CprE 381: Computer Organization and Assembly Level Programming

Henry Duwe
Electrical and Computer Engineering
Iowa State University

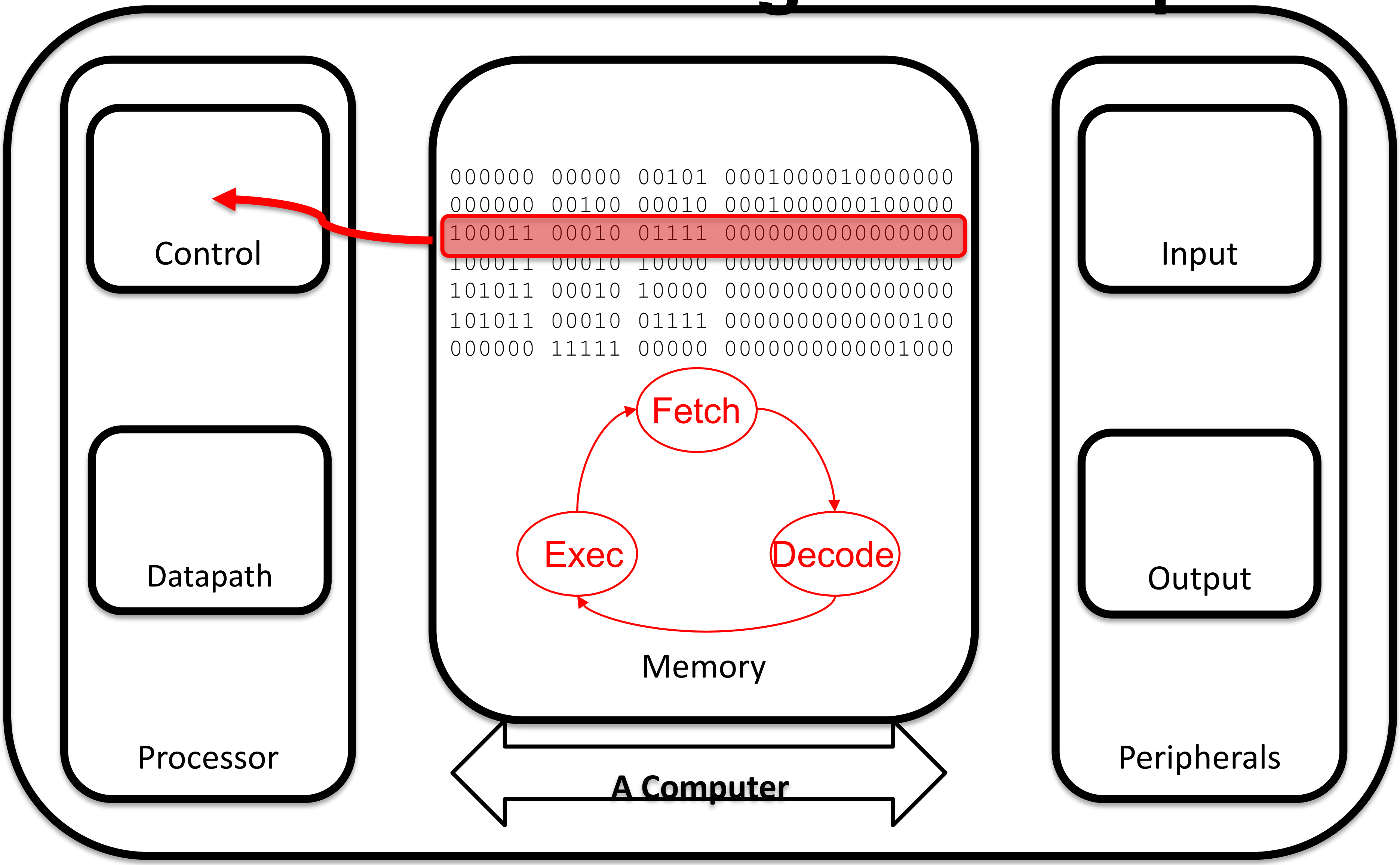
Administrative

- HW1 posted (due next Mon Jan 28)
  - Hard deadline – submit what you have several minutes before deadline (at least)
  - Typeset: in the future, generate figures with professional software (don’t include snapshots of paper or whiteboard)
    - Visio, Powerpoint, etc. are all free
    - You will begin to lose points
- Labs
  - Prelab must be completed ***prior*** to start of lab
    - Starting with Lab 3, separate submission assignment – no points if not submitted by start of your lab
  - Reminder:
    - As noted on the eval sheet, demoing does not mean you get 100% on that portion – you must turn in the lab
    - Each 24hr period after the deadline max score of additional work is reduced by 10%
  - Submit what you have done before start of lab!

