SPIM Tutorial

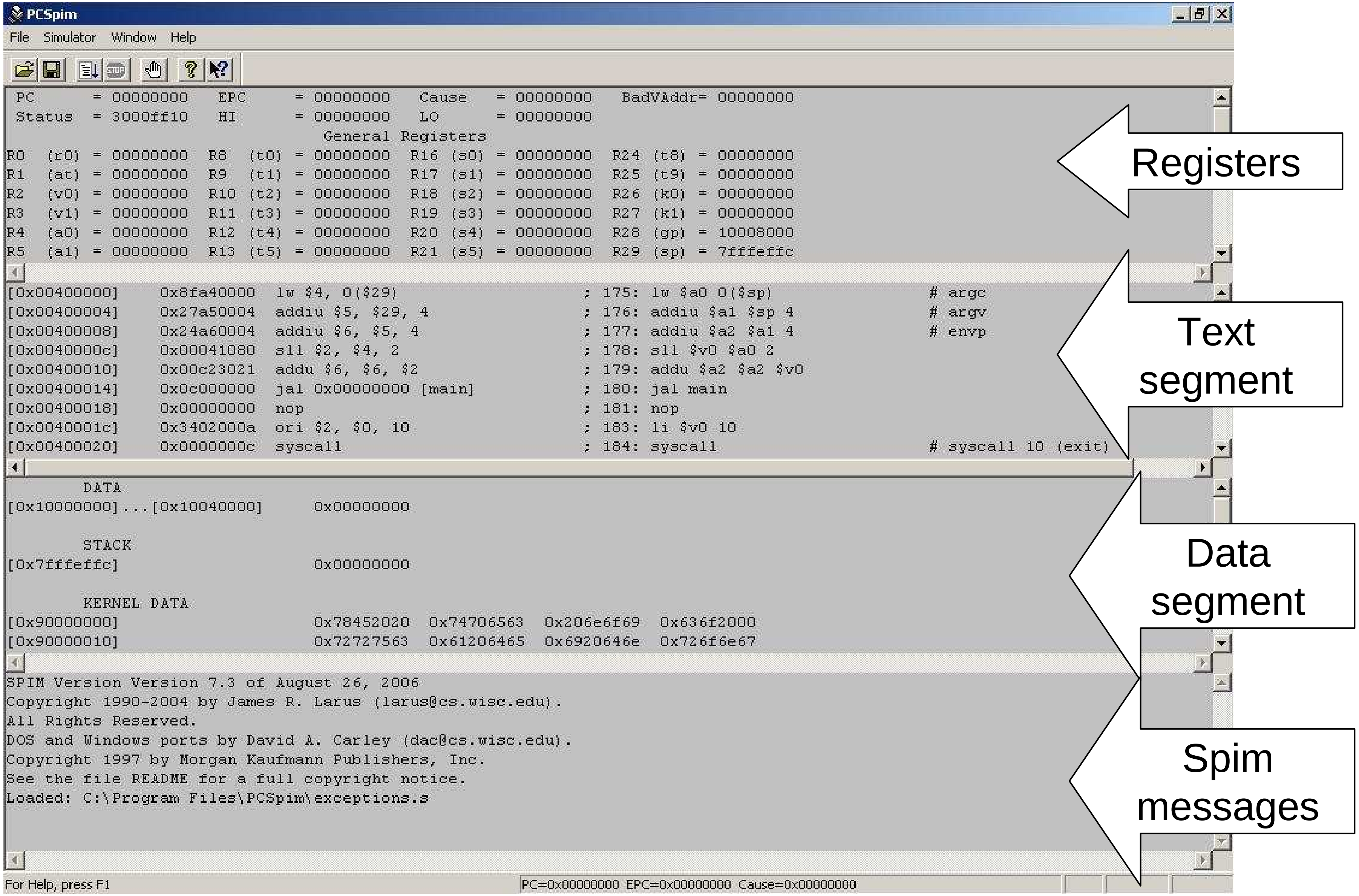
CSE 410 Computer Systems

Introduction

- SPIM: MIPS simulator
  - Reads/executes assembly source programs
- Does not execute binaries

