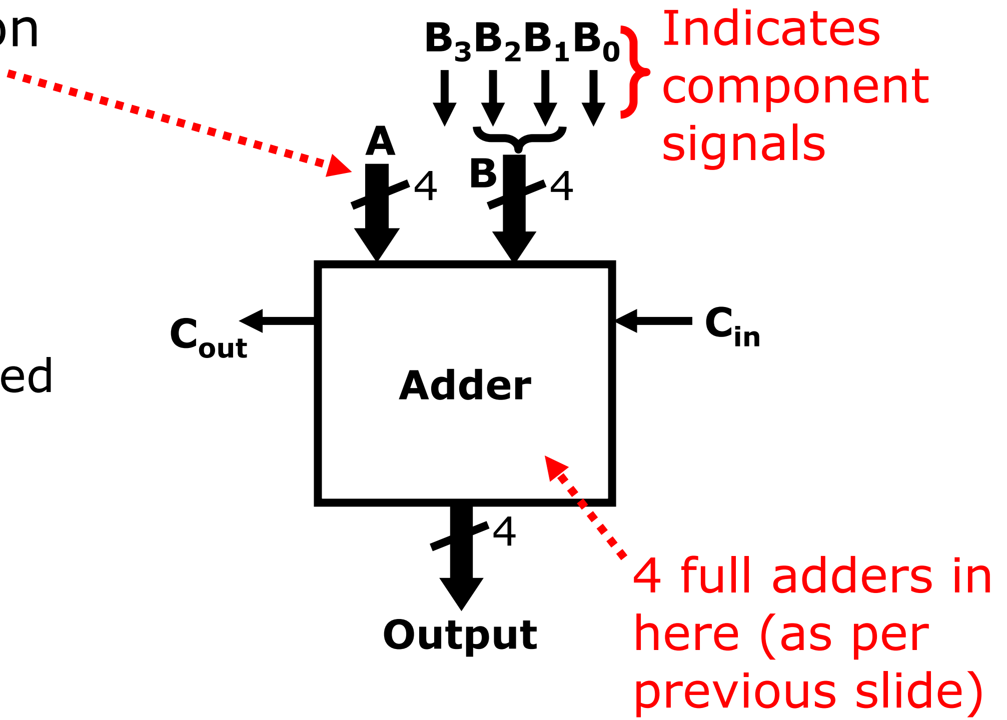
Bit-slice architecture

- This is a **ripple-carry adder**

[Figure from Mano, "Digital Design", 3rd edition]

Simplifying Representation

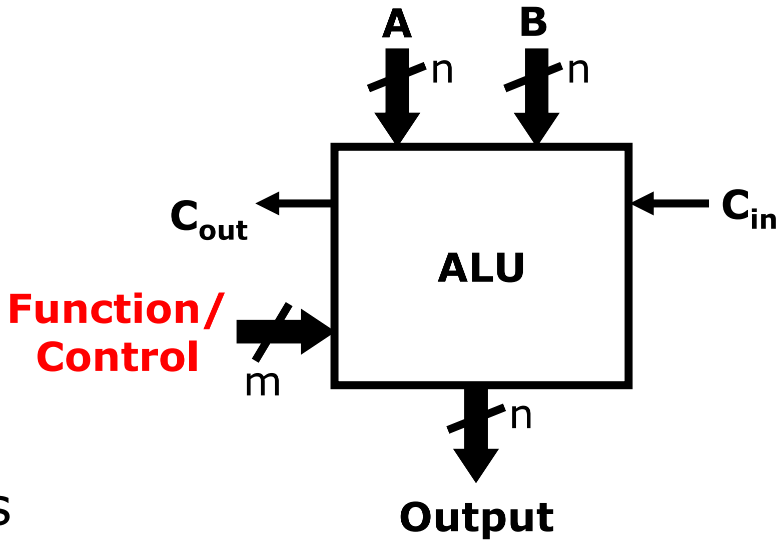
- To simplify representation we group bits together
- **Bus**
  - collection of wires
- **Bus width**
  - number of bits transferred together (in parallel)
  - 4 in this example


$B_3B_2B_1B_0$
Indicates component signals
A
4
B
4
C_{out}
C_{in}
Adder
4
Output
4 full adders in here (as per previous slide)

Arithmetic Logic Unit (ALU)

- Does more than adding…
- **Function/control** input indicates the operation that the ALU is to perform, e.g.
  - Addition
  - Increment (+1)
  - Subtraction
  - Bitwise AND
  - Bitwise OR
  - …
- Like adders, ALUs can be made up from 1-bit slices

