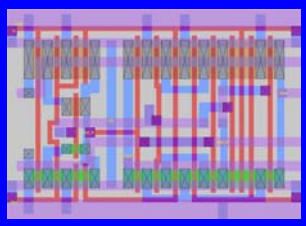
Computer Arithmetic

Logical, Integer Addition & Subtraction

Chapter 3.1-3.3

EEC170 FQ 2005



Layout of a full adder cell

MIPS Integer Representation

- 32-bit signed integers, e.g., for numeric operations
  - 2's complement: one representation for zero, balanced, allows add/subtract to be treated uniformly
- 32-bit unsigned integers, e.g., for address operations
  - Address considered 32-bit unsigned integer
- Provides distinct instructions for signed/unsigned:
  - ADD, ADDI: add signed register, add signed immediate
    - causes exception on overflow
  - ADDU, ADDIU: add unsigned register, add unsigned immediate
    - no exception on overflow

OP	Rs	Rt	Rd	0	ADD/U

ADDI/U	Rs	Rt	immediate data

Comparison

- Distinct instructions for comparison of signed/unsigned integers
  - Which is larger: 1111...1111 or 0000...0000 ? Depends of type, signed or unsigned
- Two versions of slt for signed/unsigned:
  - slt, sltu: set less than signed, unsigned

OP	Rs	Rt	Rd	0	SLT/U

- Two versions of immediate comparison also provided:
  - slti, sltiu: set less than immediate signed, unsigned

SLTI/U	Rs	Rt	immediate data

Sign Extension

- Sign of immediate data extended to form 32-bit representation:

1111111111111111**1**0010101010101100

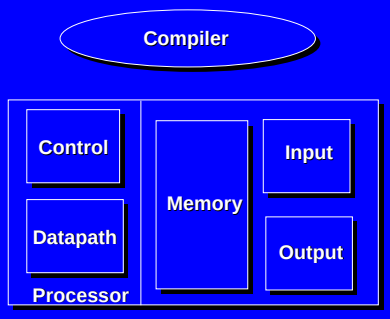
0000000000000000**0**1010101010101100

- Thus, ALU always uses 32-bit operands
- Extension occurs for signed and unsigned arithmetic

Overflow

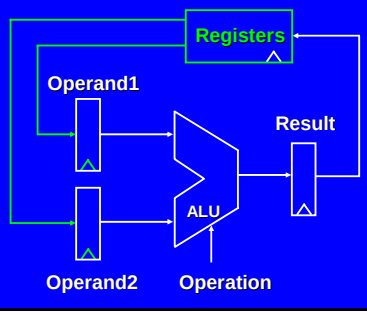
- MIPS has no flag (status) register
  - complicates pipeline (see Chapter 6)
- Overflow (underflow):
  - Occurs if operands are same sign, result is different sign.
  - Can be checked in software if necessary
  - MIPS generates interrupt on overflow for signed arithmetic to notify program
  - C compiler only generates unsigned arithmetic instructions (avoids interrupts)
    - Representation of result is the same if signed/unsigned

