CprE 381: Computer Organization and Assembly Level Programming

Henry Duwe

Electrical and Computer Engineering

Iowa State University

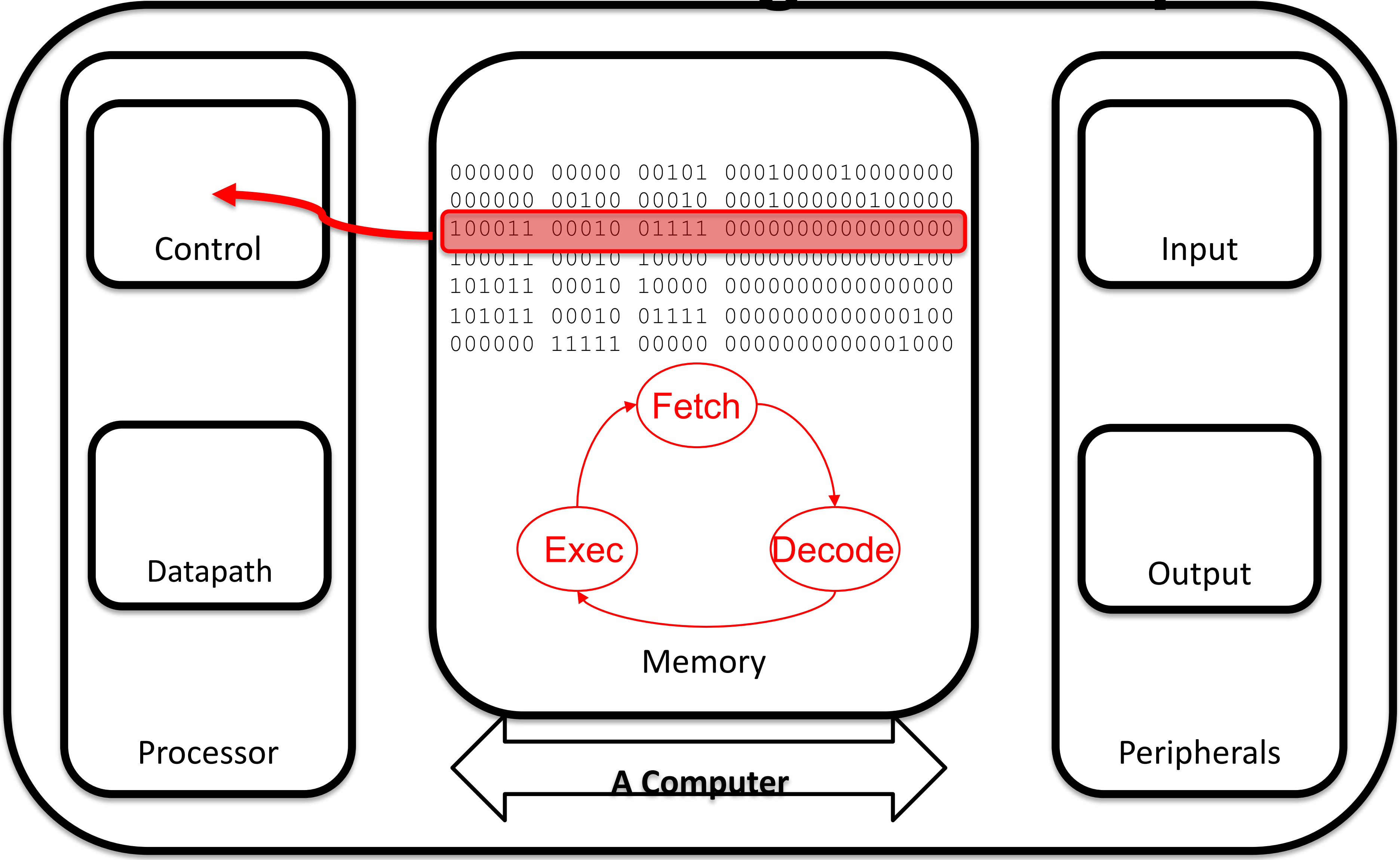
Administrative

- HW1 posted (due Jan 28b – TODAY!)
  - 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
  - Show your work!
    - HW intended to be formative → in order to get feedback, need to show process
    - Partial credit if answer is wrong
  - ABI vs ISA vs uArch → Who cares?
    - ISA is the interface between HW and SW
    - uArch (i.e., implementations) is what the HW designer can choose
    - ABI further constrains what SW will interoperate