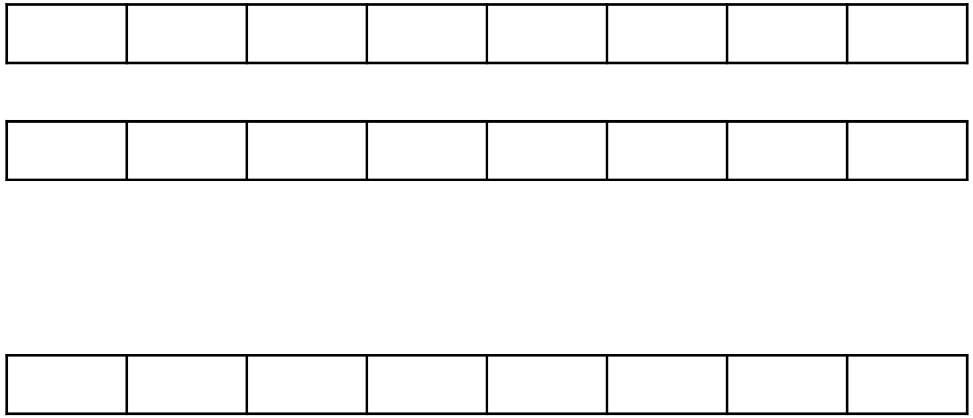
Versatile hardware which can perform various operations based on some control inputs



Bitwise Operations

- Operate on corresponding bits, e.g.

  bitwise A and B



Clicker question: What is the bitwise OR of 0x74 and 0x85?

17% A. 0x04

17% B. 0x74

17% C. 0x85

17% D. 0xF1

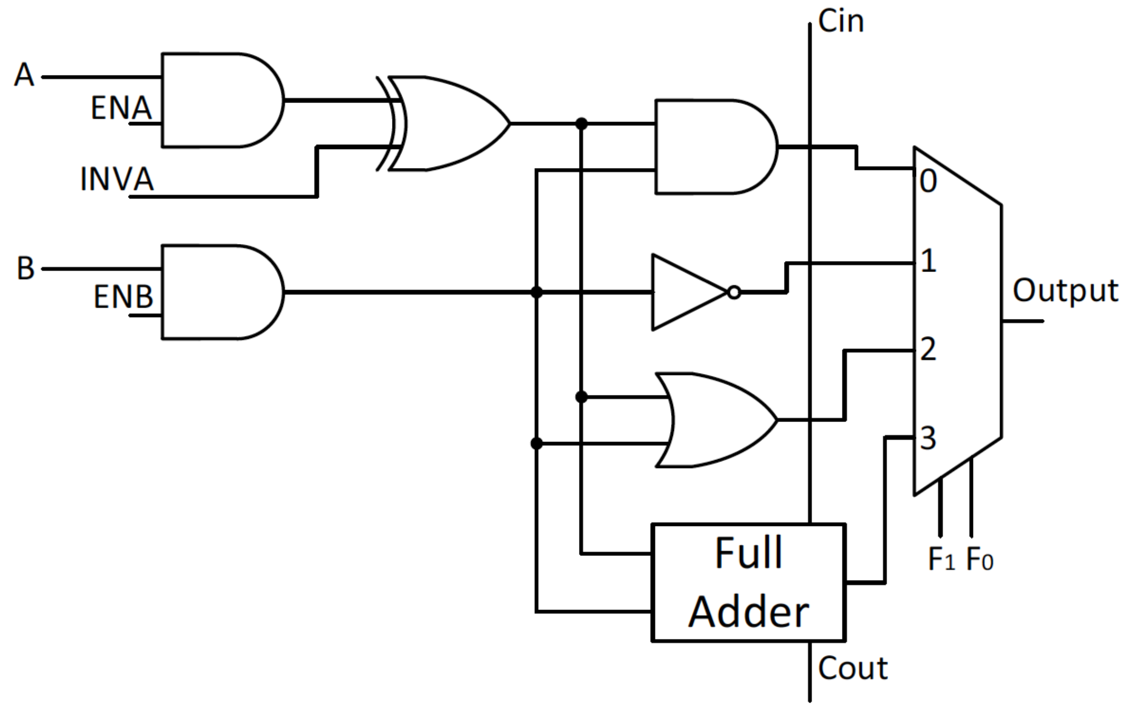
17% E. 0xF5

17% F. 0xF9

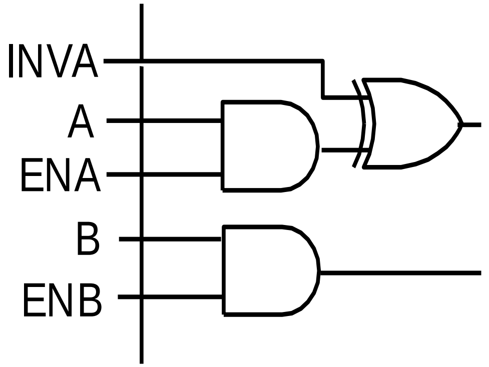
10

1-bit ALU (bit slice)

