Decision making, SPIM intro

CSE 410 - Computer Systems

October 5, 2001

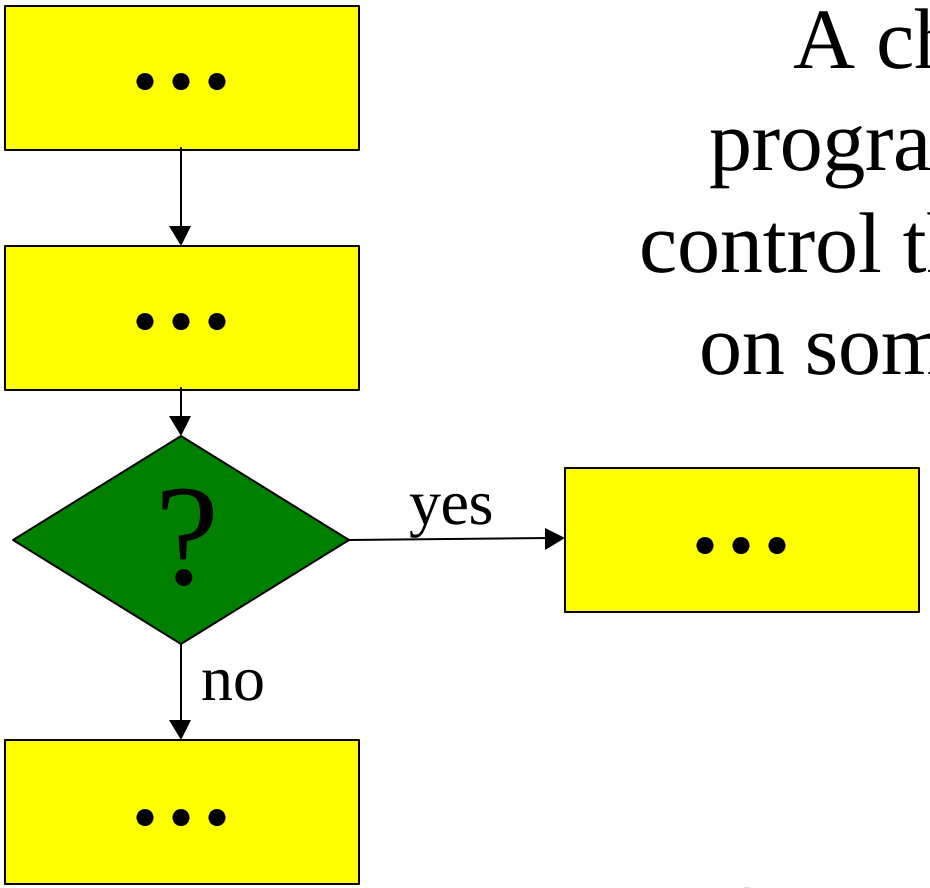
Readings and References

- 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
```

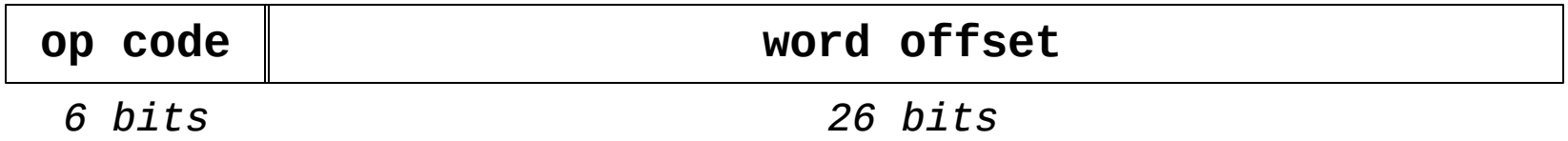
- Actual instructions:

