CprE 381: Computer Organization and Assembly Level Programming

MIPS Arithmetic

Henry Duwe
Electrical and Computer Engineering
Iowa State University

Administrative

- Exam 1: T-7 days
  - Practice Exam Session
    - Feb 12 (tomorrow) @ 6pm in 2155 Marston
    - Download and print a copy of the practice exam
- Lab 4 will be due at your lab section
  - Late policy enforced
  - You must turn in your lab report prior to your lab section
- SIGN UP FOR YOUR TERM PROJECT TEAM!!!

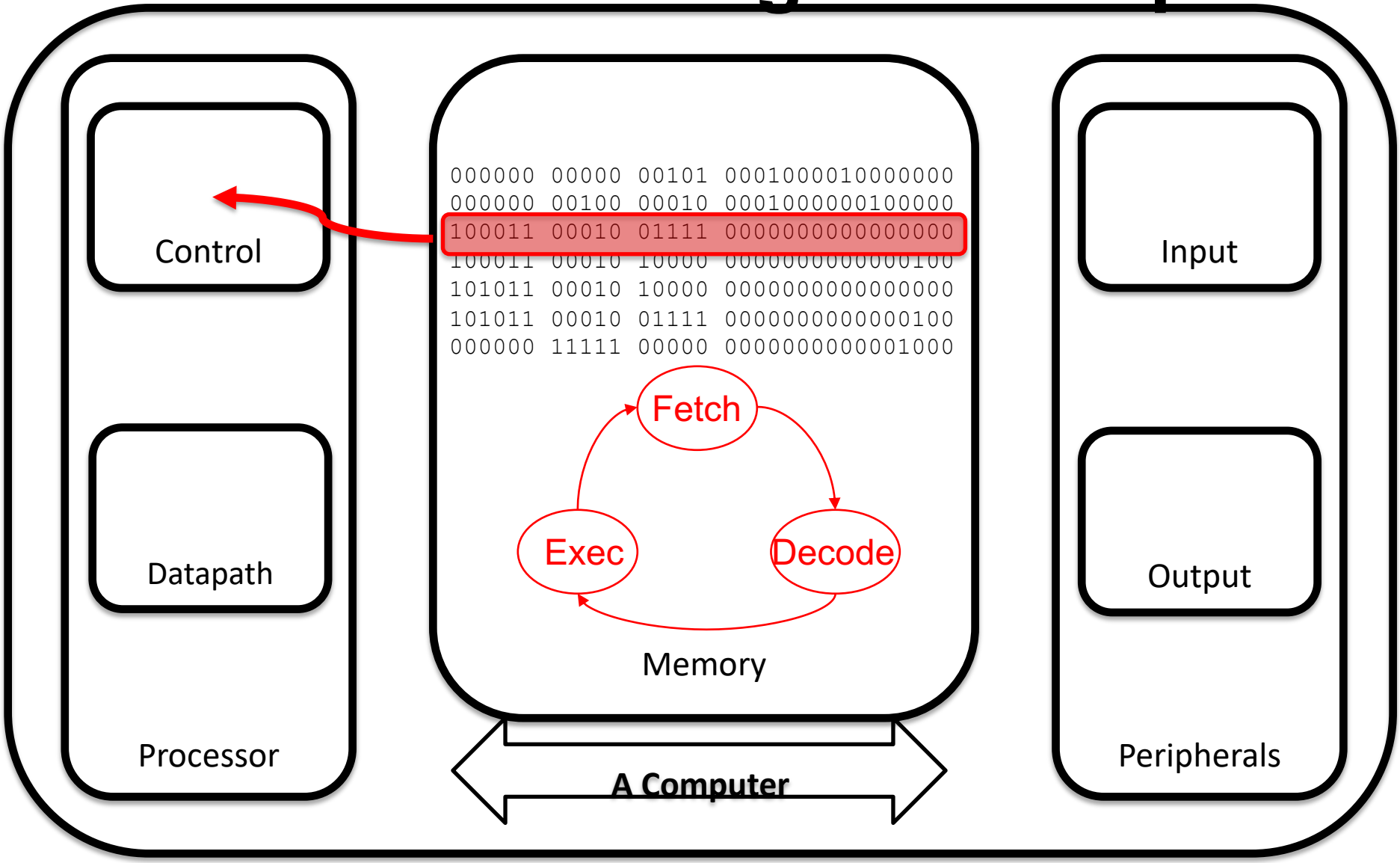
Administrative

- In lab: starting term projects!!!
  - Minimal prelab: write a list of best VHDL coding practices for your team based on the first 4 individual labs
  - The Process Statement Considered Harmful
    - Filled with sequential constructs that semantically occur in parallel with structural instantiation, concurrent dataflow statements, and other process statements
      - Challenging to reason about in designs → a lot of wasted time during debug
    - Useful for testbenches
      - Clk generation
      - Sequential input assignment with wait
      - Self-checking aspects (will see in the automated test framework for part 2 and beyond)
    - Useful to describe sequential components
      - DFFs, registers, and memory structures
      - Already have these components – don't use process statements in your future designs!