Administrative

- 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
  - Submit what you have done before start of lab!
- Added Prof Office Hours:
  - M 10am (following lecture, walk back to office)
- TA Office Hours
  - You may go to any TA's office hours
  - You may demo at office hours, although I'd prefer you do so at the TA for your section
  - All office hours are located in 2050 Coover
  - **Ashraf: M 11am – Noon**
  - **Ryan: T 11am – Noon**
  - **Trent: W Noon – 1pm**
  - **Rohit: R 3pm – 4pm**

Review: Stored Program Computer



```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```

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

More 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`

Preview: C Code Example

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

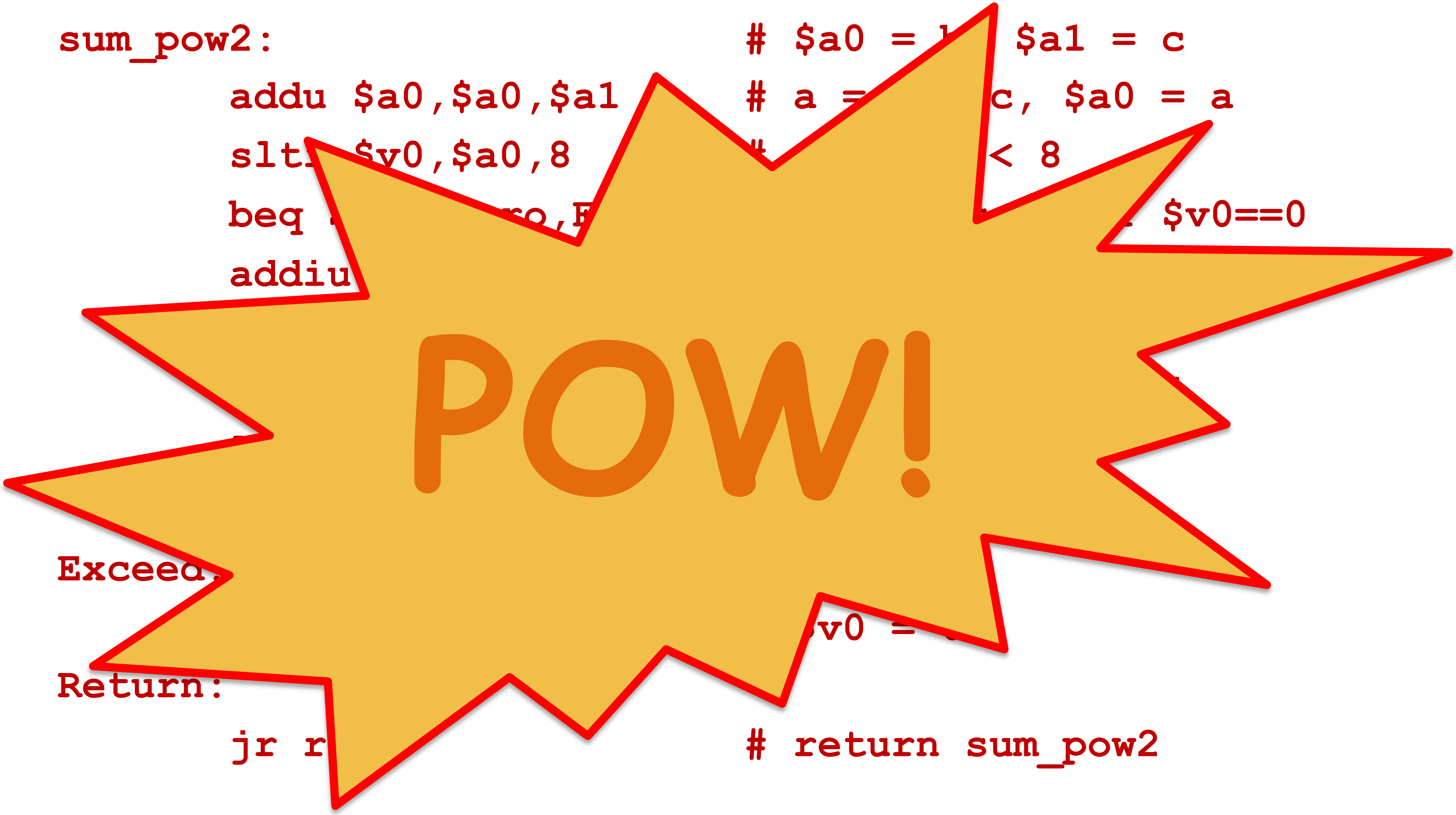
```c
int sum_pow2 (int b, int c)
{
  int pow2[8] = {1, 2, 4, 8, 16, 32, 64, 128};
  int a, ret;
  a = b + c;
  if (a < 8)
     ret = pow2[a];
  else
     ret = 0;
  return(ret);
}
```

Equivalent MIPS Assembly

