RISC Design

Virendra Singh
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-309: Microprocessors

Lecture 29 (01 Oct 2015)

CADSL

RISC Architecture

- Simple instructions
- Fixed Instruction Encoding
- Limited Addressing Mode
- Instruction count increases
- Simple controller
- Load/Store architecture

Arithmetic Instructions

- Design Principle: simplicity favors regularity.
- Of course this complicates some things...

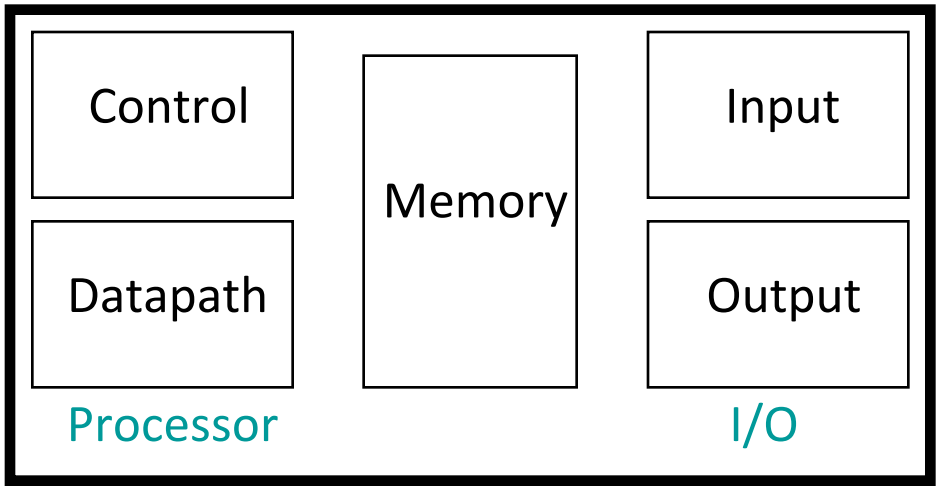
C code: **a = b + c + d;**

DLX code: **add a, b, c**
add a, a, d

- Operands must be registers (why?)
- 32 registers provided
- Each register contains 32 bits

Registers vs. Memory

- Arithmetic instructions operands must be registers
  - **32 registers provided**
- Compiler associates variables with registers.
- What about programs with lots of variables? Must use memory.



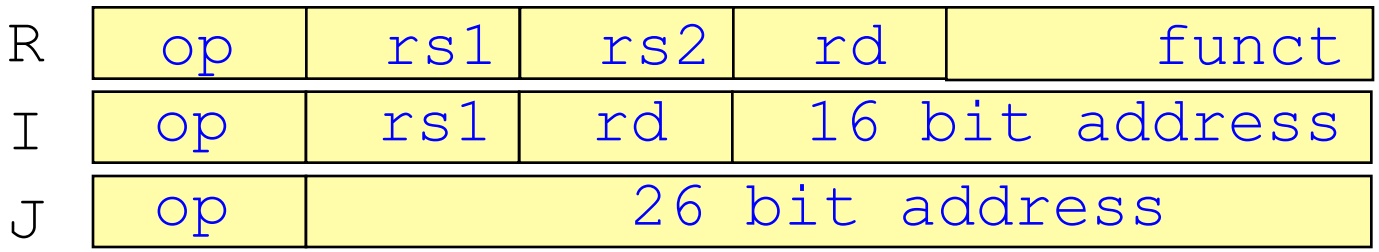
Memory Organization

- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array.
- "Byte addressing" means that the index points to a byte of memory.

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data
.	. . .
.	. . .

Overview of DLX

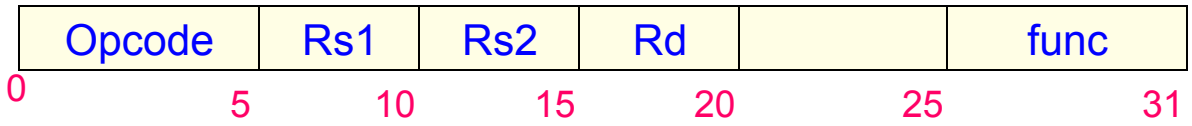
- Simple instructions, all 32 bits wide
- Very structured, no unnecessary baggage
- Only three instruction formats