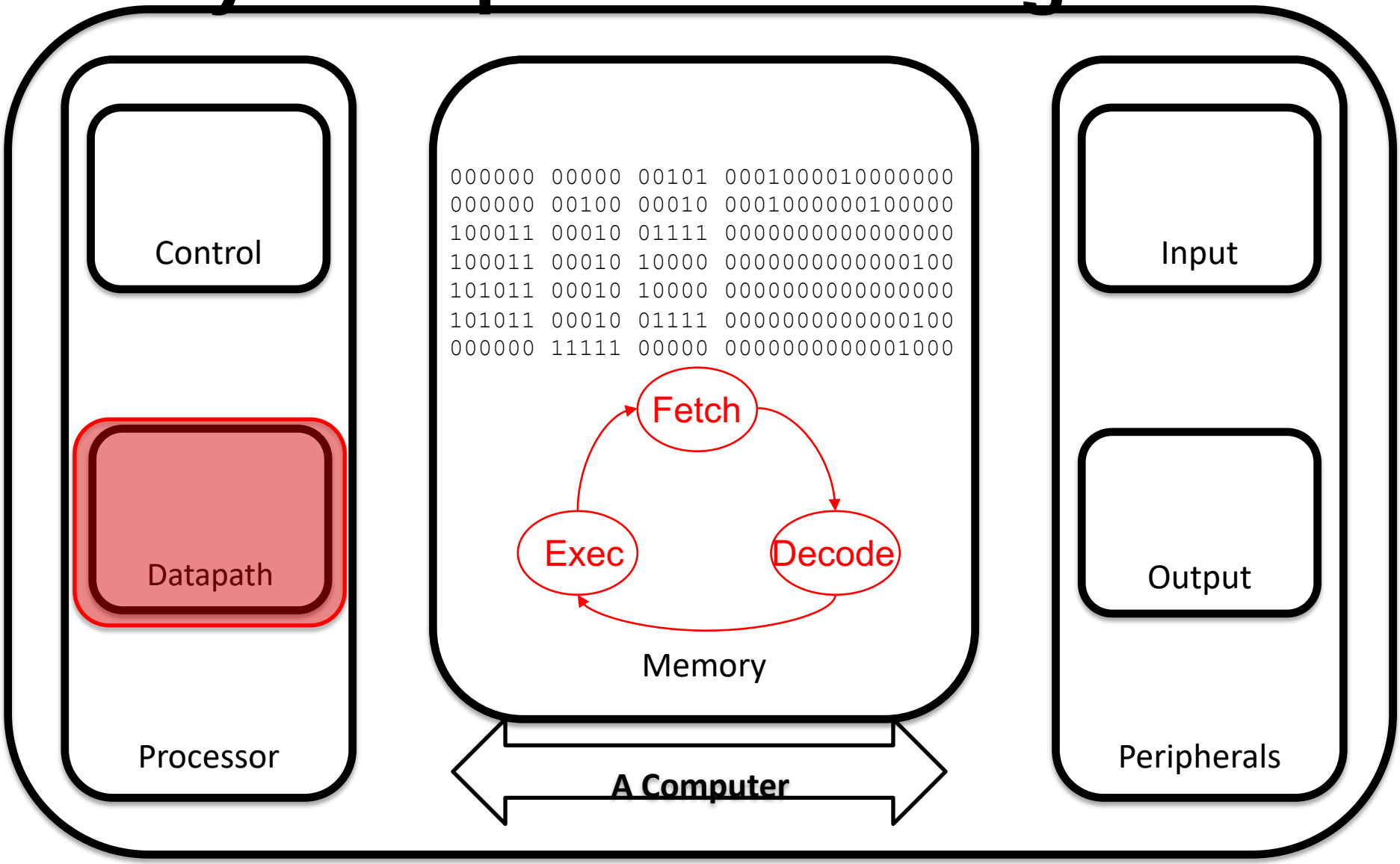
Administrative

Great moments are born from great opportunities. – Herb Brooks

Review: Stored Program Computer



Today: Datapath/ALU Design



Review: MIPS Simple Arithmetic

Instruction	Example	Meaning	Comments
add	`add $1,$2,$3`	`$1 = $2 + $3`	3 operands; Overflow
subtract	`sub $1,$2,$3`	`$1 = $2 - $3`	3 operands; Overflow
add immediate	`addi $1,$2,100`	`$1 = $2 + 100`	+ constant; Overflow
add unsigned	`addu $1,$2,$3`	`$1 = $2 + $3`	3 operands; No overflow
sub unsigned	`subu $1,$2,$3`	`$1 = $2 - $3`	3 operands; No overflow
add imm unsign	`addiu $1,$2,100`	`$1 = $2 + 100`	+ constant; No overflow

- Your task: check out logical and shift instructions

Review: MIPS Simple Arithmetic

Instruction	Example	Meaning	Comments
add	`add $1,$2,$3`	`$1 = $2 + $3`	3 operands; Overflow
subtract	`sub $1,$2,$3`	`$1 = $2 - $3`	3 operands; Overflow
add immediate	`addi $1,$2,100`	`$1 = $2 + 100`	+ constant; Overflow