```
sum_pow2:                       # $a0 = b     $a1 = c
        addu $a0,$a0,$a1        # a =       c, $a0 = a
        slti $v0,$a0,8          #            < 8
        beq       ro,E                     $v0==0
        addiu

Exceed

                                   v0 =
Return:
        jr r                    # return sum_pow2
```



Equivalent MIPS Assembly

```
sum_pow2:                     # $a0 = b, $a1 = c
        addu $a0,$a0,$a1      # a = b + c, $a0 = a
        slti $v0,$a0,8        # $v0 = a < 8
        beq $v0,$zero,Exceed  # goto Exceed if $v0==0
        addiu $v1,$sp,8       # $v1 = pow2 address
        sll $v0,$a0,2         # $v0 = a*4
        addu $v0,$v0,$v1      # $v0 = pow2 + a*4
        lw $v0,0($v0)         # $v0 = pow2[a]
        j Return              # goto Return
Exceed:
        addu $v0,$zero,$zero  # $v0 = 0
Return:
        jr ra                 # return sum_pow2
```

Support for Simple Branches Only

- Notice there is no branch less than instruction for comparing two registers?
  - The reason is that such an instruction would be too complicated and might require a longer cycle time
  - Therefore, two conditionals that do not compare against zero take at least two instructions
    - First is a set
    - Second is a conditional branch

- We'll see this later as a design trade-off
  - Less time per instruction vs. fewer instructions
    - How do you decide what to do?
  - Other RISC ISAs made a different choice (e.g. HP's PA-RISC)

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) goto Exceed
if (a <= 8)goto Exceed
if (8 < a) goto Exceed
if (8 <= a)goto Exceed
```

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) goto Exceed
```

```
slti $v0, $a0, 8          # $v0 = $a0<8
bne $v0, $zero, Exceed # goto if $v0!=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)goto Exceed
```

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

