Decision making, SPIM intro

CSE 410 - Computer Systems
October 5, 2001

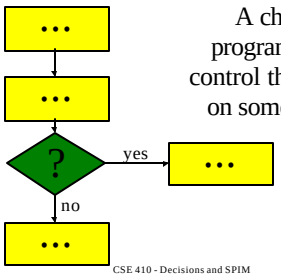
Readings and References

- Reading
  - P&H: Sections 3.5, A.9, A.10 through page A-54

<rant>goto considered harmful</rant>

- "Oh what a tangled web we weave, When first we practice to deceive!"
  - *Sir Walter Scott*
- Branching in assembly language can turn your program into a rat's nest that cannot be debugged
- Keep control flow simple and logical
- Use comments describing the overall logic

Conditional Branch



A change in the program's flow of control that depends on some condition

Branch instructions

- Branch instructions are I-format instructions
  - op code field
  - two register fields
  - 16-bit offset field
- Simplest branches check for equality
  - `beq $t0, $t1, address`
  - `bne $t0, $t1, address`

Go to where?

- Calculating the destination address
  - 4*(the 16-bit offset value)
  - is added to the Program Counter (PC)
- The offset is a word offset in this case
- The base register is always the PC, so we don't need to specify it in the instruction
- Covers a range of 2^{16} words (64 KW)

if (i==j) then a=b;

- Assume all values are in registers
- Note that the test is inverted!

```
# $t0=i, $t1=j, $s0=a, $s1=b

     bne  $t0, $t1, skip
     move $s0, $s1
skip:
```


while (s[i]==k) i = i+j;

```
# $s0=addr(s), $v1=i, $a0=k, $a1=j

loop:
    sll     $v0,$v1,2    # v0 = 4*i
    addu    $v0,$s0,$v0  # v0 = addr(s[i])
    lw      $v0,0($v0)   # v0 = s[i]
    addu    $v1,$v1,$a1  # i = i+j
    beq     $v0,$a0,loop # loop if equal
    subu    $v1,$v1,$a1  # i = i-j
```


for (i=0; i<10; i++) s[i] = i;

```
# $s0=addr(s), $t1=i
move      $t1,$zero      # i = 0
loop:
  sll     $t0,$t1,2      # t0 = i*4
  addu    $t0,$s0,$t0    # t0 = addr(s[i])
  sw      $t1,0($t0)     # s[i] = i
  addu    $t1,$t1,1      # i++
  slt     $t0,$t1,10     # if (i<10) $t0=1
  bnez    $t0,loop       # loop if (i<10)
```


Comparison instructions

- For comparisons other than equality
  - **slt** : set less than
  - **sltu** : set less than unsigned
  - **slti** : set less than constant value
  - **sltiu** : set less than unsigned constant
- set t0 to 1 if t1<t2

```
slt $t0, $t1, $t2
```


Pseudo-instructions

- The assembler is your friend and will build instruction sequences for you
- Original code:

```
bge   $a0,$t1,end     # if a0>=t1 skip
```

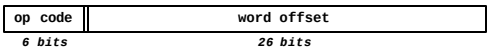
- Actual instructions:

```
slt   $at,$a0,$t1     # if a0<t1 at=true
beq   $at,$0,end      # skip if at==false
```


Jump Instructions

- Jump instructions provide longer range than branch instructions
- 26-bit word offset in J-format instructions
  - j : jump
  - jal : jump and link (store return address)
- 32-bit address in register jumps
  - jr : jump through register
  - jalr : jump through register and link

J-format fields



- The word offset value is multiplied by 4 to create a byte offset
  - the result is 28 bits wide
- Then concatenated with top 4 bits of PC to make a 32 bit destination address

Important Jumps

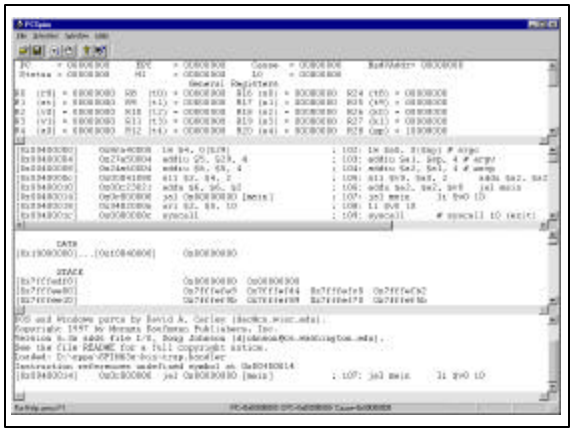
- Jump and link (**`jal`**)
  - call procedure and store return address in $ra
- Jump through register (**`jr`**)
  - return to caller using the address in $ra
- We will talk about procedure calls in excruciating detail next lecture

SPIM simulator

- SPIM lets you write MIPS assembly language code and run it on a PC
- We will use an extended version of PCSpim
  - 6.3a extensions add file reading and writing
- PCSpim is installed on the machines in the Math Sciences Computing Center
- You can download it from the web site

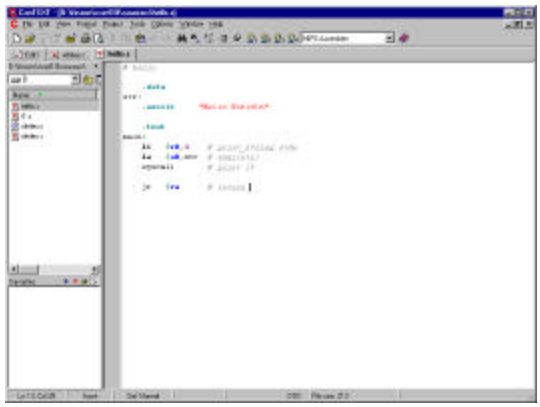
Spim display

- Register panel
  - register names and numbers
- Text segment panel
  - note jump and link to “main” at [0x00400014]
  - your code defines the label “main”
- Data and Stack segment panel
- Message panel



Context editor

- You can use any text editor you like to write the source code
- Context editor provided in MSCC
  - it has a highlighter for MIPS assembly language
  - it doesn’t try to be a word processor



hello.s

```
   .data
str:
   .asciiz     "Hello World\n"

   .text
main:
   li    $v0,4     # print_string code
   la    $a0,str   # addr(str)
   syscall         # print it

   jr    $ra       # return
```