add unsigned	`addu $1,$2,$3`	`$1 = $2 + $3`	3 operands; No overflow
sub unsigned	`subu $1,$2,$3`	`$1 = $2 - $3`	3 operands; No overflow
add imm unsign	`addiu $1,$2,100`	`$1 = $2 + 100`	+ constant; No overflow

Adder

Register File (nearly every instruction)

One's Complementor

Overflow Detection

Sign Extender

MUX to select between outputs

- OR, AND, NOR, XOR, SLL, SRL, SRA

Gate Arrays

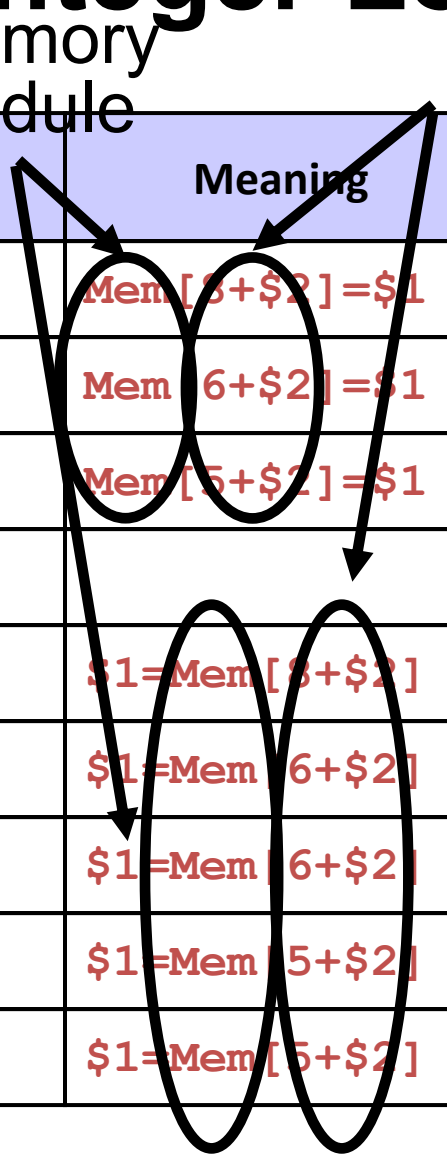
Shifter

Review: MIPS Integer Load/Store

Instruction	Example	Meaning	Comments
store word	`sw $1,8($2)`	`Mem[8+$2]=$1`	Store word
store half	`sh $1,6($2)`	`Mem[6+$2]=$1`	Stores only lower 16b
store byte	`sb $1,5($2)`	`Mem[5+$2]=$1`	Stores only lowest byte
load word	`lw $1,8($2)`	`$1=Mem[8+$2]`	Load word
load halfword	`lh $1,6($2)`	`$1=Mem[6+$2]`	Load half; sign extend
load half unsign	`lhu $1,6($2)`	`$1=Mem[6+$2]`	Load half; zero extend
load byte	`lb $1,5($2)`	`$1=Mem[5+$2]`	Load byte; sign extend
load byte unsign	`lbu $1,5($2)`	`$1=Mem[5+$2]`	Load byte; zero extend