- Download
  - http://www.cs.wisc.edu/~larus/spim.html
  - Windows: PCSpim
  - Linux: xspim
- Read
  - Hennessy & Patterson, Appendix A
  - Resources at SPIM web site

Environment



Registers

Number	Mnemonic	Usage	Number	Mnemonic	Usage
\$0	zero	Permanently 0	\$24, \$25	\$t8, \$t9	Temporary
\$1	\$at	Assembler Temporary	\$26, \$27	\$k0, \$k1	Kernel
\$2, \$3	\$v0, \$v1	Value returned by a subroutine	\$28	\$gp	Global Pointer
\$4-\$7	\$a0-\$a3	Subroutine Arguments	\$29	\$sp	Stack Pointer
\$8-\$15	\$t0-\$t7	Temporary	\$30	\$fp	Frame Pointer
\$16-\$23	\$s0-\$s7	Saved registers	\$31	\$ra	Return Address

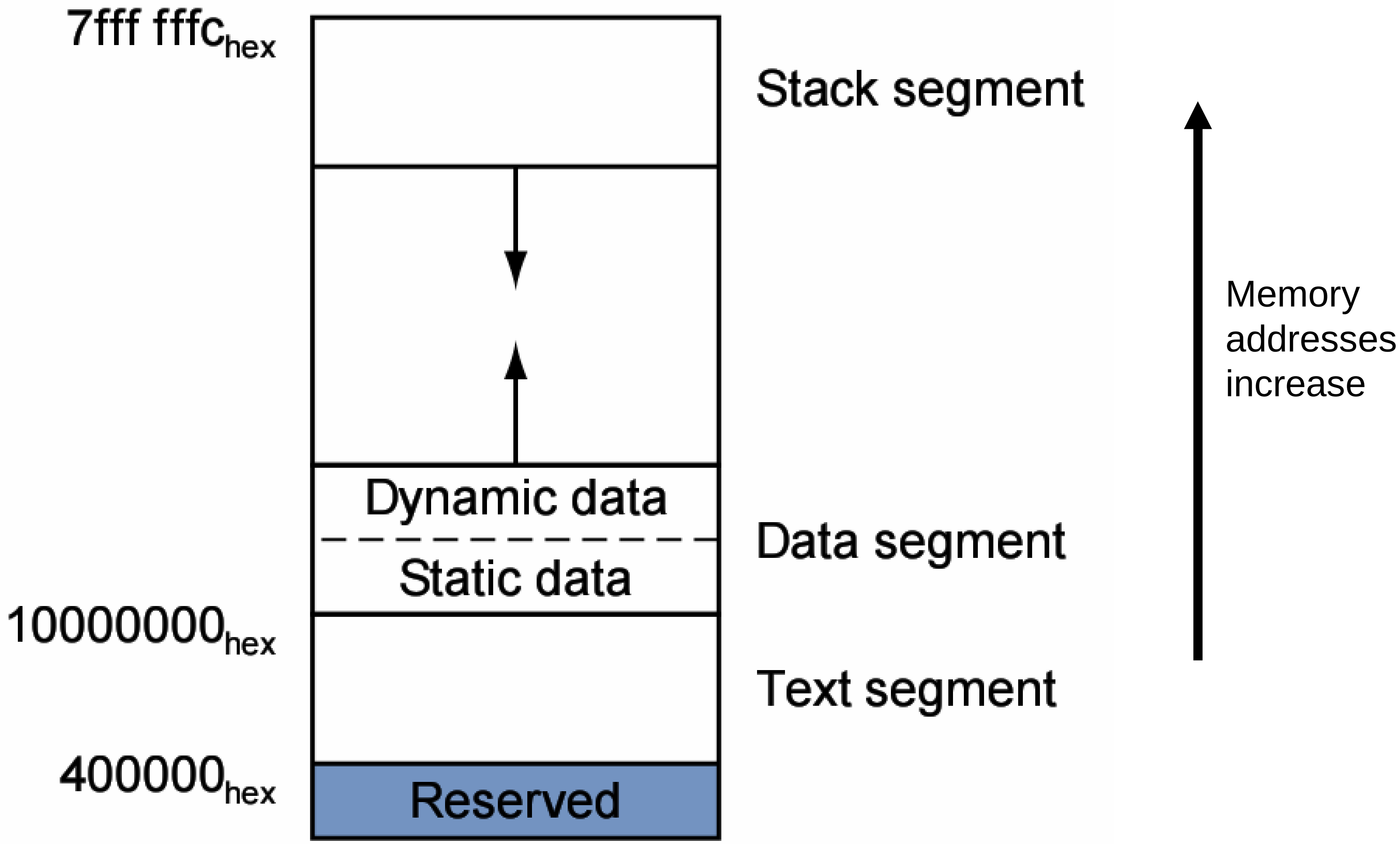
Let’s try

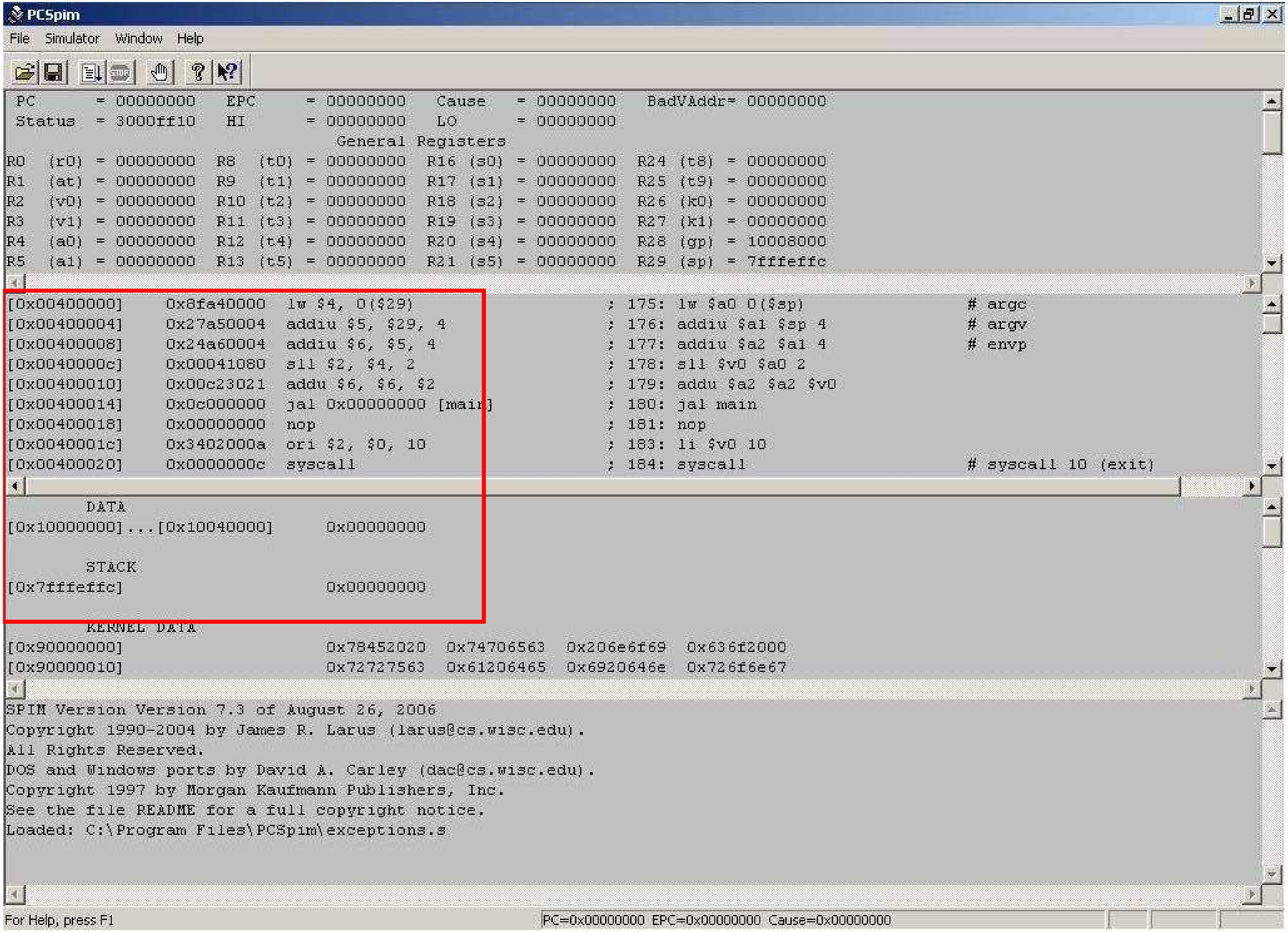
```
        .text
        .globl   main
main:
        li       $t0, 0x2            #  $t0 ← 0x2
        li       $t1, 0x3            #  $t1 ← 0x3
        addu     $t2, $t0, $t1       #  $t2 ← ADD($t0, $t1)
```

