CS352H: Computer Systems Architecture

Lecture 4: Instruction Set Architectures III + MIPS ALU

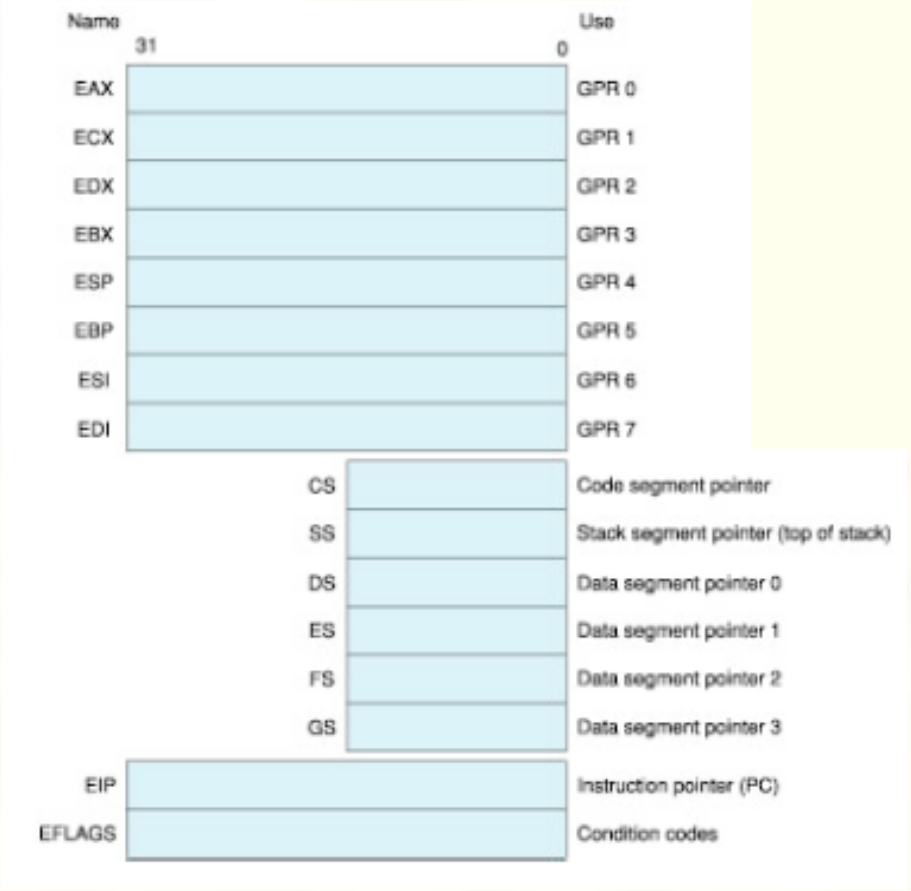
September 10, 2008

ISA Example: IA-32

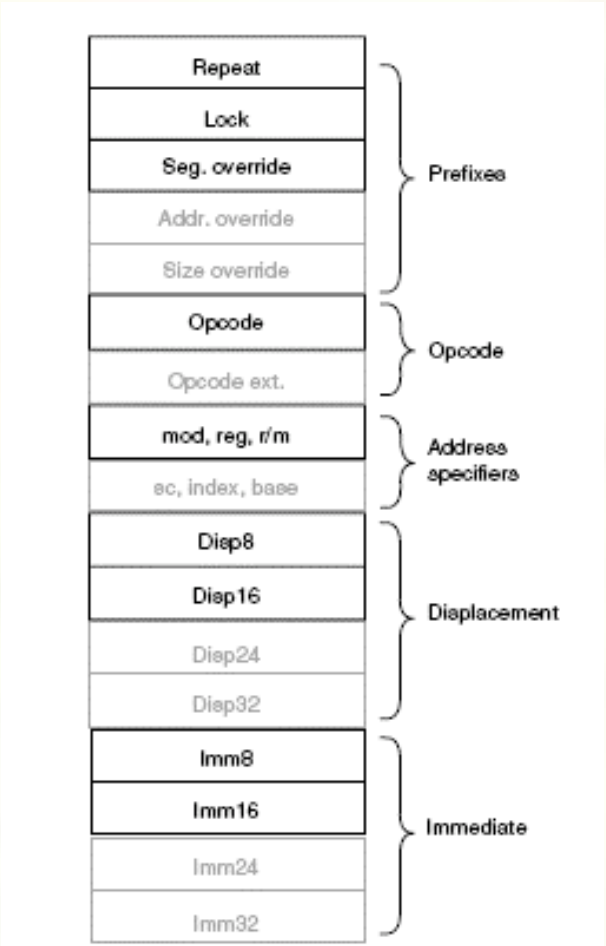
- A 20-year old ISA! (30 years if you count earlier x86)
- Blend of RR & RM ISA
  - Two-operand instructions
  - Seven addressing modes
  - Mode to indicate whether 8086, 286 or IA-32 (>=386)
- More than 630 instructions!
  - Evolved to include multimedia, graphics
  - Support operations on 8, 16 and 32-bit operands
- Variable-length instructions
  - 1 – 15 bytes long (3 bytes on average)

