Decisions: `if` Statements

- **2 kinds of `if` statements in HLL (e.g. C)**
  - `if (`*condition*`)` *clause*
  - `if (`*condition*`)` *clause1* `else` *clause2*
- **Rearrange 2nd `if` into following:**

```
if  (condition) goto L1;
      clause2;
      go to L2;
L1: clause1;
 L2:
```

  - Not as elegant as if - else, but same meaning

MIPS Decision Instructions

- **Decision instruction in MIPS:**
  - `beq  register1, register2, L1`
  - `beq` is 'Branch if (registers are) equal'
    Same meaning as :

```
if   (register1==register2) goto L1
```

- **Complementary MIPS decision instruction**
  - `bne  register1, register2, L1`
  - `bne` is 'Branch if (registers are) not equal'
    Same meaning as :

```
if   (register1!=register2) goto L1
```

- **Called conditional branches**

MIPS Goto Instruction

- **In addition to conditional branches, MIPS has an unconditional branch:**

```
b label
```

- **Called a Jump Instruction: jump (or branch) directly to the given label without needing to satisfy any condition**
- **Same meaning as :**

```
goto label
```

- **Technically, it's the same as:**

```
beq   $0, $0, label
```

since it always satisfies the condition.

14

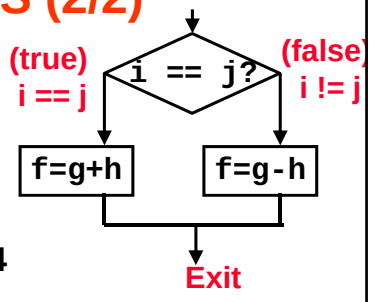
Compiling `if` into MIPS (2/2)

- **Compile by hand**

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

- **Use this mapping:**

f: s0, g: s1, h: s2, i: s3, j: s4



- **Final compiled MIPS code:**

```
      beq s3, s4, True  # branch i==j
      sub s0, s1, s2    # f=g-h(false)
      b   Fin           # go to Fin
True:
      add   s0,s1,s2    # f=g+h (true)
Fin:
```

- **Note: Compilers automatically create labels to handle decisions (branches) appropriately. Generally not found in HLL code.**

15

Branching Assembly Instructions

```
beq   Rs1,  Rs2,  Label      # goto Label if Rs1==Rs2
bne   Rs1,  Rs2,  Label      # goto Label if Rs1!=Rs2

blt   Rs1,  Rs2,  Label      #goto Label if Rs1 < Rs2
bgt   Rs1,  Rs2,  Label      #goto Label if Rs1 > Rs2
ble   Rs1,  Rs2,  Label      #goto Label if Rs1 <= Rs2
bge   Rs1,  Rs2,  Label      #goto Label if Rs1 >= Rs2
b         Label              #unconditional goto Label

jal sub                      #Jump and link to sub(sub is the label
                             starting the subroutine sub
jr  Rs                       #jump to address specified by register
                             Rs
```

16

Example

- Get a few numbers from keyboard until you see zero
- Calculate their sum.

17

repeat-until loop

repeat … until v0=0

```
loop:
        jal getnum              # get a number from keyboard
        beq v0, zero, finish    # if v0=zero break the loop to
                                    finish
        add s1, s1, v0          # s1 is the sum
        b       loop

finish:
```

Other loop Structures

- while
- do while
- for

Key Concept: Though there are multiple ways of writing a loop in MIPS, conditional branch is key to decision making

Exercise: get N numbers from keyboard, and calculate the sum.

For Loop Example

- **Total: s0**
- **Index: t0**
- **N: s1**

20

The Switch Statement

- **Choose among four alternatives depending on whether k has the value 0, 1, 2 or 3. Compile this code:**

```
switch (k) {
    case 0: f=i+j; break; /* k=0*/
    case 1: f=g+h; break; /* k=1*/
    case 2: f=g-h; break; /* k=2*/
    case 3: f=i-j; break; /* k=3*/
    }
```

21

The Switch Statement

- **This is complicated, so simplify.**
- **Rewrite it as a chain of if-else statements, which we already know how to compile:**

```
if(k==0) f= i + j;
  else if(k==1) f= g + h;
    else if(k==2) f= g - h;
      else if(k==3) f= i - j;
```

- **Further rewriting:**

```
if(k==0) f= i + j;
 else if((k-1)==0) f= g + h;
   else if((k-2)==0) f= g - h;
     else if((k-3)==0) f= i - j;
```

- **Use this mapping:**

f: s0, g: s1, h: s2, i: s3, j: s4, k: s5

22

Example: The Switch Statement

- **Final compiled MIPS code:**

```
    bne s5, 0, L1          # branch k!=0
    add s0, s3, s4         # k==0 so f=i+j
    b      Exit               # end of case so Exit
L1:
    addi   t0, s5, -1         # t0 = k-1
    bne    t0, 0, L2          # branch k != 1
    add    s0, s1, s2         # k==1 so f=g+h
    b      Exit               # end of case so Exit
L2:
    addi   t0, s5, -2         # t0=k-2
    bne    t0, 0, L3          # branch k != 2
    sub    s0, s1, s2         # k==2 so f=g-h
    b      Exit               # end of case so Exit
L3:
    addi   t0, s5, -3         # t0 = k-3
    bne    t0, 0, Exit        # branch k != 3
    sub    s0, s3, s4         # k==3 so f=i-j
Exit:
```