Let’s try

```
        .text
        .globl   main
main:
        ori      $t0, $0, 0x2        #  $t0 ← OR(0, 0x2)
        ori      $t1, $0, 0x3        #  $t1 ← OR(0, 0x3)
        addu     $t2, $t0, $t1       #  $t2 ← ADD($t0, $t1)
```

Memory layout




PCSpim
File Simulator Window Help
PC = 00000000 EPC = 00000000 Cause = 00000000 BadVAddr= 00000000
Status = 3000ff10 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
R5 (a1) = 00000000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7fffeffc
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 175: lw $a0 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 176: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 177: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 178: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 179: addu $a2 $a2 $v0
[0x00400014] 0x0c000000 jal 0x00000000 [main] ; 180: jal main
[0x00400018] 0x00000000 nop ; 181: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 183: li $v0 10
[0x00400020] 0x0000000c syscall ; 184: syscall # syscall 10 (exit)
DATA
[0x10000000]...[0x10040000] 0x00000000
STACK
[0x7fffeffc] 0x00000000
KERNEL DATA
[0x90000000] 0x78452020 0x74706563 0x206e6f69 0x636f2000
[0x90000010] 0x72727563 0x61206465 0x6920646e 0x726f6e67
SPIM Version Version 7.3 of August 26, 2006
Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Loaded: C:\Program Files\PCSpim\exceptions.s
For Help, press F1
PC=0x00000000 EPC=0x00000000 Cause=0x00000000

How to use memory

1. LOAD from memory to register
   - lw, lb, ld, ... (lw $t0, address)
2. COMPUTE in registers
   - add, ori, beq, jal,... (add $t2, $t0, $t1)
3. STORE from register to memory
   - sw, sb, sd,... (sw $t2, address)

Addressing modes

Format	**Address Computation**
(register)	contents of register
imm	immediate
imm (register)	immediate + contents of register
symbol	address of symbol
symbol ± imm	address of symbol + or − immediate
symbol ± imm (register)	address of symbol + or − (immediate + contents of register)

Addressing modes

Loading from memory to $t0: lw $t0, address?

Imm+Register: <u>(Only mode in bare machine)</u>

```
la   $t1, label          # load address of label to $t1
lw  $t0, 2($t1)          # address: address of label + 2
```

Immediate:

```
lw  $t0, 0x000AE430   # address: address 0x000AE430
```

Symbol:

```
lw  $t0, label            # address: address of label
```

Register:

```
la   $t1, label          # load address of label to $t1
lw  $t0, $t1             # address: address in $t1
```