IA-32 Registers

- Originally 16-bit registers
- As of 386 eight GP 32-bit regs
- Segment registers encode address prefix
  - Certain accesses particular segment registers by default
- Condition code register



IA Instruction Formats



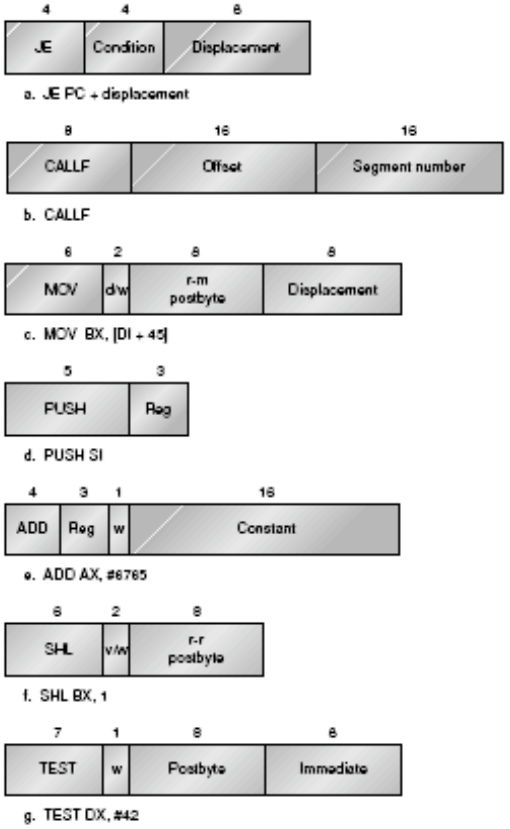
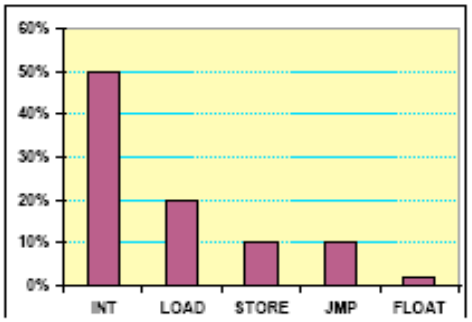


Figure D.8 Typical 8086 instruction formats. The encoding of the postbyte is shown in Figure D.9. Many instructions contain the 1-bit field w, which says whether the operation is a byte or a word. Fields of the form v/w or d/w are a d-field or v-field followed by the w-field. The d-field in MOV is used in instructions that may move to or from memory and shows the direction of the move. The field v in the SHL instruction indicates a variable-length shift; variable-length shifts use a register to hold the shift count. The ADD instruction shows a typical optimized short encoding usable only when the first operand is AX. Overall instructions may vary from one to six bytes in length.

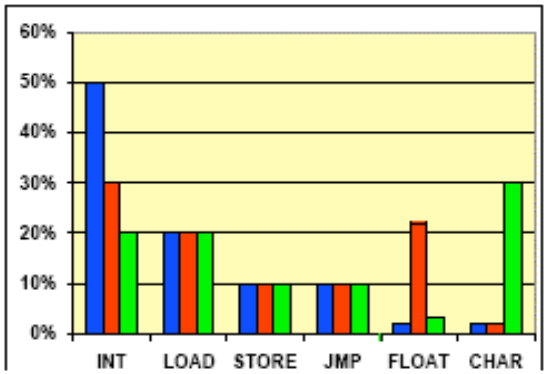
Principles of ISA Design

- KISS — Keep It Simple, Stupid (Cray)
  - Complexity increases
    - Logic area
    - Pipe stage duration
    - Development time
  - Evolution leads to kludges
- Orthogonality
  - Simple rules, few exceptions
  - All ops on all registers in all addressing modes
- Frequency
  - Make the common case fast
    - Instructions are not born equal!