Administrative

- Lab 2
  - Download new version (V2)
  - Fixed and2 naming
  - Added **required** feedback portion to lab report template

Review: Stored Program Computer



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

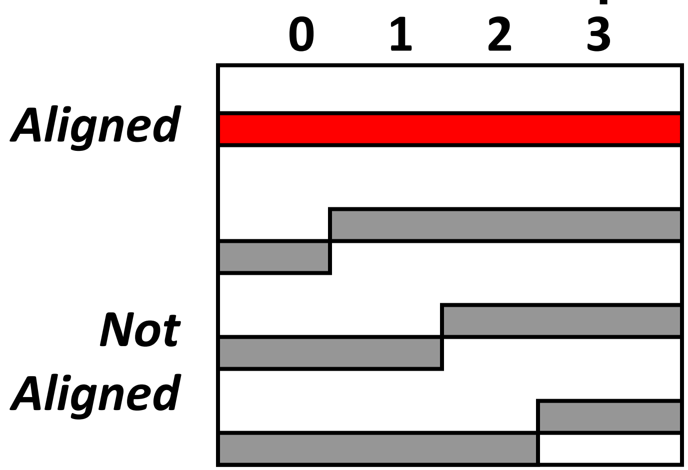
Review: MIPS Integer Load/Store

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

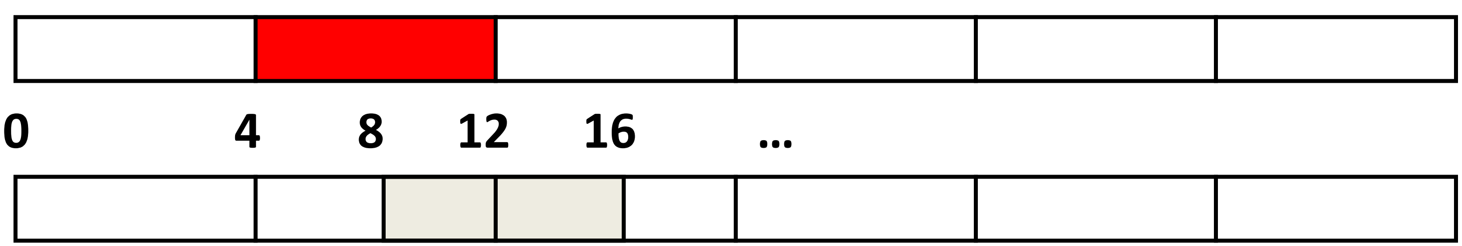
Memory

Alignment Restrictions

- In MIPS, data is required to fall on addresses that are multiples of the data size



- Consider word (4 byte) memory access



Alignment Restrictions (cont.)

- C example

What is the size of this structure?

```
struct foo {
    char sm;
    short med;
    char sm1;
    int lrg;
}
```