- Rely on compiler to achieve performance

Instruction Set

Register-Register Instructions



Arithmetic and Logical Instruction

- ADD Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] + Reg[Rs2]
- SUB Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] - Reg[Rs2]
- AND Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] and Reg[Rs2]
- OR Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] or Reg[Rs2]
- XOR Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] xor Reg[Rs2]
- SUB Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1]-Reg[Rs2]

DLX Instruction Set

ADD Rd, Rs1, Rs2	Rd ← Rs1 + Rs2 (overflow – exception)	R	000_000 000_100
SUB Rd, Rs1, Rs2	Rd ← Rs1 - Rs2 (overflow – exception)	R	000_000 000_110
AND Rd, Rs1, Rs2	Rd ← Rs1 and Rs2	R	000_000/ 001_000
OR Rd, Rs1, Rs2	Rd ← Rs1 or Rs2	R	000_000/ 001_001
XOR Rd, Rs1, Rs2	Rd ← Rs1 xor Rs2	R	000_000/ 001_010
SLL Rd, Rs1, Rs2	Rd ← Rs1 << Rs2 (logical) (5 lsb of Rs2 are significant)	R	000_000 001_100
SRL Rd, Rs1, Rs2	Rd ← Rs1 >> Rs2 (logical) (5 lsb of Rs2 are significant)	R	000_000 001_110
SRA Rd, Rs1, Rs2	Rd ← Rs1 >> Rs2 (arithmetic) (5 lsb of Rs2 are significant)	R	000_000 001_111

DLX Instruction Set

ADDI Rd, Rs1, Imm	Rd ← Rs1 + Imm (sign extended) (overflow – exception)	I	010_100
SUBI Rd, Rs1, Imm	Rd ← Rs1 – Imm (sign extended) (overflow – exception)	I	010_110
ANDI Rd, Rs1, Imm	Rd ← Rs1 and Imm (zero extended)	I	011_000
ORI Rd, Rs1, Imm	Rd ← Rs1 or Imm(zero extended)	I	011_001
XORI Rd, Rs1, Imm	Rd ← Rs1 xor Imm(zero extended)	I	011_010
SLLI Rd, Rs1, Imm	Rd ← Rs1 << Imm (logical) (5 lsb of Imm are significant)	I	011_100
SRLI Rd, Rs1, Imm	Rd ← Rs1 >> Imm (logical) (5 lsb of Imm are significant)	I	011_110
SRAI Rd, Rs1, Imm	Rd ← Rs1 >> Imm (arithmetic) (5 lsb of Imm are significant)	I	011_111

DLX Instruction Set

LHI Rd, Imm	Rd(0:15) ← Imm Rd(16:32) ← hex0000 (Imm: 16 bit immediate)	I	011_011
NOP	Do nothing	R	000_000 000_000

DLX Instruction Set

SEQ Rd, Rs1, Rs2	Rs1 = Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_000
SNE Rd, Rs1, Rs2	Rs1 /= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_010
SLT Rd, Rs1, Rs2	Rs1 < Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_100
SLE Rd, Rs1, Rs2	Rs1 <= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_110
SGT Rd, Rs1, Rs2	Rs1 > Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 011_000
SGE Rd, Rs1, Rs2	Rs1 >= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 011_010

DLX Instruction Set

SEQI Rd, Rs1, Imm	Rs1 = Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000 (Imm: Sign extended 16 bit immediate)	I	100_000
SNEI Rd, Rs1, Imm	Rs1 /= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_010
SLTI Rd, Rs1, Imm	Rs1 < Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_100
SLEI Rd, Rs1, Imm	Rs1 <= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_110
SGTI Rd, Rs1, Imm	Rs1 > Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	101_000
SGEI Rd, Rs1, Imm	Rs1 >= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	101_010

DLX Instruction Set

BEQZ Rs, Label	Rs = 0: PC ← PC+4+Label Rs /= 0: PC ← PC+4 (Label: Sign extended16 bit immediate)	I	010_000
BNEZ Rs, Label	Rs /= 0: PC ← PC+4+Label Rs = 0: PC ← PC+4	I	010_001
J Label	PC ← PC + 4 + sign_extd(imm26)	J	001_100
JAL Label	R31 ← PC + 4 PC ← PC+ 4 + sign_extd(imm26)	J	001_100
JAL Label	R31 ← PC + 4 PC ← PC+ 4 + sign_extd(imm26)	J	001_101
JR Rs	PC ← Rs	I	001_110
JALR Rs	R31 ← PC + 4 PC ← Rs	I	001_111