Principles of ISA Design (cont)

- Generality
  - Not all problems need the same capabilities
    - Toaster oven vs. supercomputer



- Performance should be easy to predict
  - Peter Denning: Don't build it if you can't model it
- Permit efficient implementation
  - Today
  - 10 years from now

CISC vs. RISC

Complex Instruction Set Computer (CISC)

- Typically includes:
  - Variable-length instructions
  - RM instructions
  - Many, complex addressing modes
  - Complex instructions
- Examples:
  - VAX, IBM 360/370, x86
- Advantages
  - Code density
  - Legacy SW

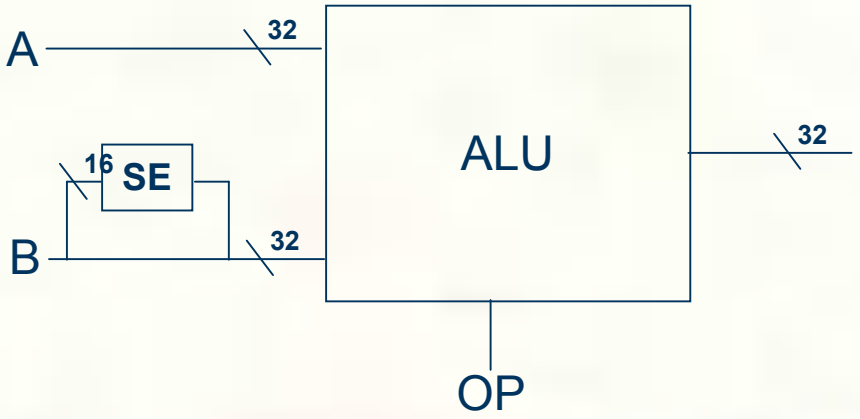
Reduced Instruction Set Computer (RISC)

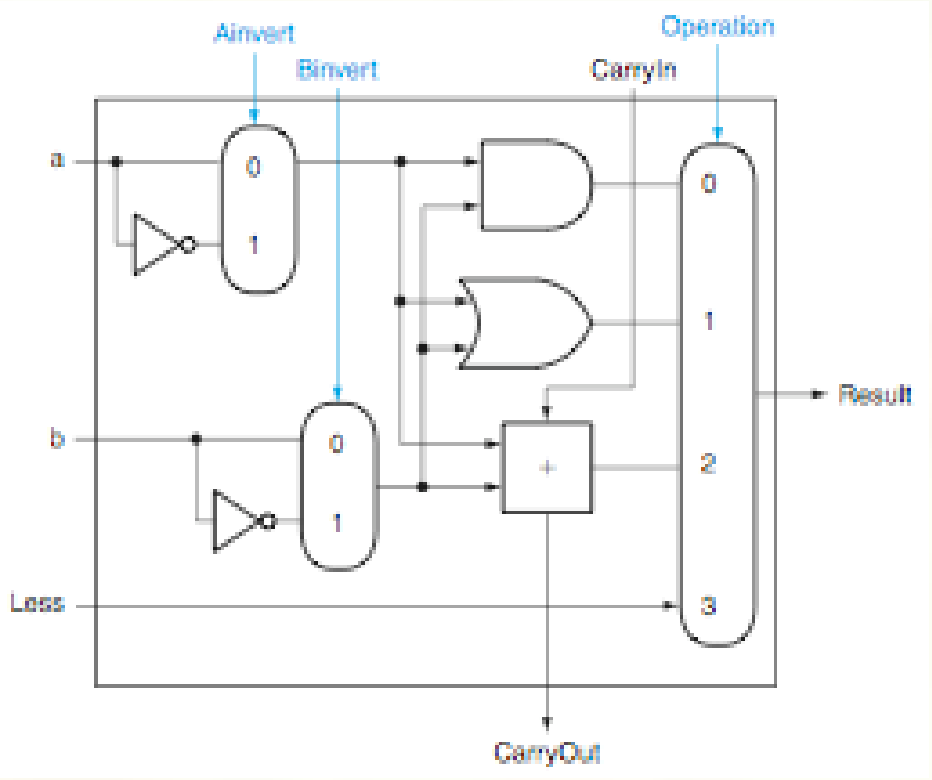
- Typically includes:
  - GP registers
  - Fixed 3-address format
  - Strict load/store conformance
  - Few, simple addressing modes
  - Simple instructions
- Examples:
  - DEC Alpha, MIPS
- Advantages
  - Good compiler target
  - Easy to implement/pipeline

In practice, other than a handful of ISAs, most contain elements of both.