0	1	2	3	4	5	6	7	8	9	10	11
sm		med		sm1				lrg			

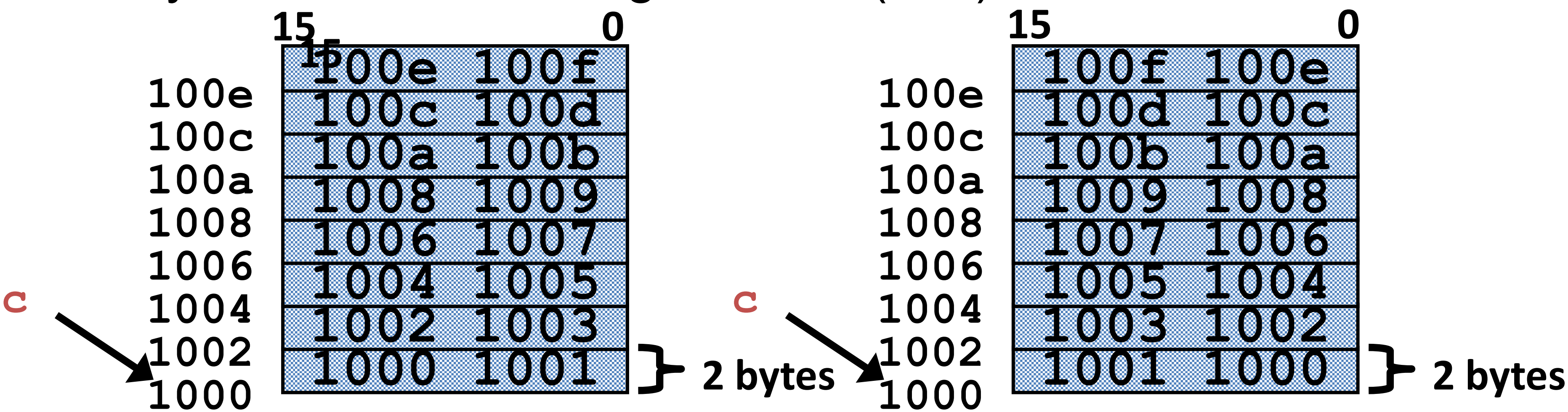
- Historically
  - Early machines (IBM 360 in 1964) required alignment
  - Removed in 1970s to reduce impact on programmers
  - Reintroduced by RISC to improve performance
- Also introduces challenges with memory organization with virtual memory, etc.

So who enforces this?

- Unaligned memory access
  - Causes exception (more about exceptions after Spring Break)

Bonus: Endianness

- Byte address ordering within a (half)word

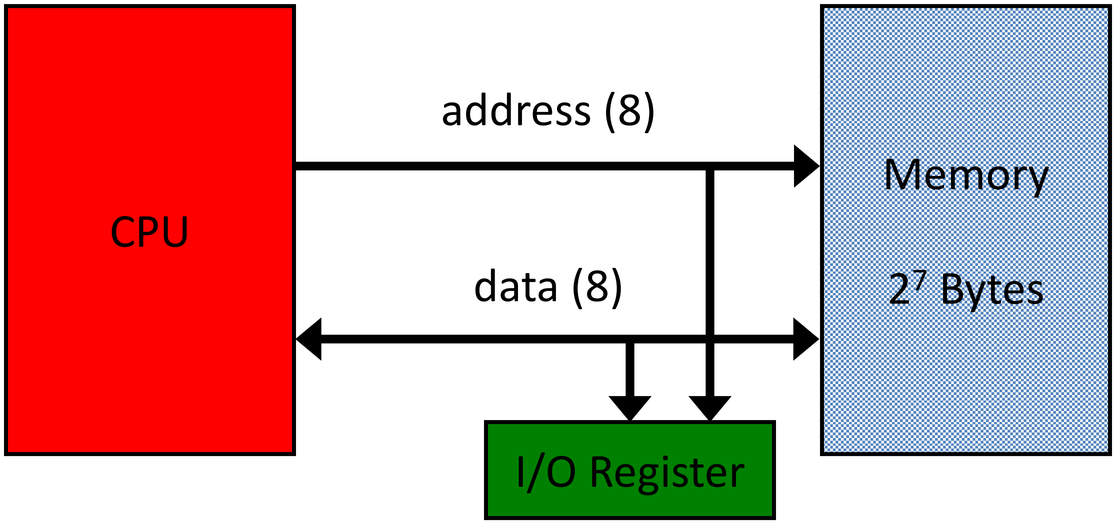


Big Endian **Little Endian**

- How do I figure this out if I don't have the manual? Or how do I verify the manual is correct?

Memory Mapped I/O

- Data transfer instructions can be used to move data to and from I/O device registers
- A load operation moves data from a an I/O device register to a CPU register and a store operation moves data from a CPU register to a I/O device register



I/O register at address 0x80 (128)

C Code Example

Simple C procedure: sum_pow2=2^{b+c}

```c
int sum_pow2 (int b, int c)
{
  int pow2[8] = {1, 2, 4, 8, 16, 32, 64, 128};
  int a, ret;
  a = b + c;
  if (a < 8)
     ret = pow2[a];
  else
     ret = 0;
  return(ret);
}
```