Symbol±Imm:

```
lw  $t0, label+2         # address: address of label + 2
```

Symbol±Imm+Register:

```
lw  $t0, label+2($t1)  # address: address of label + 2 + $t1
```

```
        .data
n:      .word   0x2
m:      .word   0x3
r:      .space  4

        .text
        .globl  main
main:
        lw      $t0, n              # load n to $t0
        lw      $t1, m              # load m to $t1

        addu    $t2, $t0, $t1       #  $t2 ← ADD($t0, $t1)

        sw      $t2, r              # store $t2 to r
```

```
        .data
n:      .word   0x2
m:      .word   0x3
r:      .space  4

        .text
        .globl  main
main:
        la      $t5, n              # load address of n to $t5
        lw      $t0, 0($t5)         # load n to $t0

        la      $t5, m              # load address of m to $t5
        lw      $t1, 0($t5)         # load m to $t1

        addu    $t2, $t0, $t1       #  $t2 ← ADD($t0, $t1)

        la      $t5, r              # load address of r to $t5
        sw      $t2, 0($t5)         # store $t2 to r
```

```
        .data
n:      .word   0x2, 0x3, 0x4

        .text
        .globl  main
main:
        la      $t5, n              # load address of n to $t5

        lw      $t0, 0($t5)         # load n to $t0

        lw      $t1, 4($t5)         # load n+4 to $t1

        addu    $t2, $t0, $t1       #  $t2 ← ADD($t0, $t1)

        sw      $t2, 8($t5)         # store $t2 to n+8
```

System calls

Service	System Call Code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

System calls – print_str

```
        .data
str:    .asciiz "Hello World"     # H,e,l,l,o, ,W,o,r,l,d,\0

        .text
        .globl   main
main:
        li   $v0,  4          # code for print_str
        la  $a0,  str         # argument
        syscall               # executes print_str
```

System calls – read _int

```
        .data
num:    .space 4

        .text
        .globl   main
main:
        li  $v0,  5             # code for read_int
        syscall                 # executes read_int
                                # return value is stored in $v0
        la   $t0, num           # load address of num to $t0
        sw  $v0, 0($t0)         # sw $v0, num
```

Branching

```
x  ← read_int
y  ← read_int

if x == y
   then print “Equal”
   else print “Not equal”
```

Branching

```
        .text
        .globl   main
main:
        li   $v0,  5
        syscall
        move  $t0, $v0

        li   $v0,  5
        syscall
        move  $t1, $v0

        bne  $t0, $t1, printNe
```

```
printEq:
        la  $a0,  strEq
        j  print

printNe:
        la  $a0, strNe
        j  print

print:
        li   $v0,  4
        syscall

.data
strEq:   .asciiz "Equal"
strNe:   .asciiz "Not equal"
```

Branching

```
        .text
        .globl   main
main:
        li   $v0,  5
        syscall
        move  $t0, $v0

        li   $v0,  5
        syscall
        move  $t1, $v0

        seq $t2, $t0, $t1

        beq  $t2, $0, printNe
```

```
printEq:
        la  $a0,  strEq
        j  print

printNe:
        la  $a0, strNe
        j  print

print:
        li   $v0,  4
        syscall

.data
strEq:   .asciiz “Equal”
strNe:   .asciiz “Not equal”
```

Looping

```
n ← read_int

counter ← 0
total   ← 0

do
    counter ← counter + 1
    total   ← total + counter
until counter == n

print total
```