MIPS ALU

- Needs to support
  - add, addi, sub
  - and, andi, or, ori, xor, xori
  - sll, srl, sra, sllv, srlv, srav
  - slt, slti
  - All the unsigned versions

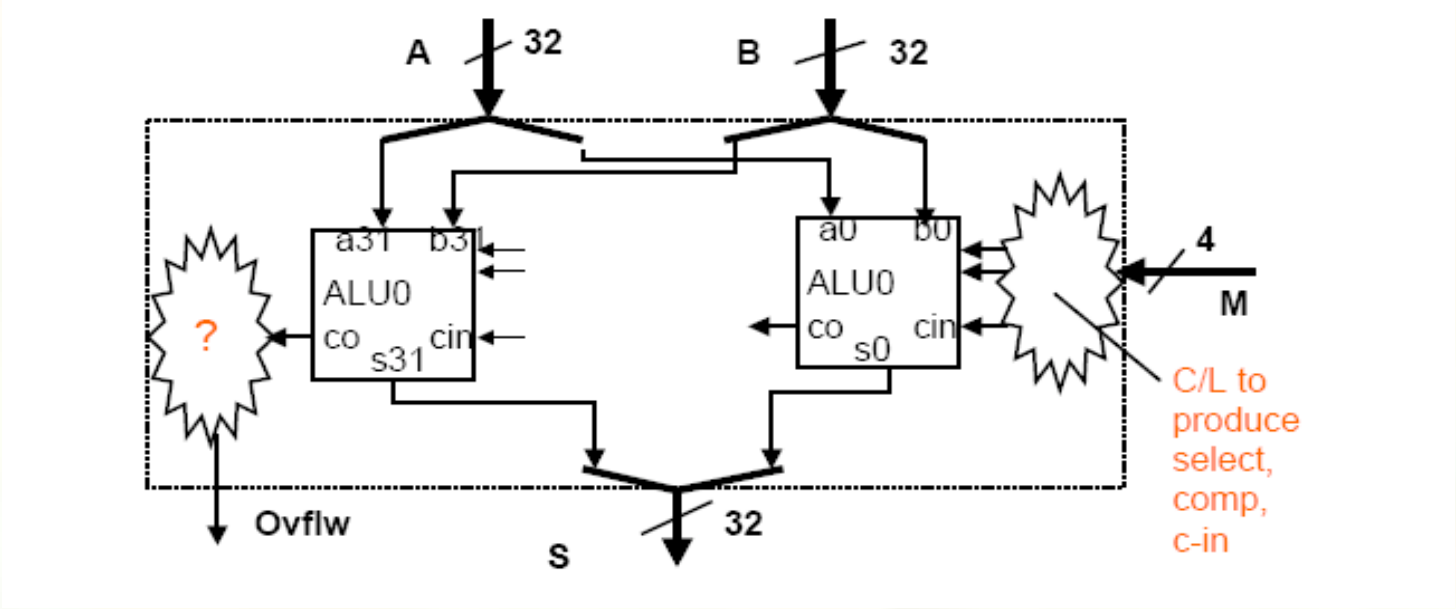




Bit Slice Approach

- For subtraction: set invert & CarryIn to 1 (2's complement arithmetic)
- How about SLT?

LSB and MSB Need to Do Extra



Overflow Detection

Overflow occurs when the result is too large to represent in the number of bits allocated

adding two positives yields a negative

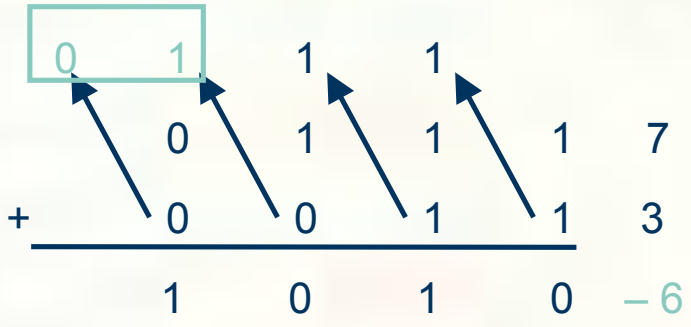
or, adding two negatives gives a positive