Computer System, Top Level View



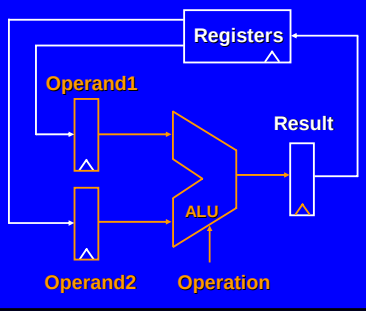
Simple Processor Datapath

- Includes registers and ALU
  - Cycle 1: Register Fetch



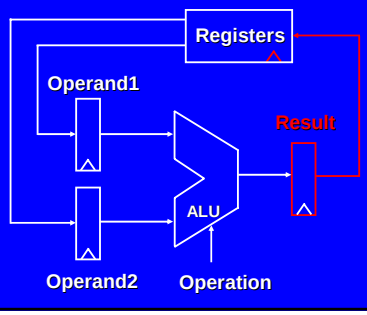
Simple Processor Datapath

- Includes registers and ALU
  - Cycle 2: ALU Operation



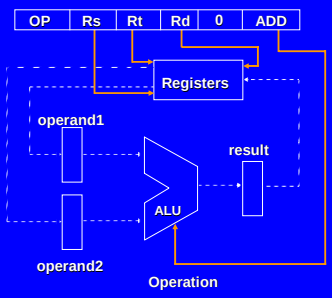
Simple Processor Datapath

- Includes registers and ALU
  - Cycle 3: Register Write Back

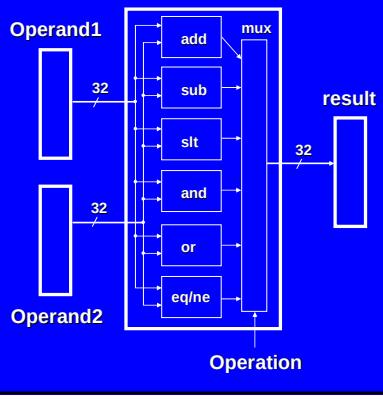


Processor Control

- Control directs actions in the data path. Instruction is top level of control

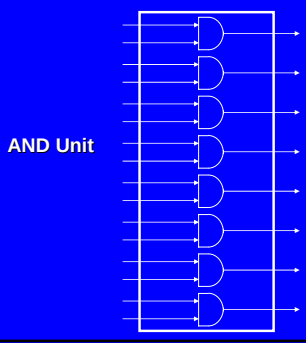


Inside the ALU



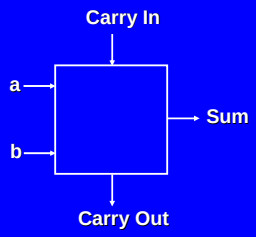
Boolean Operations

- Each pair of inputs resolved using a corresponding gate



One Bit Full Adder

- Multiple-bit adder can be built from series of one-bit full adders



Full Adder Truth Table

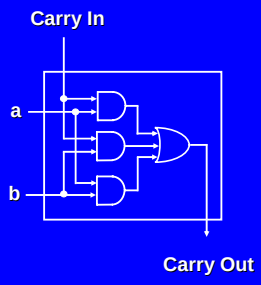
- Adder truth table maps inputs to the outputs

Input			Output	
a	b	Carry_In	Carry_Out	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

Carry_Out = b•Carry_In + a•Carry_In + a•b

Sum = $\bar{a}\cdot\bar{b}\cdot$Carry_in + $\bar{a}\cdot b\cdot\overline{\text{Carry\_in}}$ + $a\cdot\bar{b}\cdot\overline{\text{Carry\_in}}$ + a•b•Carry_in

Carry_Out Implementation



Two's Complement Subtraction

- For two's complement arithmetic the following is true:

$X + \bar{X} = -1$ Proof: result of $X + \bar{X}$ is all ones
101001 + 010110 = 111111

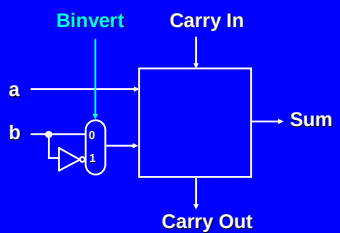
$X + (\bar{X} + 1) = 0$

$-X = \bar{X} + 1$

- Our strategy for computing A – B: convert to A + (-B) and use the above rule for -B

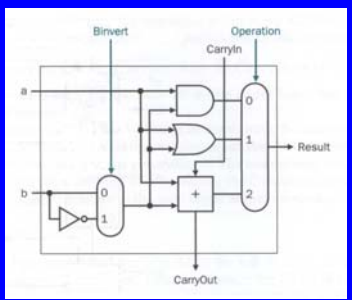
One Bit Add/Subtract Cell

- Modify the adder cell so that the B input is selectable



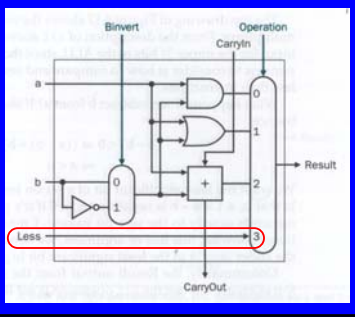
Single Bit ALU

- Combine gates with add/subtract unit to form 1-bit ALU
  - Operation selects which result is sent out
  - 1-bit ALUs will be connected in a chain to form 32-bit ALU