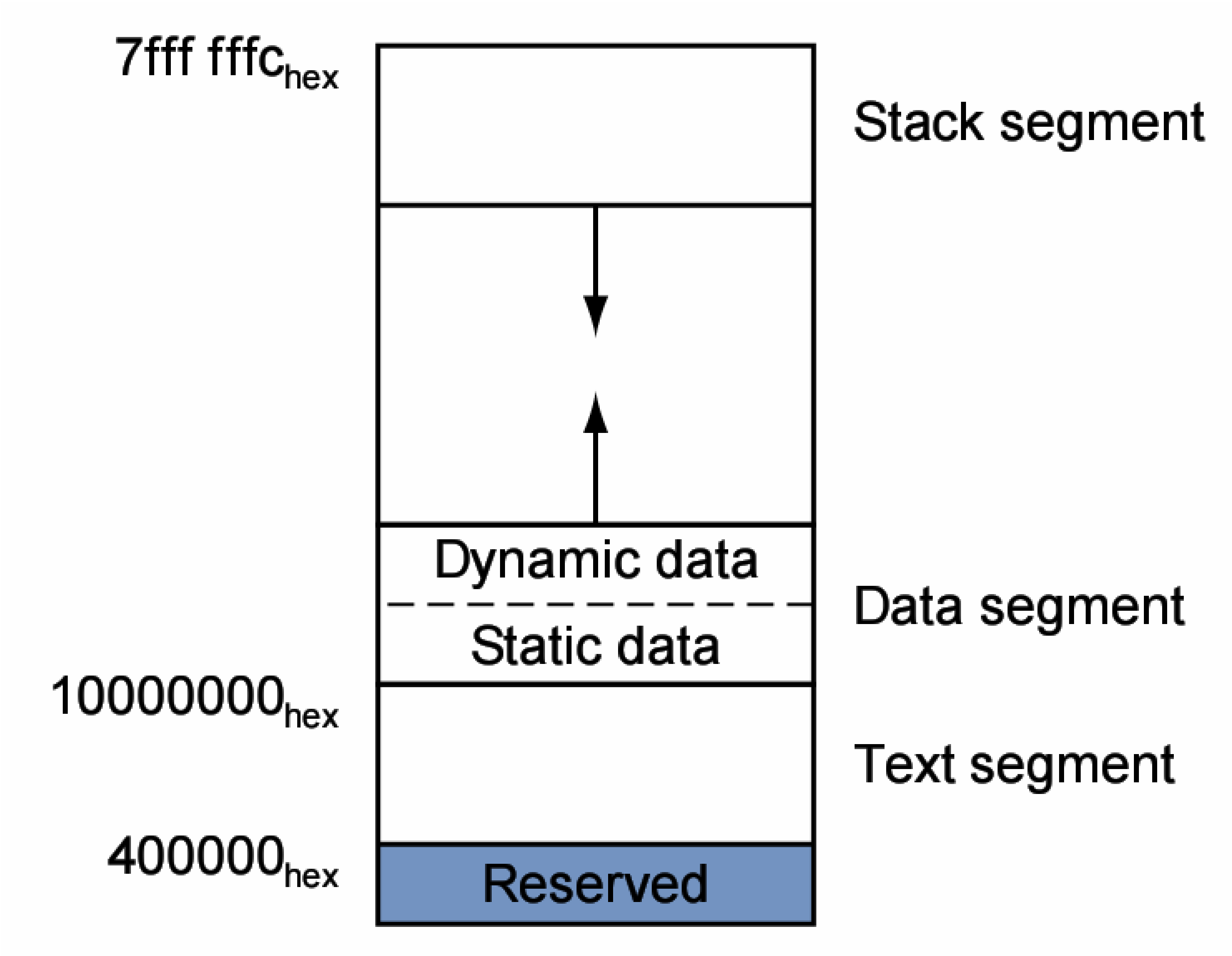
Looping

```
    .text
    .globl main
main:
    li   $v0,  5
    syscall

    move  $t0, $v0

    # $t0 is the original value

    li      $t1, 0   # counter
    li      $t2, 0   # sum

loop:
    addi $t1, $t1, 1
    add  $t2, $t2, $t1

    # counter = original value ?
    beq  $t0, $t1, done
    j loop

done:
    li   $v0,  1    # print_int
    move $a0, $t2
    syscall
```

Looping

```
    .text
    .globl main
main:
    li   $v0,  5
    syscall

    move  $t0, $v0

    # $t0 is the original value

    li      $t1, 0   # counter
    li      $t2, 0   # sum

loop:
    addi $t1, $t1, 1
    add  $t2, $t2, $t1

    # counter = original value ?
    bne  $t0, $t1, loop

done:
    li   $v0,  1    # print_int
    move $a0, $t2
    syscall
```