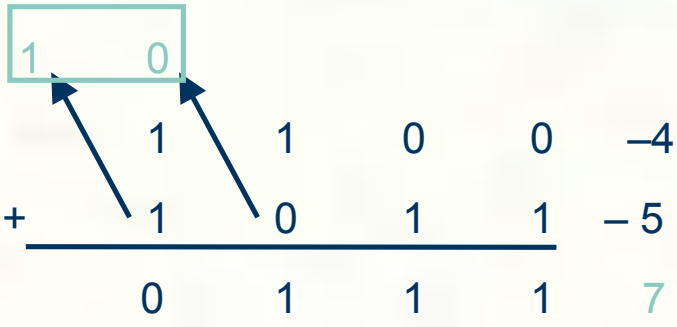
or, subtract a negative from a positive gives a negative

or, subtract a positive from a negative gives a positive

On your own: Prove you can detect overflow by:

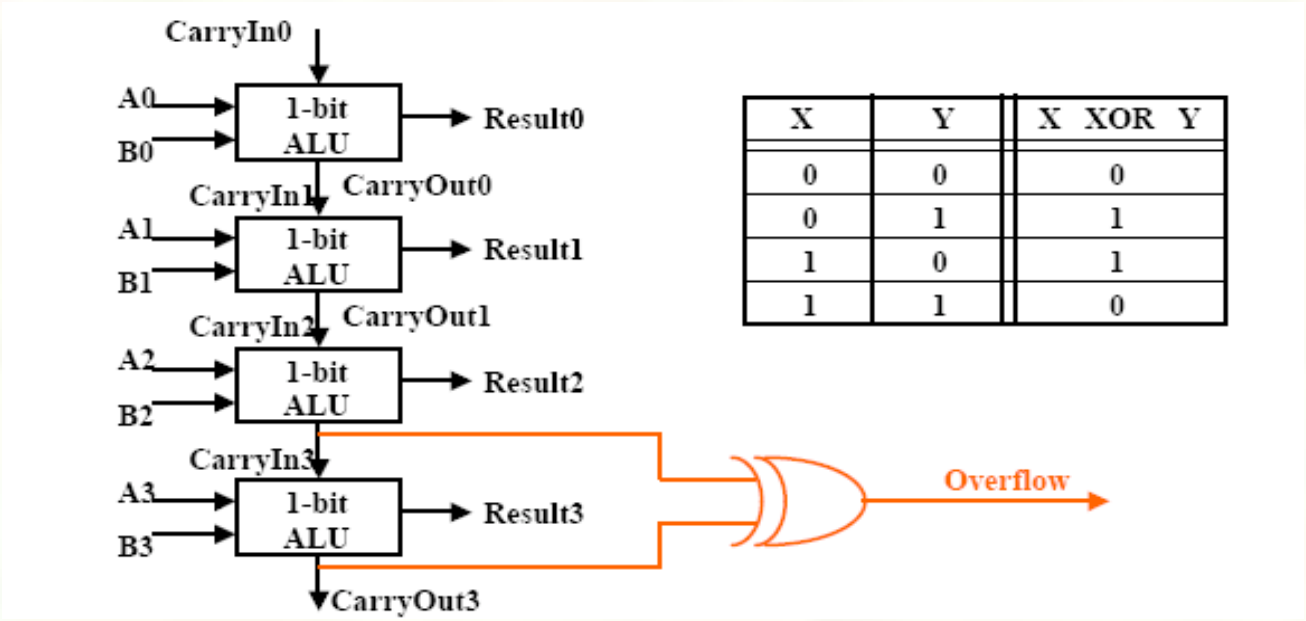
Carry into MSB xor Carry out of MSB



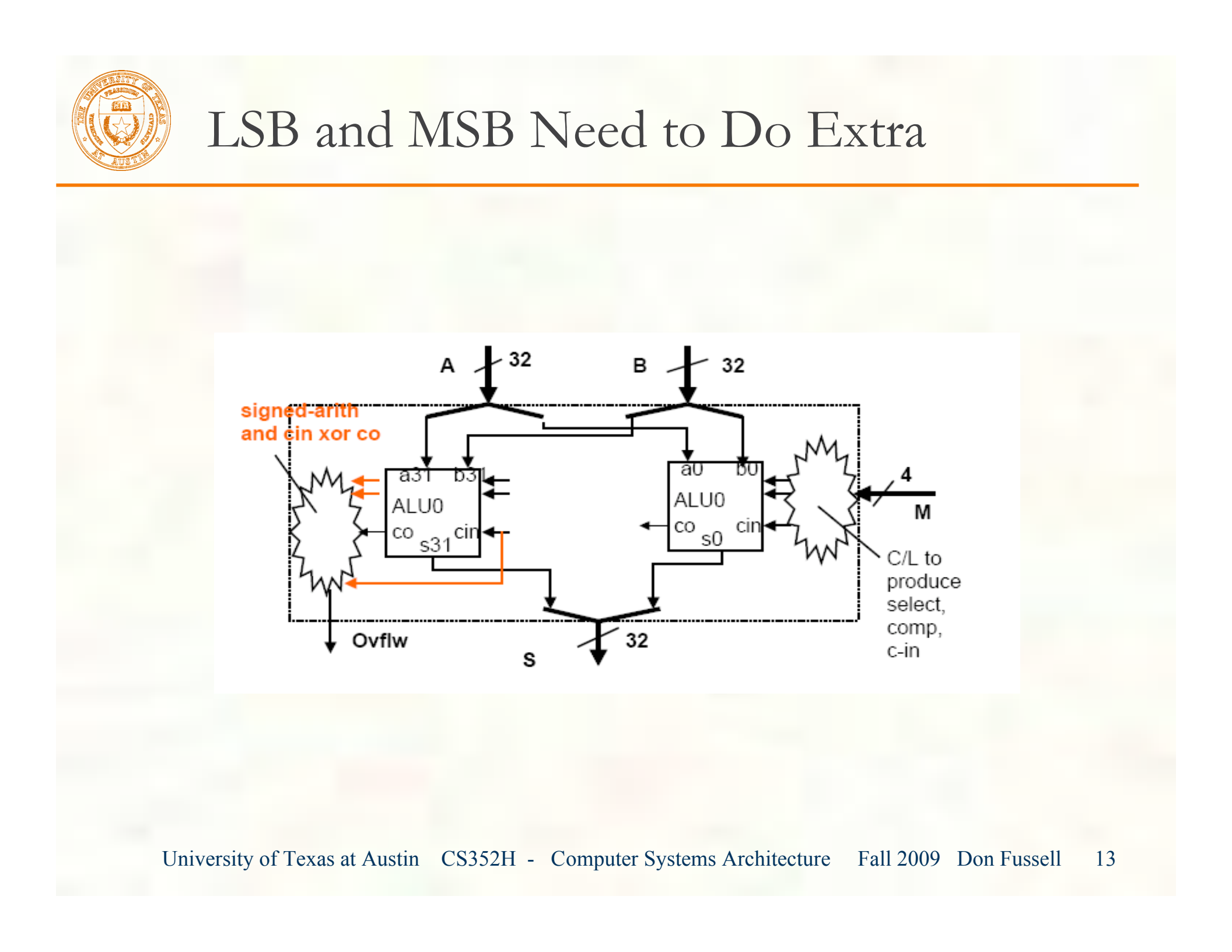


Overflow Detection Logic

For an N-bit ALU: Overflow = CarryIn[N-1] XOR CarryOut[N-1]



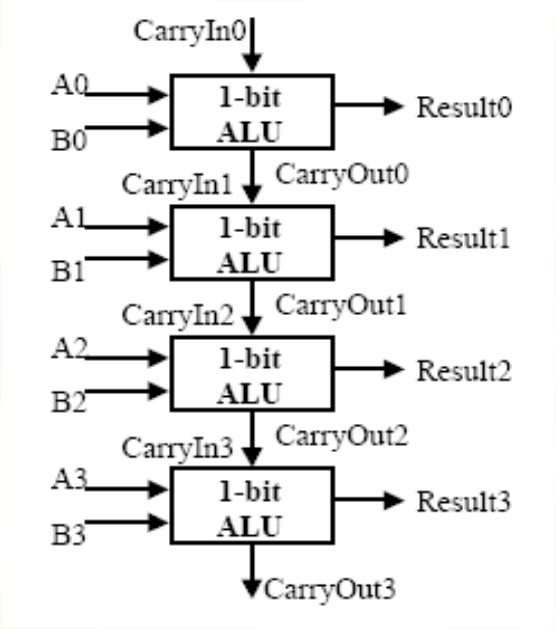
X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0



LSB and MSB Need to Do Extra

What About Performance?

- Critical path of N-bit ripple-carry adder is N * T_{add}



- Throw hardware at it! (Next lecture)

Next Lecture

- Finish MIPS ALU
  - See Appendix C (on CD)
    - In particular, Sections C-5 & C-6 for fast adders
  - See Chapter 3.3 & 3.4 for Multiplication & Division
  - See Chapter 3.5 for Floating point ops