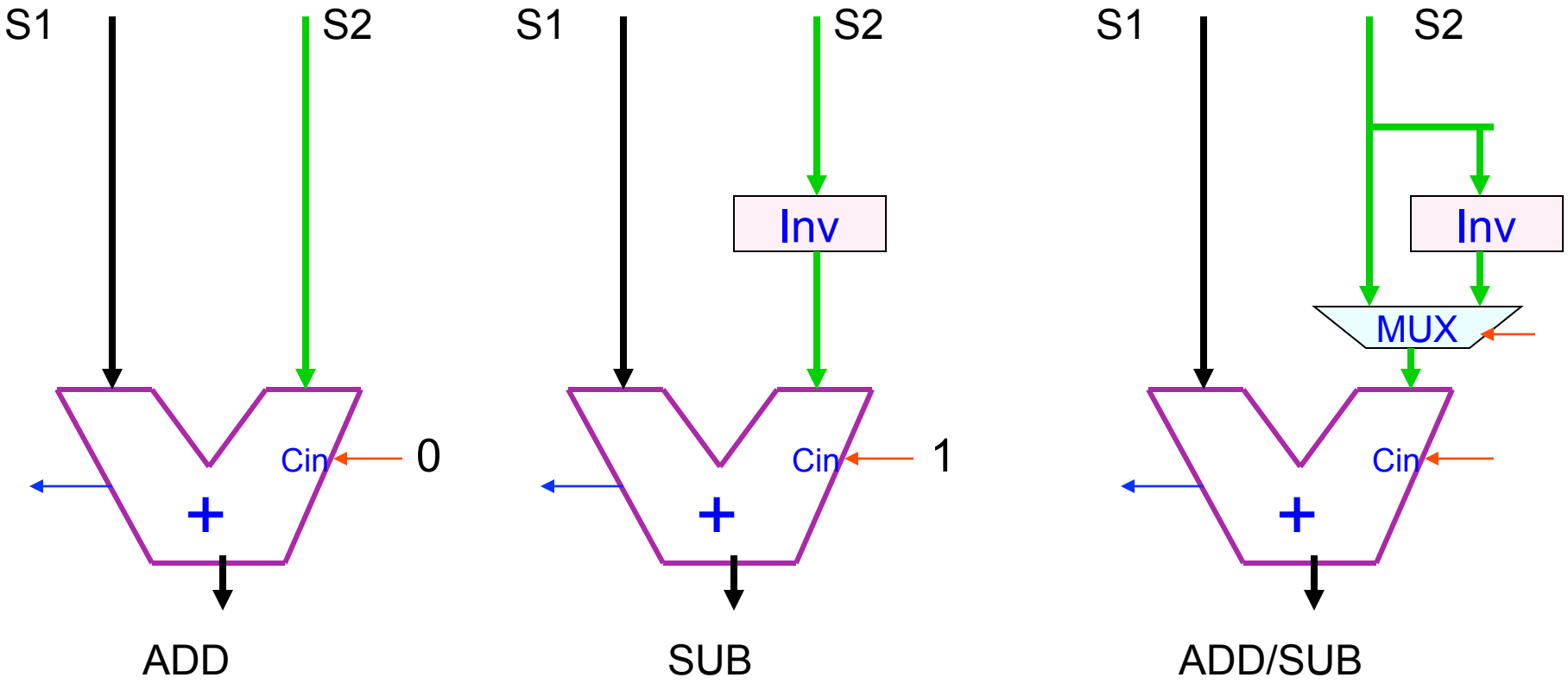
DLX Instruction Set

LW Rd, Rs2 (Rs1)	Rd ← M(Rs1 + Rs2) (word aligned address)	R	000_000 100_000
SW Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd	R	000_000 101_000
LH Rd, Rs2 (Rs1)	Rd (16:31)← M(Rs1 + Rs2) (Rd sign extended to 32 bit)	R	000_000 100_001
SH Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd(16:31)	R	000_000 101_001
LB Rd, Rs2 (Rs1)	Rd (24:31)← M(Rs1 + Rs2) (Rd sign extended to 32 bit)	R	000_000 101_010
SB Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd(24:31)	R	000_000 101_010