```
slti $v0, $a0, 9          # $v0 = $a0<9
beq $v0, $zero, Exceed # goto if $v0!=0
```

MIPS Comparisons

Instruction	Example	Meaning	Comments
set less than	`slt $1,$2,$3`	`$1=($2<$3)`	Comp less than signed
set l			
set l			
slt in			

In-class Assessment!

Access Code: <=|>?

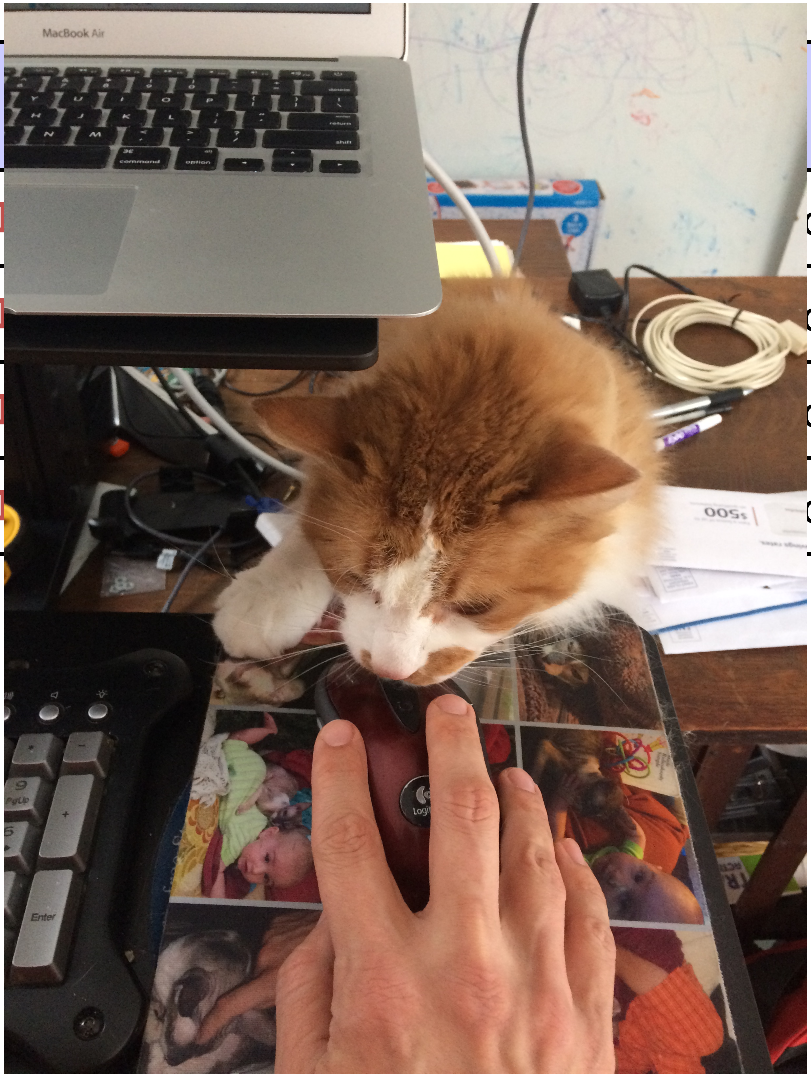
Note: sharing access code to those outside of classroom or using access while outside of classroom is considered cheating

- (

```
if (8 <= a)goto Exceed
```

MIPS Comparisons

Instruction		Comments
set less than	sl	omp less than signed
set less than imm	sl	omp w/const signed
set less unsgn	sl	omp less than unsigned
slt imm unsgn	sl	omp w/const unsigned



- C

```
if (8 <=

slti $v0,                $a0<8
beq $v0, $               f $v0!=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 (8 <= a)goto Exceed

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

While Loops in C

- Consider a **`while`** loop

```
while (A[i] == k)
      i = i + j;
```

- MIPS assembly loop
- Assume **`i=$s0`**, **`j=$s1`**, **`k=$s2`**, **`&A=$s3`**

While Loops in C

- Consider a **`while`** loop

```
while (A[i] == k)
      i = i + j;
```

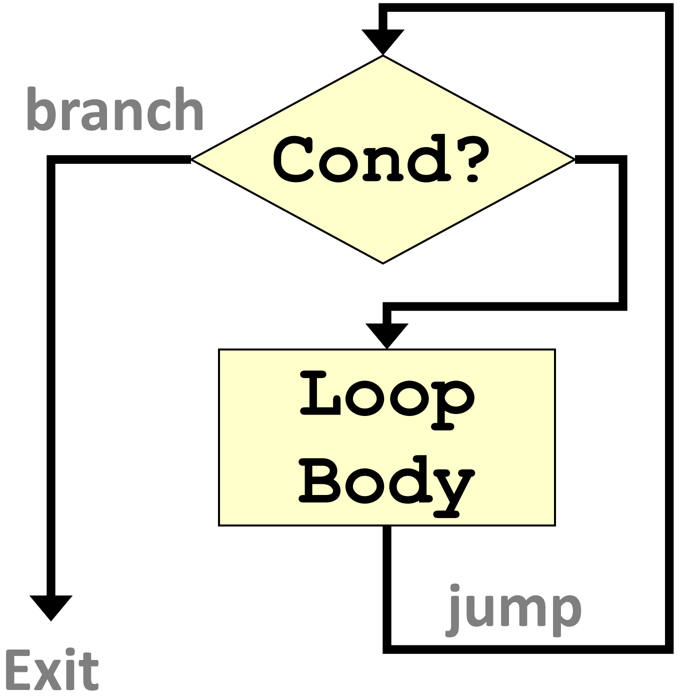
- MIPS assembly loop
- Assume **`i=$s0`**, **`j=$s1`**, **`k=$s2`**, **`&A=$s3`**

```
Loop: sll $t0, $s0, 2        # $t0 = 4 * i
      addu $t1, $t0, $s3     # $t1 = &(A[i])
      lw $t2, 0($t1)         # $t2 = A[i]
      bne $t2, $s2, Exit     # goto Exit if !=
      addu $s0, $s0, $s1     # i = i + j
      j Loop                 # goto Loop
  Exit:
```