TODAY: Control Flow

Changing Control Flow

- One of the distinguishing characteristics of computers is the ability to evaluate conditions and change control flow
  - All instructions so far just manipulate data (we've built a 'calculator' instead of a 'computer')
- C examples:
  - If-then-else
  - Loops
  - Case statements
  - Any others?
- MIPS Assembly
  - Conditional control flow changes are known as <u>branches</u>
  - Unconditional control flow changes are known as <u>jumps</u>
  - The target of a branch/jump is a <u>label</u>

Conditional: Equality

- The simplest conditional test is the **beq** instruction for equality

```
beq reg1, reg2, label
```

- Consider the code

```
if (a == b) goto L1
   // Do something
L1: // Continue
```

- Using the **beq** instruction:

```
    beq $s0, $s1, L1
    # Do something
L1: # Continue
```

Conditional: Equality

- The simplest conditional test is the `beq` instruction for equality

```
beq reg1, reg2, label
```

- Confusing, but also equivalent to

```
if (!(a == b)) {
   // Do something
}
L1: // Continue
```

- Using the `beq` instruction:

```
beq $s0, $s1, L1
# Do something
L1: # Continue
```

Conditional: Not Equal

- The **bne** instruction for not equal

```
bne reg1, reg2, label
```

- Consider the code

```
  if (a != b) goto L1
  // Do something
L1: // Continue
```

- Using the **bne** instruction:

```
  bne $s0, $s1, L1
  # Do something
L1: # Continue
```