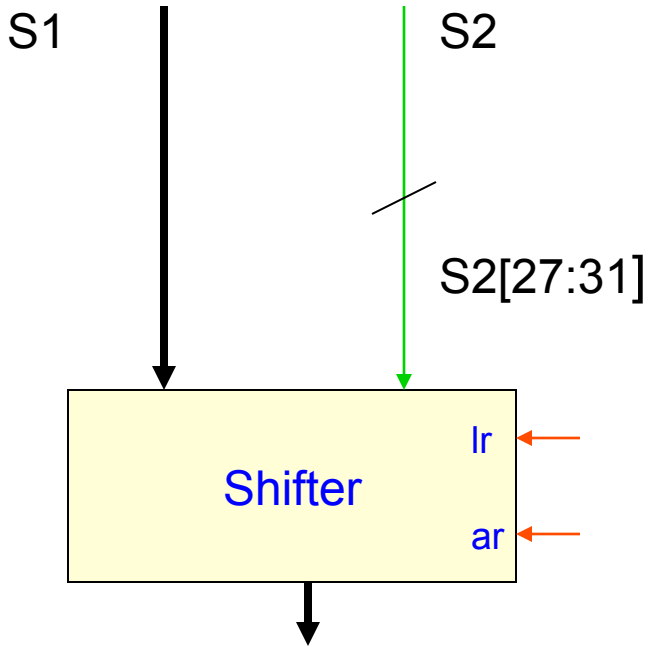
DLX Instruction Set

LWI Rd, Imm (Rs)	Rd ← M(Rs + Imm) (Imm: sign extended 16 bit) (word aligned address)	I	000_100
SWI Imm(Rs), Rd	M(Rs + Imm) ← Rd	I	001_000
LHI Rd, Imm (Rs)	Rd (16:31)← M(Rs + Imm) (Rd sign extended to 32 bit)	I	000_101
SHI Imm(Rs), Rd	M(Rs1 + Rs2) ← Rd(16:31)	I	001_001
LBI Rd, Imm (Rs)	Rd (24:31)← M(Rs + Imm) (Rd sign extended to 32 bit)	I	000_110
SBI Imm(Rs), Rd	M(Rs + Imm) ← Rd(24:31)	I	001_010