- Basic block:
  - Maximal sequence of instructions without branches or branch targets

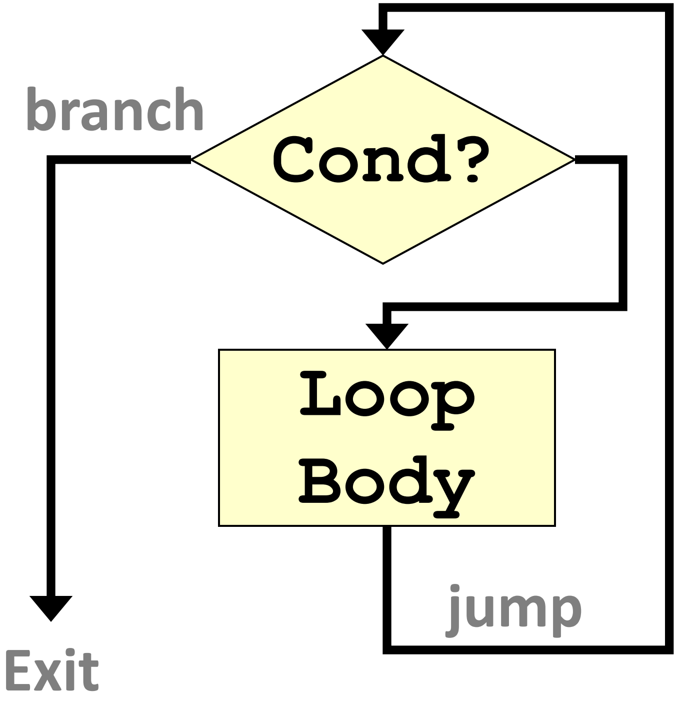
Improve Loop Efficiency

- Code uses two branches per iteration:

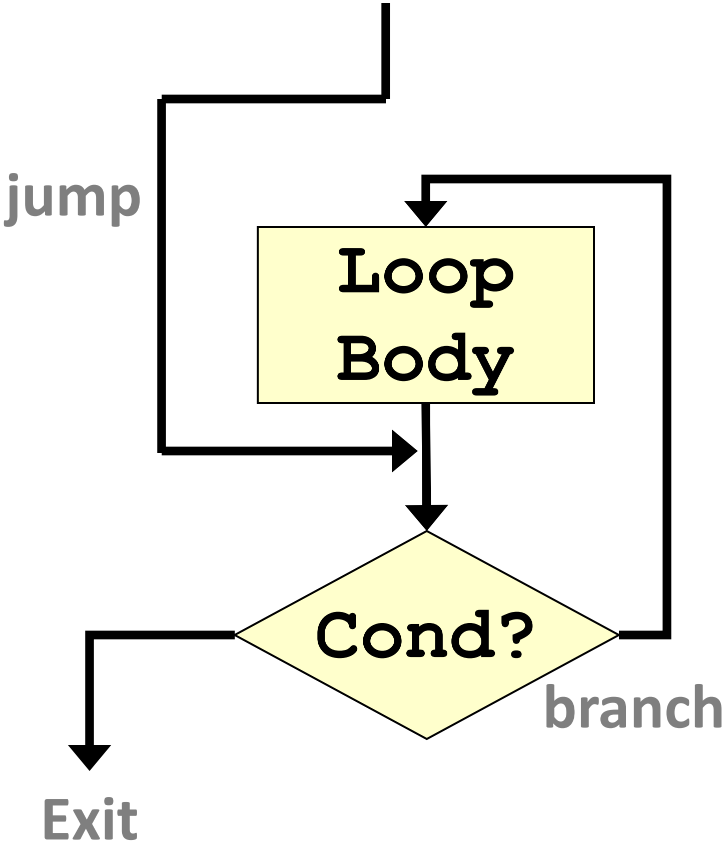


Improve Loop Efficiency

- Code uses two branches per iteration:



- More efficient structure:



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)