Unconditional: Jumps

- The `j` instruction jumps to a label

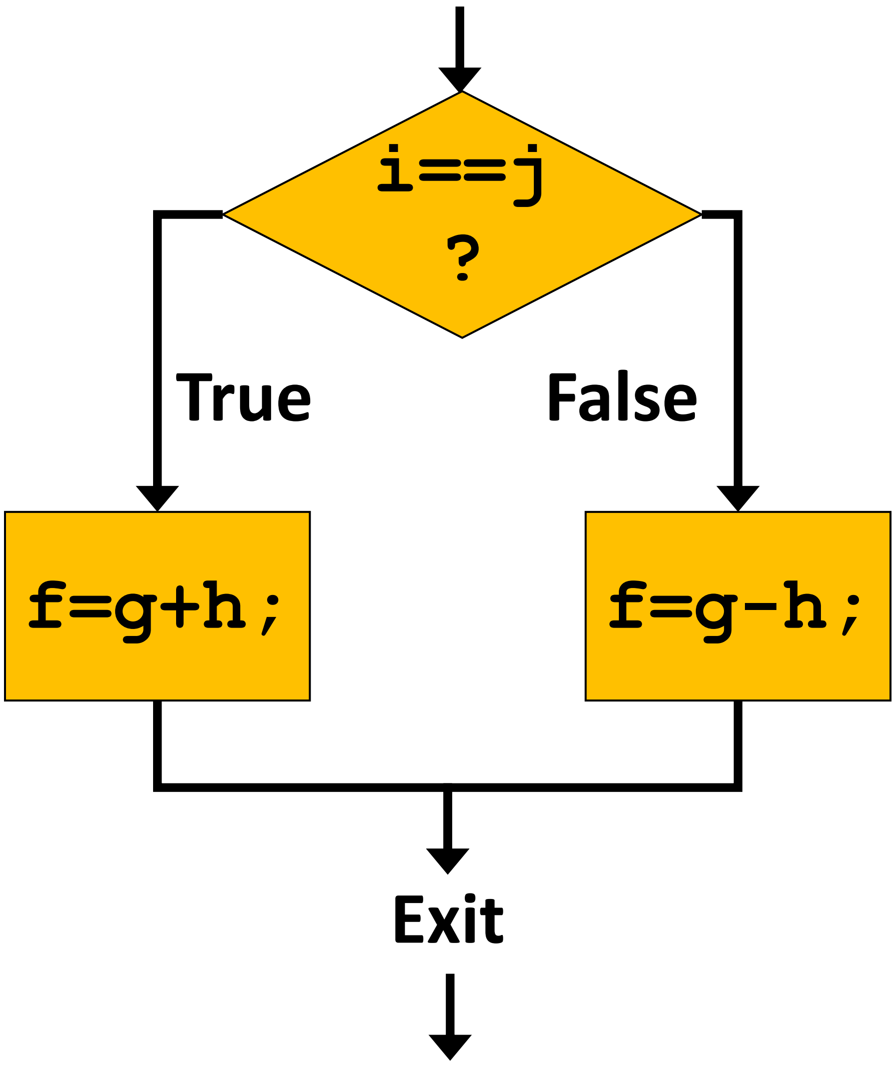
```
j label
```

- Similar to C goto statement

If-Then-Else Example

- Consider the code

```
if (i == j) {
  f = g + h;
}
else {
  f = g - h;
}
```

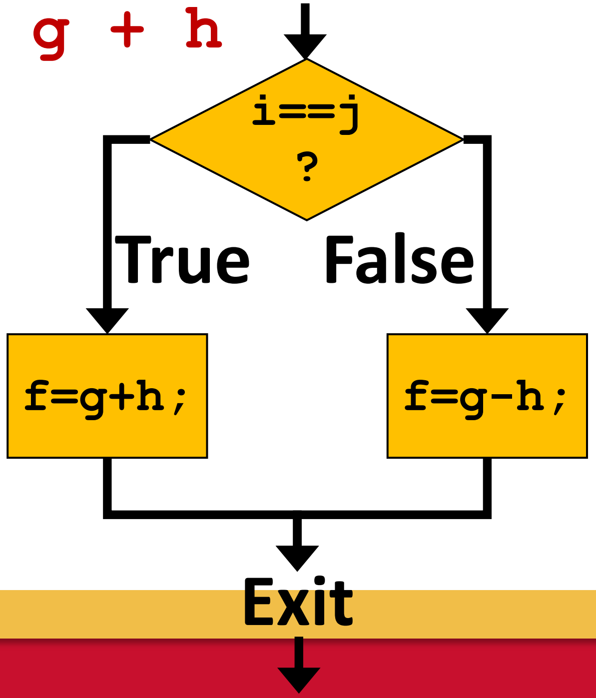


If-Then-Else Example

- Create labels and use equality instruction

```
         beq $s3, $s4, True # Branch if i==j
         sub $s0, $s1, $s2  # f = g - h
         j Exit             # Go to Exit
True:    add $s0, $s1, $s2  # f = g + h
Exit:
```

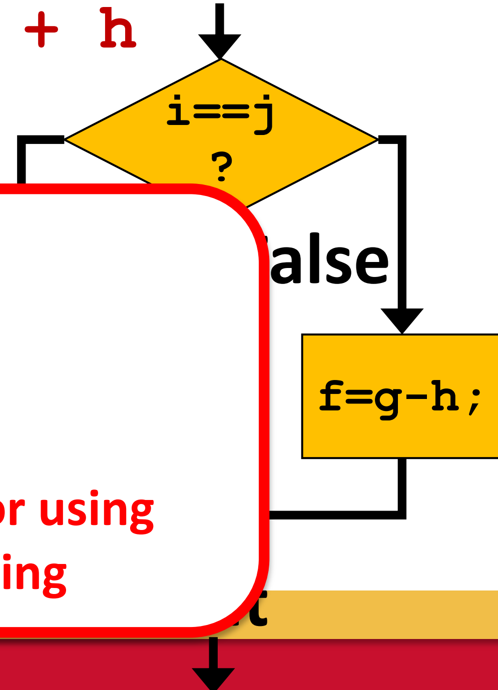
- Can you rewrite with **bne**?



If-Then-Else Example

- Create labels and use equality instruction

```
        beq $s3, $s4, True # Branch if i==j
        sub $s0, $s1, $s2  # f = g - h
        j Exit             # Go to Exit
True:   add $s0, $s1, $s2  # f = g + h
Exit:
```



In-class Assessment!

Access Code: 380++

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

Other Comparisons

- Other conditional arithmetic operators are useful in evaluating conditional expressions using <, >, <=, >=
- Register is “set” to 1 when condition is met

- Consider the following C code

```
if (f < g) goto Less;
```

- Solution

```
slt $t0, $s0, $s1     # $t0 = 1 if $s0<$s1
bne $t0, $zero, Less # goto Less if $t0!=0
```

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 (a < 8)
```

- MIPS assembly

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

Preview: C Code Example

Simple C procedure: sum_pow2=2^{b+c}

```c
int sum_pow2 (int b, int c)
{
  int pow2[8] = {1, 2, 4, 8, 16, 32, 64, 128};
  int a, ret;
  a = b + c;
  if (a < 8)
     ret = pow2[a];
  else
     ret = 0;
  return(ret);
}
```

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)