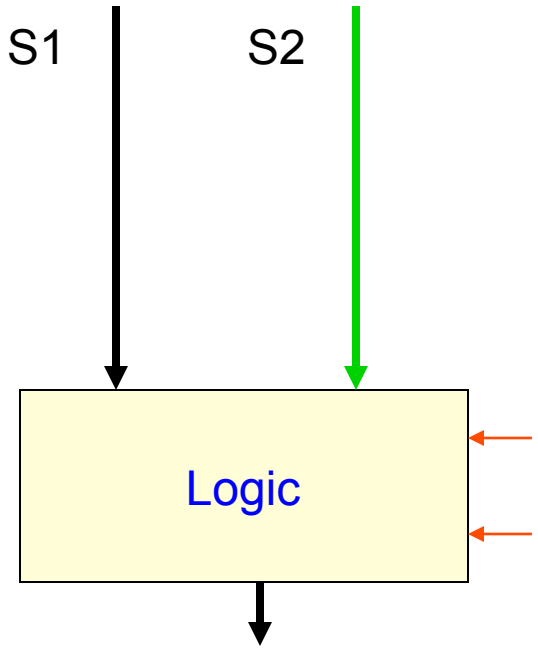
Arithmetic Logic Unit

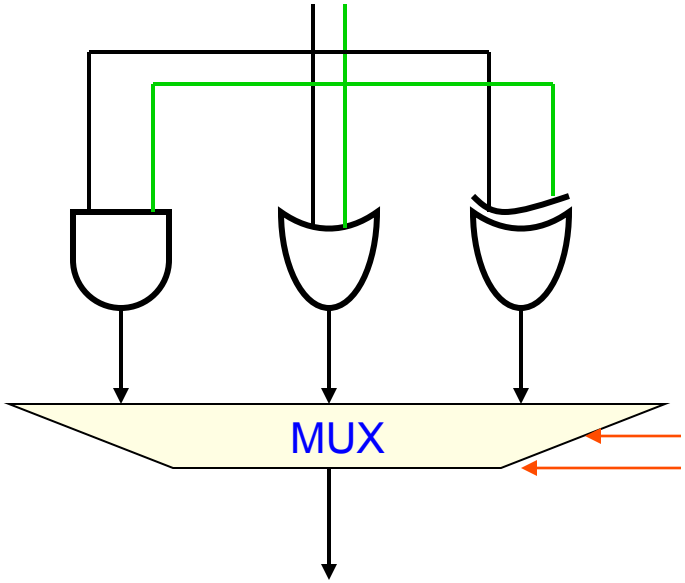


Arithmetic Logic Unit