- **F_1, F_0** select ALU function
  - 00 ➔ A and B
  - 01 ➔ invert B
  - 10 ➔ A or B
  - 11 ➔ A plus B



Example 1-bit ALU - Components



Data inputs – A, B

Control inputs

- INVA – invert A- inverts the corresponding input if set to 1
- ENA, ENB – enable A, enable B – enable the data input if set to 1
- Also some other function select control inputs (not shown above)

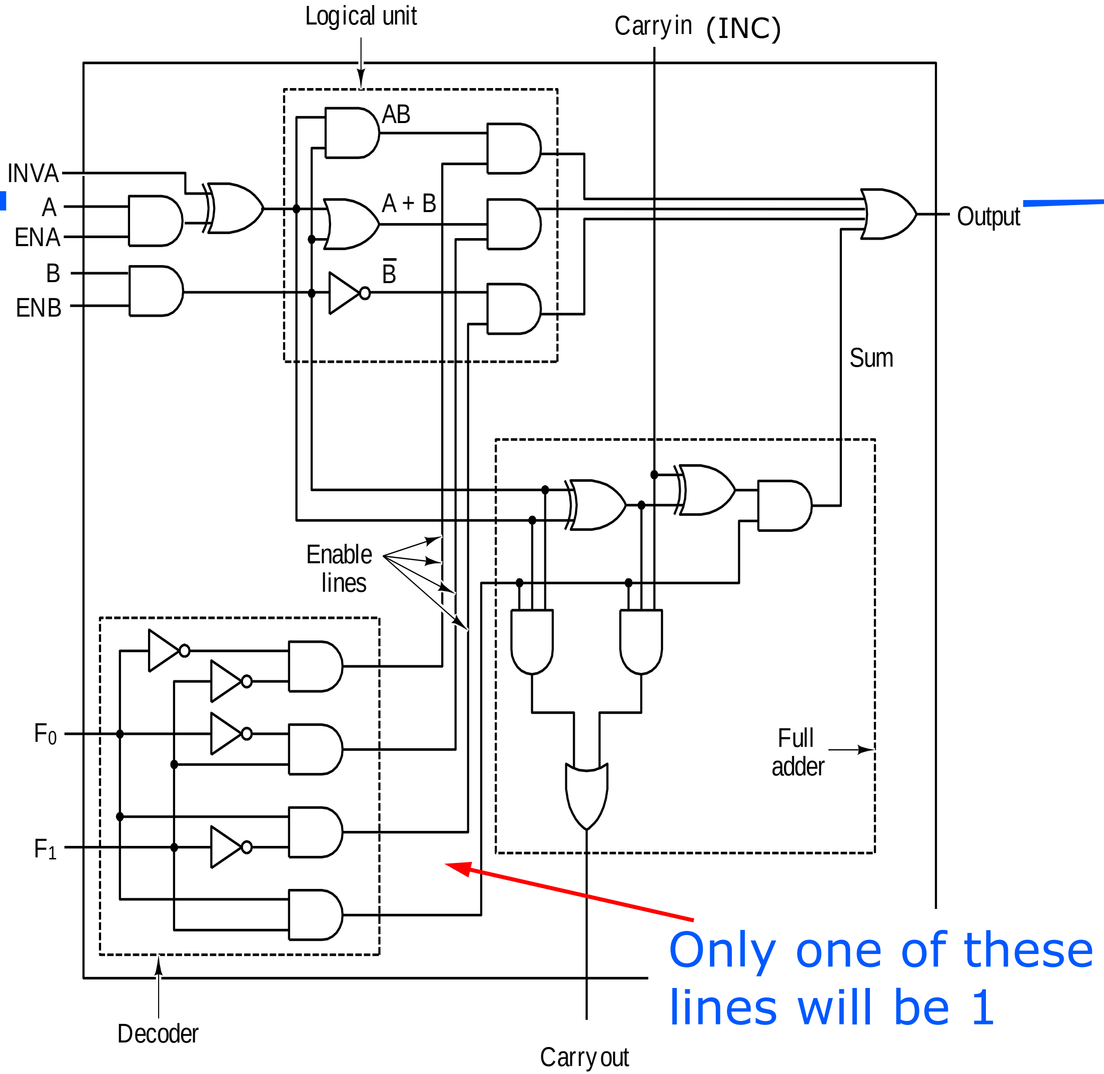
Two levels of control:

- Permutations of data inputs
- Function (operation)

Building the ALU

4 Functions

- A and B
- $\overline{B}$
- A or B
- A plus B

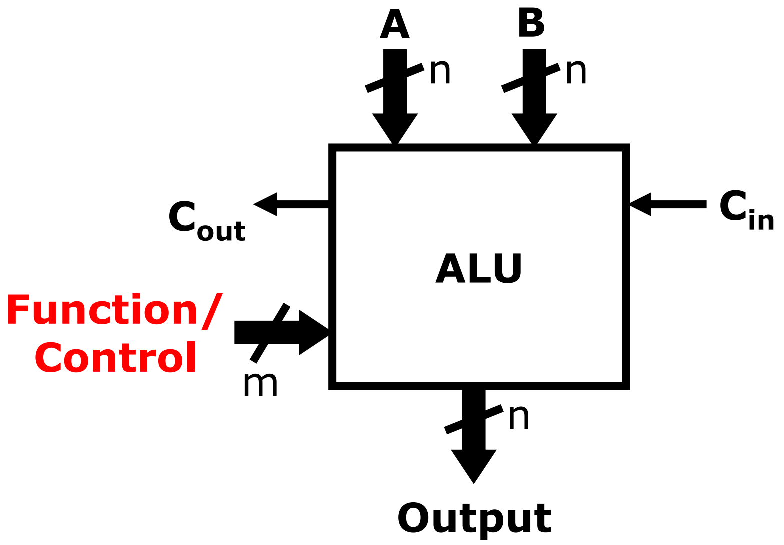


Only one of these lines will be 1

[Figure from Tanenbaum, "Structured Computer Organization"]

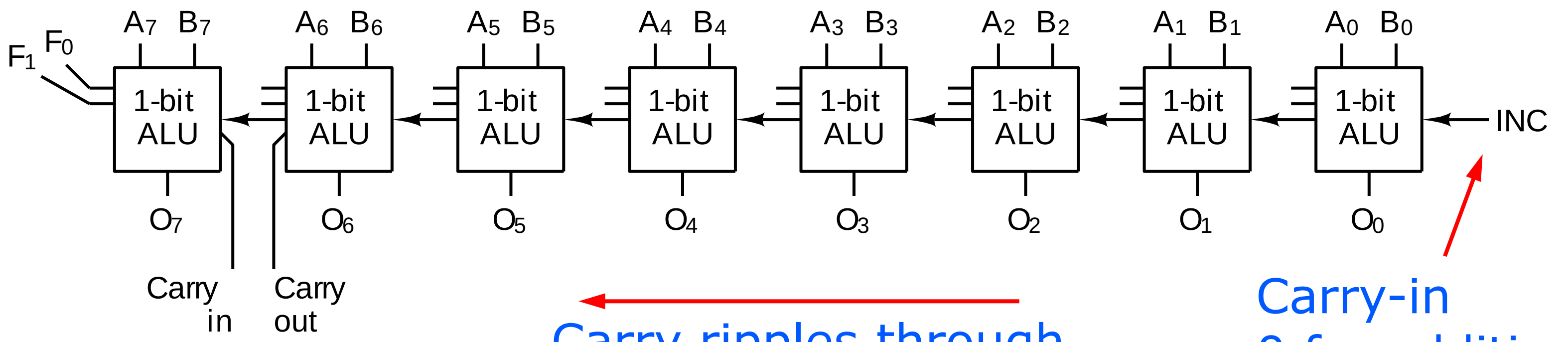
Arithmetic Logic Unit (ALU)

- Remember, machine word = *n* bits
  - (often 8, 16, 32, …)
- *n*-bit ALU can be built up of *n* identical circuits for the individual bit positions



ALU Bit Slice Architecture

- Can be cascaded to make an n-bit ALU
  - Example: 8 bits



Carry ripples through from right to left

Carry-in
0 for addition
1 for increment

- F_1, F_0, ENA, ENB, INVA inputs are shared between all bit slices

Example

- How can we calculate B–A using this ALU?
  - i.e. what are the values of F_1, F_0,ENA,ENB,INVA,INC
- Note: B–A $= B+(-A)$
  $= B+(\bar{A} + 1)$
  $= B + \bar{A} + 1$

Two's complement negation

How can we calculate B+1 (B plus 1) using this ALU?