```
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

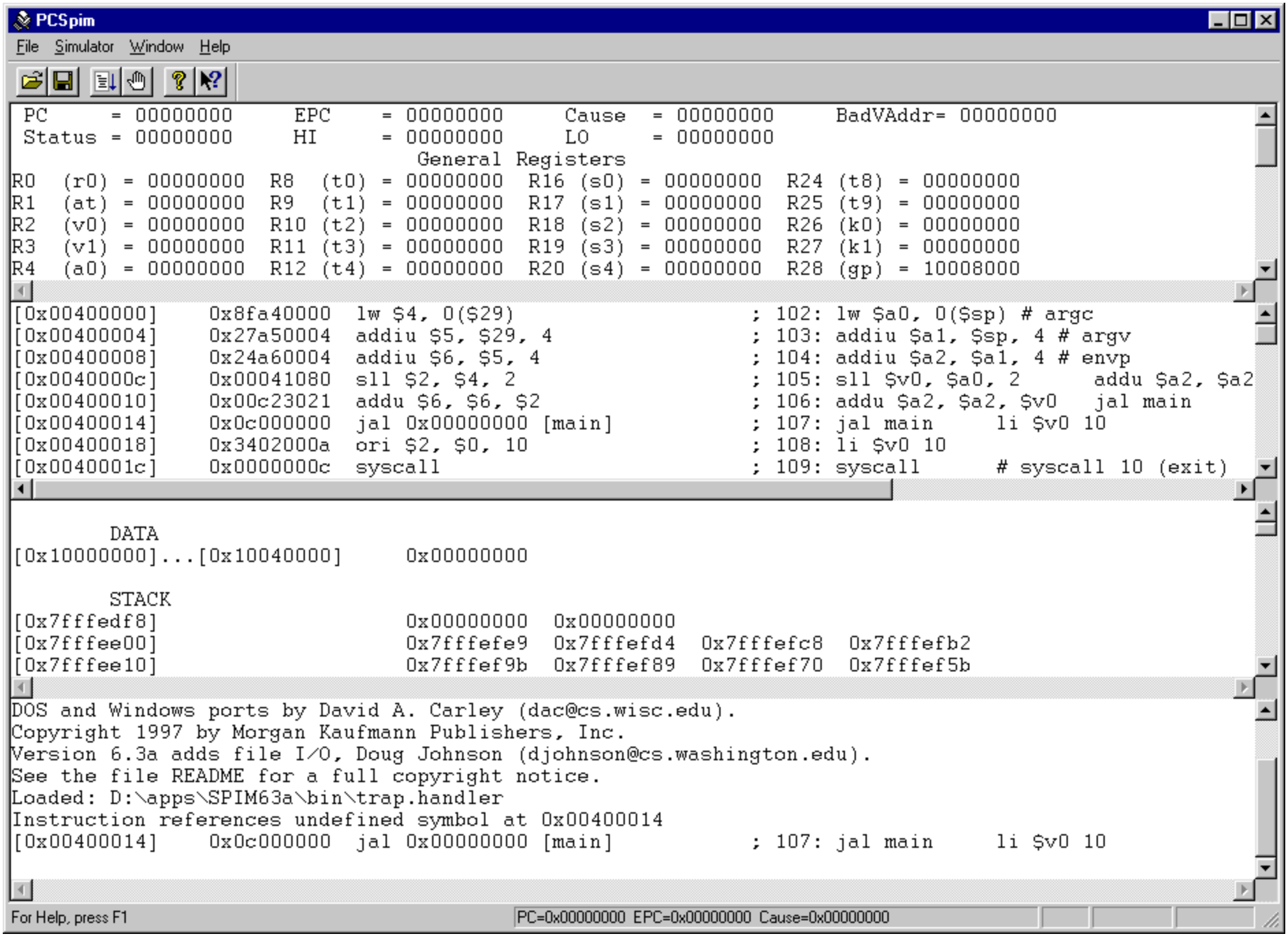
- Jump and link (**`jal`**)
  - call procedure and store return address in $ra
- Jump through register (**`jr`**)
  - return to caller using the address in $ra
- We will talk about procedure calls in excruciating detail next lecture