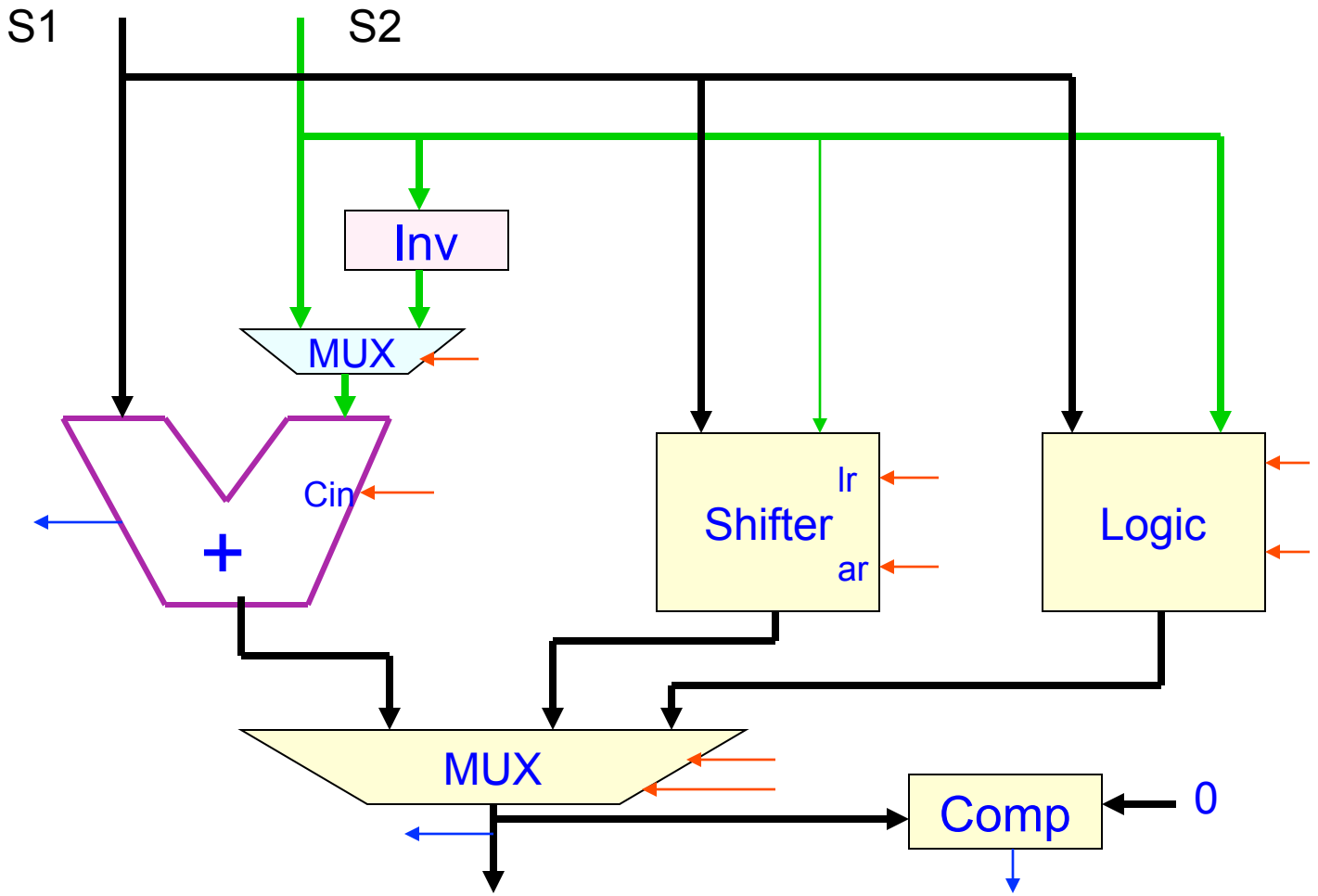


Arithmetic Logic Unit

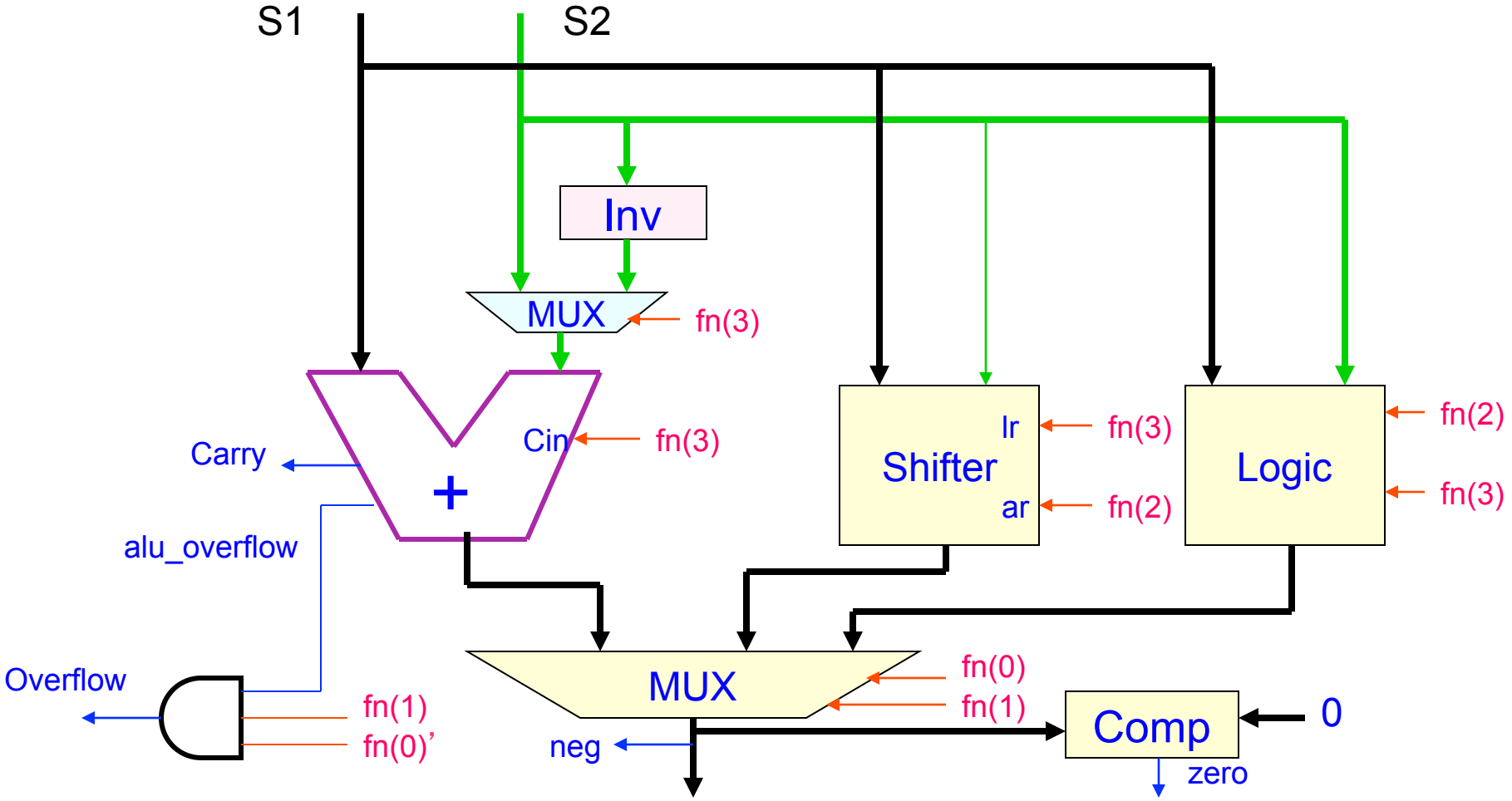




Arithmetic Logic Unit



Arithmetic Logic Unit



Thank You