Review: MIPS Integer Load/Store

Instruction	Example	Meaning	Comments
store word	`sw $1,8($2)`	`Mem[8+$2]=$1`	Store word
store half	`sh $1,6($2)`	`Mem[6+$2]=$1`	Stores only lower 16b
store byte	`sb $1,5($2)`	`Mem[5+$2]=$1`	Stores only lowest byte
load word	`lw $1,8($2)`	`$1=Mem[8+$2]`	Load word
load halfword	`lh $1,6($2)`	`$1=Mem[6+$2]`	Load half; sign extend
load half unsign	`lhu $1,6($2)`	`$1=Mem[6+$2]`	Load half; zero extend
load byte	`lb $1,5($2)`	`$1=Mem[5+$2]`	Load byte; sign extend
load byte unsign	`lbu $1,5($2)`	`$1=Mem[5+$2]`	Load byte; zero extend



Review: MIPS Control Flow

Instruction	Example	Meaning
jump	`j L`	`goto L`
jump register	`jr $1`	`goto value in $1`
jump and link	`jal L`	`goto L and set $ra`
jump and link register	`jalr $1`	`goto $1 and set $ra`
branch equal	`beq $1,$2,L`	`if ($1 == $2) goto L`
branch not equal	`bne $1,$2,L`	`if ($1 != $2) goto L`
branch less than 0	`bltz $1,L`	`if ($1 < 0) goto L`
branch less than / eq 0	`blez $1,L`	`if ($1 <= 0) goto L`
branch greater than 0	`bgtz $1,L`	`if ($1 > 0) goto L`
branch greater than / eq 0	`bgez $1,L`	`if ($1 >= 0) goto L`

Review: MIPS Control Flow

Instruction	Example	Meaning
jump	`j L`	`goto L`
jump register	`jr $1`	`goto value in $1`
jump and link	`jal L`	`goto L and set $ra`
jump and link register	`jalr $1`	`goto $1 and set $ra`
branch equal	`beq $1,$2,L`	`if ($1 == $2) goto L`
branch not equal	`bne $1,$2,L`	`if ($1 != $2) goto L`
branch less than 0	`bltz $1,L`	`if ($1 < 0) goto L`
branch less than / eq 0	`blez $1,L`	`if ($1 <= 0) goto L`
branch greater than 0	`bgtz $1,L`	`if ($1 > 0) goto L`
branch greater than / eq 0	`bgez $1,L`	`if ($1 >= 0) goto L`

MUX for PC (eventually)

MUX for reg file data input

Comparators:

- Equality (sub + OR tree or XOR + OR tree)
- < or > 0 (sub + sign bit)

Adder for PC+4+Offset

Review: MIPS Comparisons

Instruction	Example	Meaning	Comments
set less than	`slt $1,$2,$3`	`$1=($2<$3)`	Comp less than signed
set less than imm	`slti $1,$2,100`	`$1=($2<100)`	Comp w/const signed
set less unsgn	`sltu $1,$2,$3`	`$1=($2<$3)`	Comp less than unsigned
slt imm unsgn	`sltiu $1,$2,100`	`$1=($2<100)`	Comp w/const unsigned

- C

```
if (8 < a)goto Exceed
slti $v0, $a0, 9          # $v0 = $a0<9
beq $v0, $zero, Exceed # goto if $v0==0
```

Review: MIPS Comparisons

Instruction	Example	Meaning	Comments
set less than	`slt $1,$2,$3`	`$1=($2<$3)`	Comp less than signed
set less than imm	`slti $1,$2,100`	`$1=($2<100)`	Comp w/const signed
set less unsgn	`sltu $1,$2,$3`	`$1=($2<$3)`	Comp less than unsigned
slt imm unsgn	`sltiu $1,$2,100`	`$1=($2<100)`	Comp w/const unsigned

Comparators:

- < or > (sub → sign bit)

Review: Addition and Subtraction

- Just like in grade school (carry/borrow 1s)

```
  0111        0111        0110
+ 0110      - 0110      - 0101
```

- Two's complement operations easy
  - Subtraction using addition of negative numbers

```
  0111
+ 1010
```

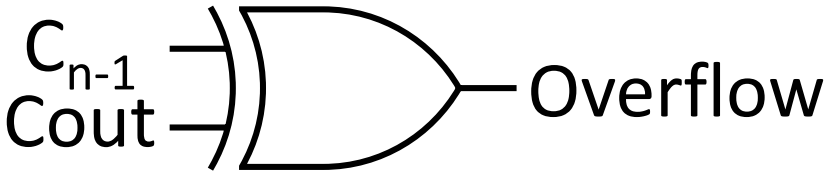
- Overflow (result too large for finite computer word):
  - e.g., adding two n-bit numbers does not yield an n-bit number

```
  0111
+ 0001
  1000
```

note that overflow term is somewhat misleading, it does not mean a carry "overflowed"

Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - Overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ?