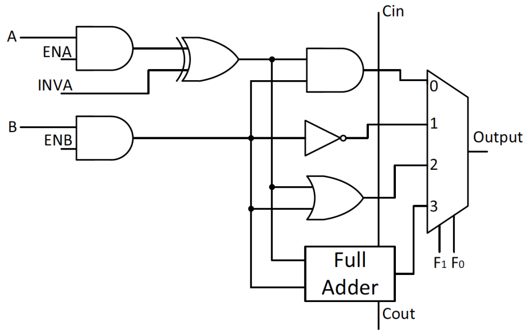
20% A. $F_0=0$ $F_1=1$ ENA=0 ENB=1 INVA=1 INC=0

20% B. $F_0=0$ $F_1=1$ ENA=0 ENB=1 INVA=0 INC=1

20% C. $F_0=1$ $F_1=1$ ENA=0 ENB=1 INVA=1 INC=0

20% D. $F_0=1$ $F_1=1$ ENA=0 ENB=1 INVA=1 INC=1

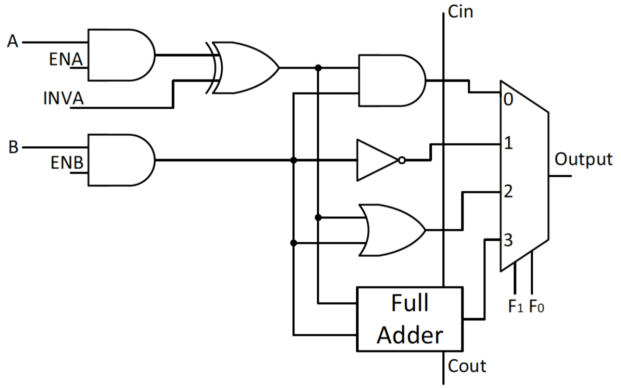
20% E. $F_0=1$ $F_1=1$ ENA=0 ENB=1 INVA=0 INC=1



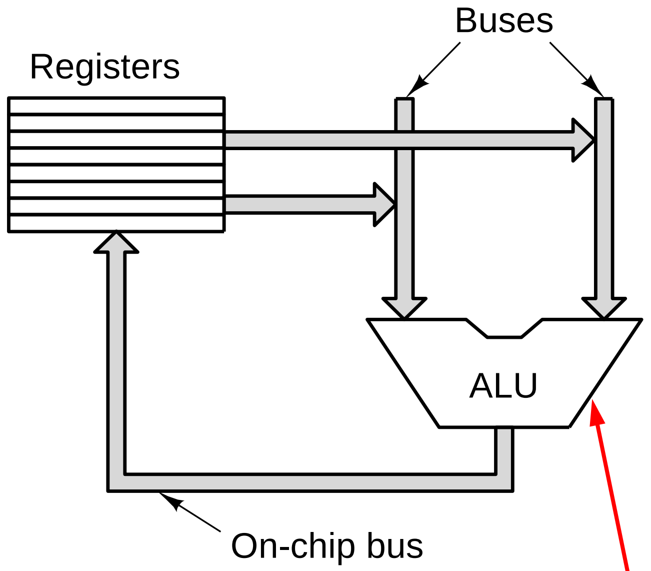
10

Some Useful Combinations of ALU Inputs

F_1	F_0	ENA	ENB	INVA	INC	Function
1	0	1	0	0	0	A
1	0	0	1	0	0	B
1	0	1	0	1	0	$\overline{A}$
0	1	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
						B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	– A
0	0	1	1	0	0	A AND B
1	0	1	1	0	0	A OR B
1	0	0	0	0	0	0
1	1	0	0	0	1	1
1	0	0	0	1	0	– 1



How the ALU is used … Data Path



ALU inside a Central Processing Unit (CPU)

Conventional ALU Symbol

- Operands come from register file
- Result written to register file
- Implements routine instructions such as arithmetic, logical, shift
- Width of registers/buses is the CPU word size

Data Path (Explanation)

- ALU performs operations on data from the register file
  - CPU control unit determines which registers the operands come from and which operation is performed
- Result returned to a register within the register file
  - Could overwrite one of the operands
- Register ⇔ memory transfers also possible
  - Read: load a register with data from a memory address
  - Write: store a register’s contents to a memory address

ALU: In Reality

- Microprocessor may have a number of processing units, e.g.
  - One or more integer processing units
  - Floating point unit(s)
- Units may be capable of operating in parallel