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

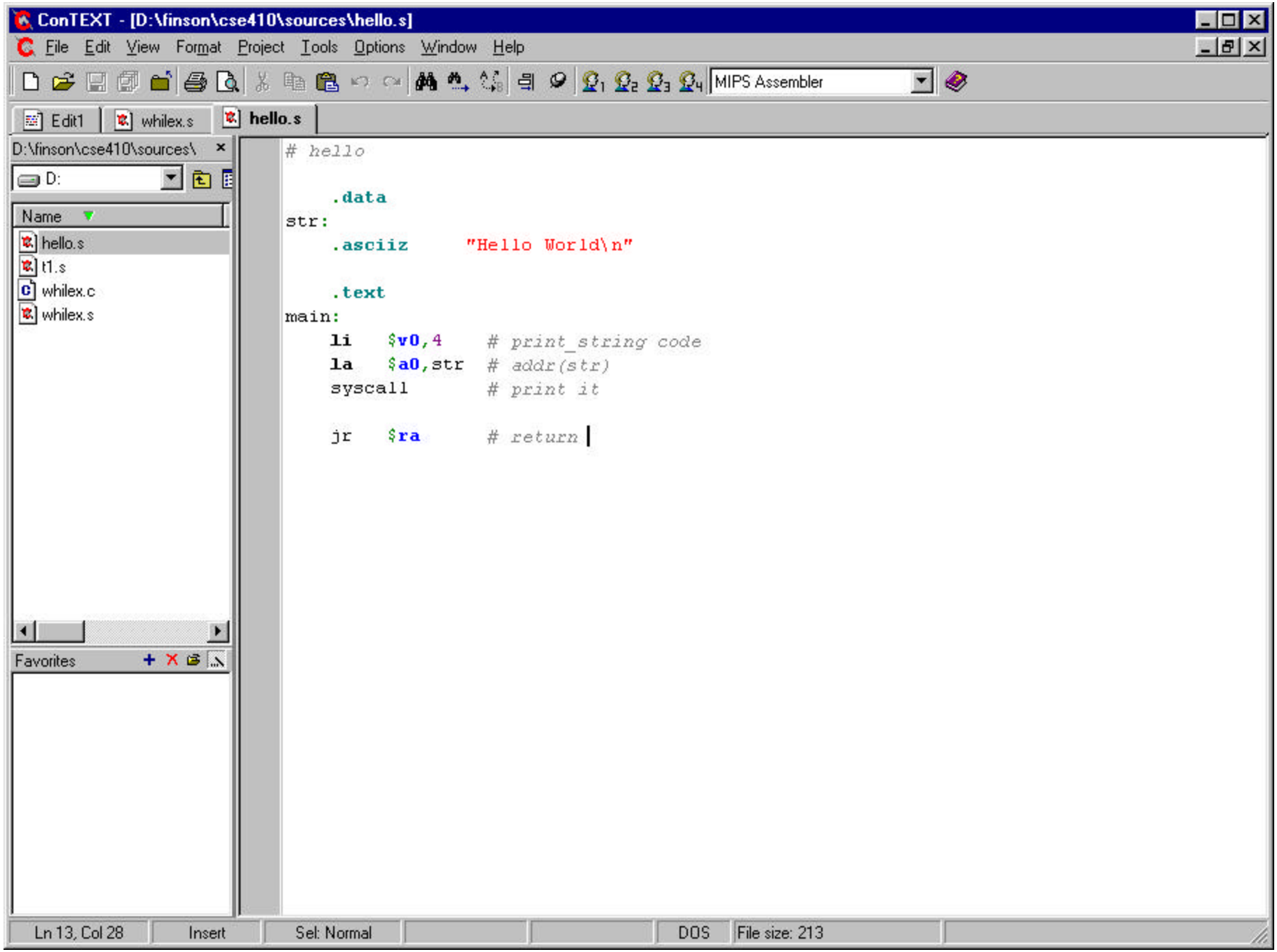
Spim display

- 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


PCSpim
File Simulator Window Help
PC = 00000000 EPC = 00000000 Cause = 00000000 BadVAddr= 00000000
Status = 00000000 HI = 00000000 LO = 00000000
General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 102: lw $a0, 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 103: addiu $a1, $sp, 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 104: addiu $a2, $a1, 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 105: sll $v0, $a0, 2 addu $a2, $a2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 106: addu $a2, $a2, $v0 jal main
[0x00400014] 0x0c000000 jal 0x00000000 [main] ; 107: jal main li $v0 10
[0x00400018] 0x3402000a ori $2, $0, 10 ; 108: li $v0 10
[0x0040001c] 0x0000000c syscall ; 109: syscall # syscall 10 (exit)
DATA
[0x10000000]...[0x10040000] 0x00000000
STACK
[0x7fffedf8] 0x00000000 0x00000000
[0x7fffee00] 0x7fffefe9 0x7fffefd4 0x7fffefc8 0x7fffefb2
[0x7fffee10] 0x7fffef9b 0x7fffef89 0x7fffef70 0x7fffef5b
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
Version 6.3a adds file I/O, Doug Johnson (djohnson@cs.washington.edu).
See the file README for a full copyright notice.
Loaded: D:\apps\SPIM63a\bin\trap.handler
Instruction references undefined symbol at 0x00400014
[0x00400014] 0x0c000000 jal 0x00000000 [main] ; 107: jal main li $v0 10
For Help, press F1
PC=0x00000000 EPC=0x00000000 Cause=0x00000000

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


ConTEXT - [D:\finson\cse410\sources\hello.s]
File Edit View Format Project Tools Options Window Help
MIPS Assembler
Edit1
whilex.s
hello.s
D:\finson\cse410\sources\
D:
Name
hello.s
t1.s
whilex.c
whilex.s
Favorites
hello
.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
Ln 13, Col 28
Insert
Sel: Normal
DOS
File size: 213

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
```