Detecting Overflow

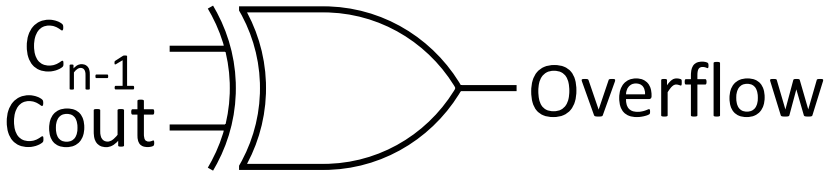
- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
-

In-class Assessment!

Access Code: Boom

Note: sharing access code to those outside of classroom or using access while outside of classroom is considered cheating

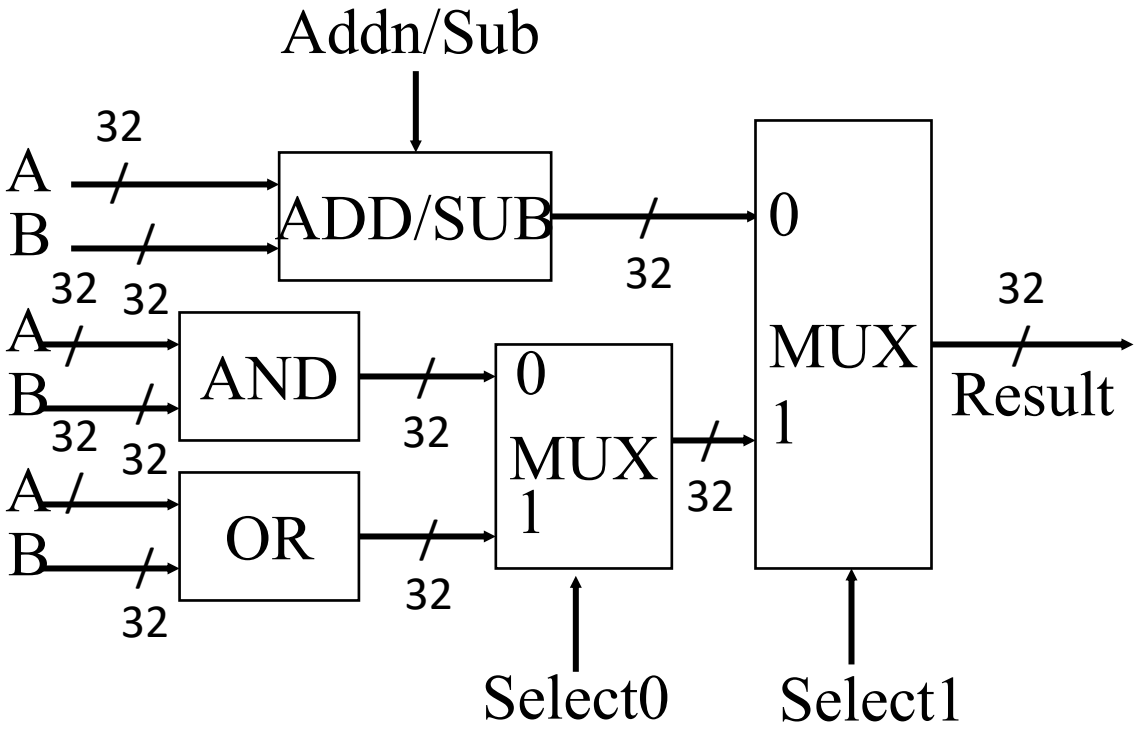
- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ?



Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address saved for resumption
- Details are software system / language dependent
  - E.g. flight control software v. CprE 381 homework assignment
- Don't always want to detect overflow (`addu`, `addiu`, `subu`)

Preview: ALU Design Description



Acknowledgments

- These slides contain material developed and copyright by:
  - Joe Zambreno (Iowa State)
  - David Patterson (UC Berkeley)
  - Mary Jane Irwin (Penn State)
  - Christos Kozyrakis (Stanford)
  - Onur Mutlu (Carnegie Mellon)
  - Krste Asanović (UC Berkeley)