23

The Switch Statement

- **Choose among four alternatives depending on whether k has the value 0, 1, 2 or 3. Compile this code:**

```
switch (k) {
    case 0: f=i+j; break; /* k=0*/
    case 1: f=g+h; break; /* k=1*/
    case 2: f=g-h; break; /* k=2*/
    case 3: f=i-j; break; /* k=3*/
    Default: f = 0; break; /* k is not any of above */
    }
```

24

The Switch Statement

- **This is complicated, so simplify.**
- **Rewrite it as a chain of if-else statements, which we already know how to compile:**

```
if(k==0) f= i + j;
   else if(k==1) f= g + h;
     else if(k==2) f= g - h;
       else if(k==3) f= i - j;
         else f = 0;
```

- **Further rewriting:**

```
if(k==0) f= i + j;
 else if((k-1)==0) f= g + h;
   else if((k-2)==0) f= g - h;
     else if((k-3)==0) f= i - j;
       else f = 0;
```

- **Use this mapping:**

f: s0, g: s1, h: s2, i: s3, j: s4, k: s5

25

Example: The Switch Statement

- **Final compiled MIPS code:**

```
    bne  s5, 0, L1       # branch k!=0
    add  s0, s3, s4         # k==0 so f=i+j
    b    Exit               # end of case so Exit
  L1:
    addi    t0, s5, -1         # t0 = k-1
    bne     t0, 0, L2       # branch k != 1
    add     s0, s1, s2         # k==1 so f=g+h
    b       Exit               # end of case so Exit
  L2:
    addi    t0, s5, -2         # t0=k-2
    bne     t0, 0, L3       # branch k != 2
    sub     s0, s1, s2         # k==2 so f=g-h
    b       Exit               # end of case so Exit
  L3:
    addi    t0, s5, -3         # t0 = k-3
    bne     t0, 0, DEFAULT     # branch k != 3
    sub     s0, s3, s4         # k==3 so f=i-j
    b       Exit
  DEFAULT:
    move    s0, zero
  Exit:
```

26

References

- **Documentation**
  - MIPS-Vol2.pdf
  - DO NOT print out this 250+ page document.

27

Steps to Do Your Homework

- **Download the MIPS plyagound tarball (http://www.mscs.mu.edu/~rge/cosc2200/homework-fall2011/xinu-cosc2200.tgz) and save it to your directory**
- **Untar**
  - tar gzxvf xinu-cosc2200.tgz
- **Rename or copy directory to distinct name using mv or cp command**
- **Edit main.S**
- **Assemble the code using "make" command**
- **Upload the code to MIPS machine using command**
  ./mipcon
- **Ctrl-Space, and then "q" to quit from playground**

To submit your three files, use

turnin –c cosc2200 –p HW6 main-q1.S main-q2.S main-q3.S

Don't submit them individually..

28

Things To Pay Attention In Programming

- **The list of authors**
- **Formatting (spacing, indentation)**
- **Commenting**
- **Style**
  - Write clearly – don't sacrifice clarity for efficiency, KISS
  - Say what you mean, simply and directly
  - Be sparing with temporary variables – registers are limited
  - Use good structure for your code
  - Use subroutines
  - Don't batch bad code – rewrite it
- **Write and test a big program in small pieces**

29

Review for HexDump

1. **Follow instructions**
   1. Exact filename
   2. Only submit your source code
   3. How should your code execute: the number of command line arguments, the meaning of arguments
2. **Check the number of command line arguments yourself using**

```
if(args.length != 1){
    print usage
    exit
}
```

3. **Specifics. A byte is read. This byte might be one of the following**
   1. Its most significant bit is 0
   2. Its most significant bit is 1

   When you use method intValue(),
   1. Case 1 gives you a positive integer that is in the range of ASCII characters
   2. Case 2 gives you a negative integer that is outside the range of ASCII characters

   For each case, how do you print the HEX code and the character?

30

- **So far, we have learnt**
  - R-type and I-type arithmetic instructions

    add addi

    sub

    mul

    div
  - Some conditional and unconditional jumps for decisions
    - bne, beq
    - b, jal
    - loops
    - Conditional jumps
- **What we are going to learn**
  - Memory-register transfers

31

Assembly Operands: Memory

- **Scalar variables map onto registers; what about large data structures like arrays?**
- **Memory contains such data structures**
- **But MIPS arithmetic instructions only operate on registers, never directly on memory.**
- **Data transfer instructions transfer data between registers and memory:**
  - Memory to register
  - Register to memory

32

Data Transfer: Memory to Reg (1/4)

- **To transfer a word of data, we need to specify two things:**
  - Register: specify this by number (0 - 31)
  - Memory address: more difficult
    - Think of memory as a single one-dimensional array, so we can address it simply by supplying a pointer to a memory address.
    - Other times, we want to be able to offset from this pointer.

33