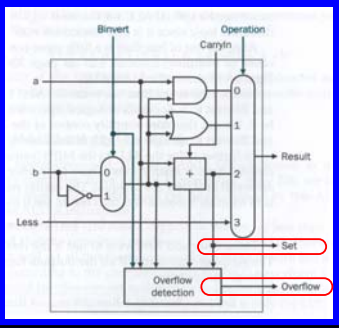
Support for Set Less Than

- And an input Less to 1-bit ALU for SLT
  - Input connections will be seen soon



Most Significant 1-Bit ALU

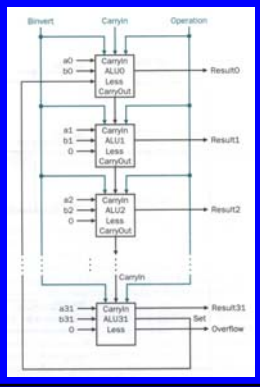
- Specialize ALU Bit 31 to produce Overflow and Set (for set less than)
  - What's inside the overflow detection box?



32-Bit ALU

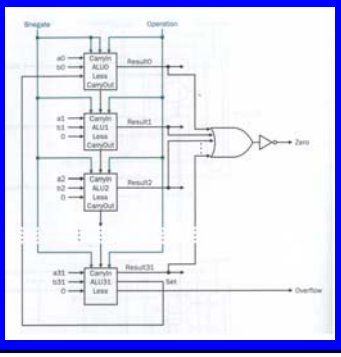
- 32 one-bit ALUs are cascaded to form 32-bit ALU
- SLT condition is computed using A-B, Set is sign-bit of result
  - All other Less inputs set to 0
  - Set is not accurate when there's overflow, why?

  How is this corrected?



Support for BEQ/BNE

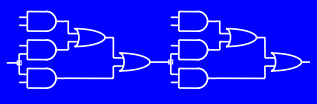
- Determine EQ/NE using subtract operation
  - All zero result is EQ, else NE



Ripple Carry Adder Performance

- This Ripple Carry Adder is very slow. Each stage causes 3 two-input gate delays:

$$c_{out} = ab + ac_{in} + bc_{in}$$



- For a 32-bit adder this would be 96 two-input gate delays, much too slow
- Can speedup addition by using more hardware
  - Carry-look ahead adder

Vector Arithmetic

- Allows arithmetic to be done in parallel on subcomponents of a register, e.g., arithmetic on bytes:

31 24	23 16	15 8	7 0
X_3	X_2	X_1	X_0
+ Y_3	Y_2	Y_1	Y_0
Z_3	Z_2	Z_1	Z_0

Vector Arithmetic (cont.)

- Vector arithmetic has been used for decades in supercomputers for scientific applications
- Also used for architecture extensions to facilitate multimedia processing for RISC processors and for x86 (MMX,SIMD), e.g.:
  - The intensity of each display dot (pixel) is often represented by 1 byte
  - Arithmetic on a vector of bytes allows parallel pixel arithmetic, e.g., adding two scenes together bit by bit
- Simple extensions to our MIPS architecture and the 32-bit adder implementation will allow both 32-bit arithmetic and byte-parallel arithmetic:

  Addv Rd,Rs,Rt

  Subv Rd,Rs,Rt

Multimedia Arithmetic

- Multimedia applications require unconventional types of arithmetic. Multimedia architecture extensions include new arithmetic operations
  - E.g., saturating addition/subtraction:
    - when altering the intensity of a pixel (e.g., adding two scenes, increasing overall intensity by 2x), we do not want overflow or underflow to cause wrap around (bright pixel becomes dark).
    - Overflow or underflow detection/special processing is not feasible
    - Rather, arithmetic result should saturate at maximum or minimum value that can be represented. Thus for 8-bit representation:

      243 + 124 = 255
- New multimedia operations require additional ALU changes