Functions

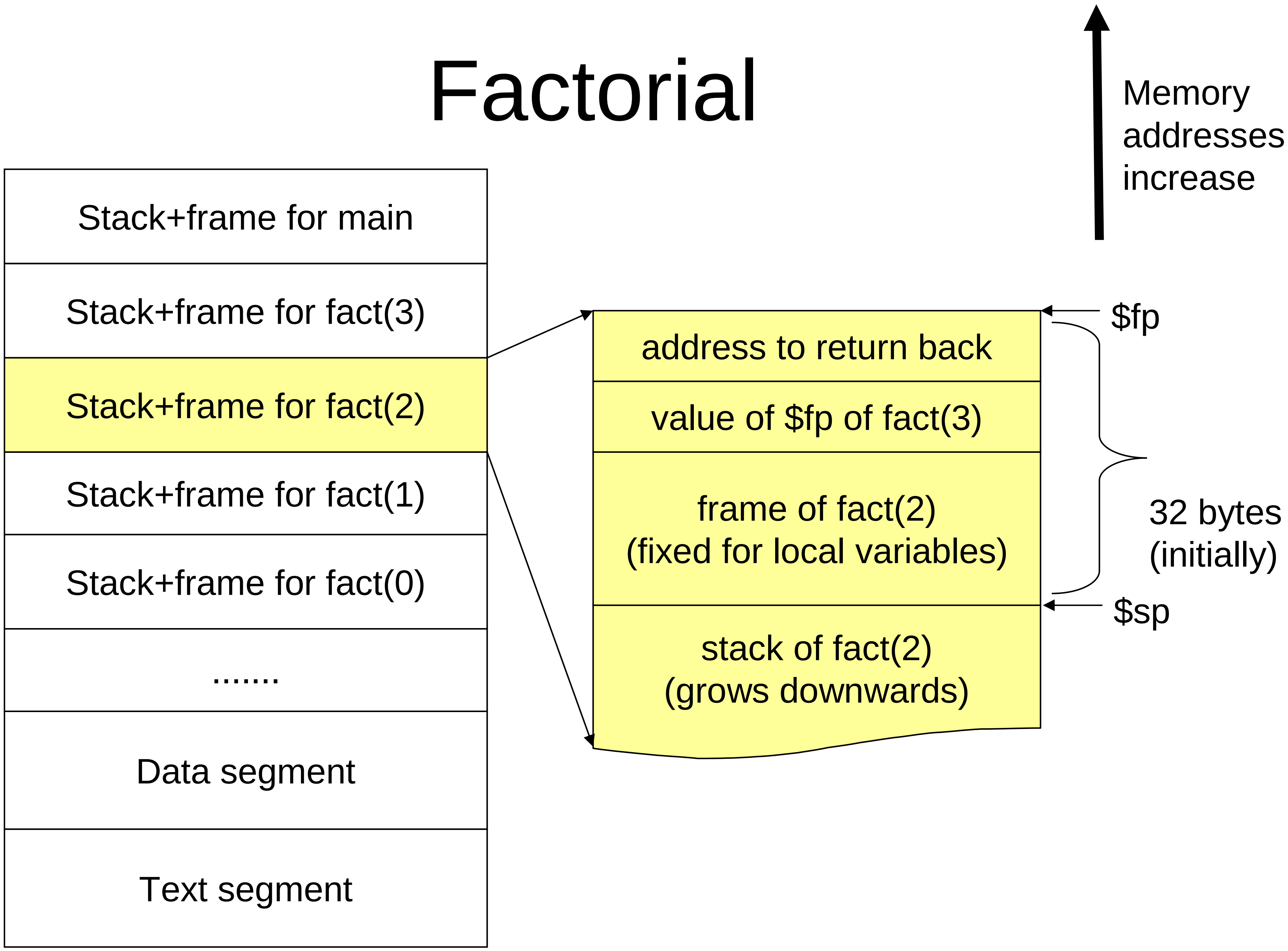


7fff fffc hex
Stack segment
Dynamic data
Data segment
Static data
10000000 hex
Text segment
400000 hex
Reserved

Factorial

```
main() {
    x = fact(5);
    ....
    y = fact(6);
}

fact(int n) {
    if n == 0 or n == 1
          then return 1;
          else return n * fact(n-1);
}
```

Factorial



Memory addresses increase
Stack+frame for main
Stack+frame for fact(3)
Stack+frame for fact(2)
Stack+frame for fact(1)
Stack+frame for fact(0)
....
Data segment
Text segment
$fp
address to return back
value of $fp of fact(3)
frame of fact(2)
(fixed for local variables)
32 bytes
(initially)
$sp
stack of fact(2)
(grows downwards)

Now it’s your turn

- Write factorial program <u>without functions</u>
- Use branches and loops only