Assemblers, Linkers, and Loaders

Hakim Weatherspoon

CS 3410, Spring 2013

Computer Science

Cornell University

See: P&H Appendix B.3-4 and 2.12

Goal for Today: Putting it all Together

Review Calling Convention

Compiler output is assembly files

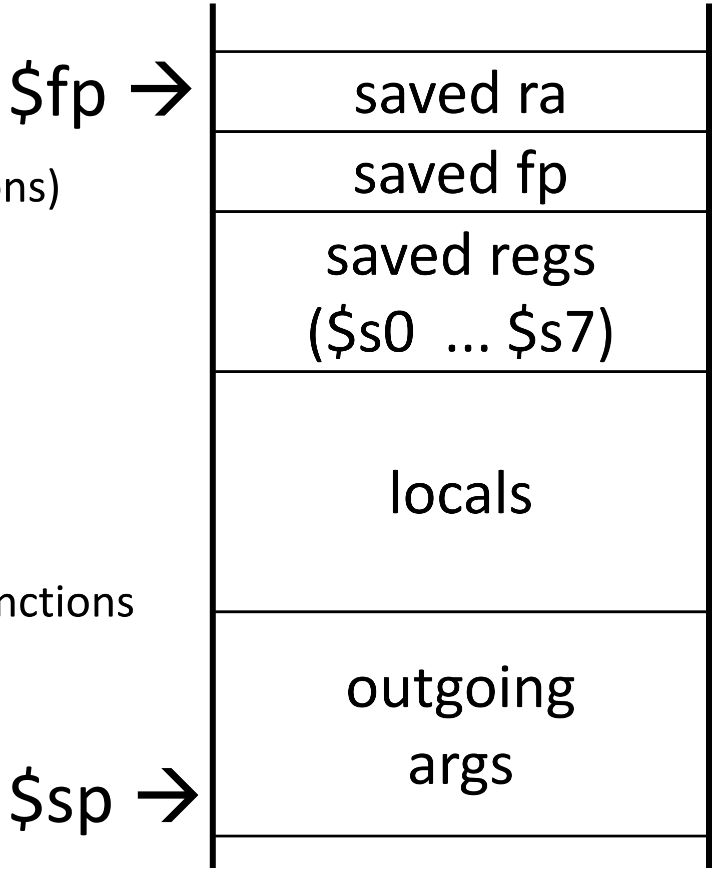
Assembler output is obj files

Linker joins object files into one executable

Loader brings it into memory and starts execution

Recap: Calling Conventions

- first four arg words passed in $a0, $a1, $a2, $a3
- remaining arg words passed in parent’s stack frame
- return value (if any) in $v0, $v1
- stack frame at $sp
  - contains $ra (clobbered on JAL to sub-functions)
  - contains $fp
  - contains local vars (possibly clobbered by sub-functions)
  - contains extra arguments to sub-functions (i.e. argument “spilling)
  - contains space for first 4 arguments to sub-functions
- callee save regs are preserved
- caller save regs are not
- Global data accessed via $gp



$fp →

saved ra

saved fp

saved regs
($s0 ... $s7)

locals

outgoing
args

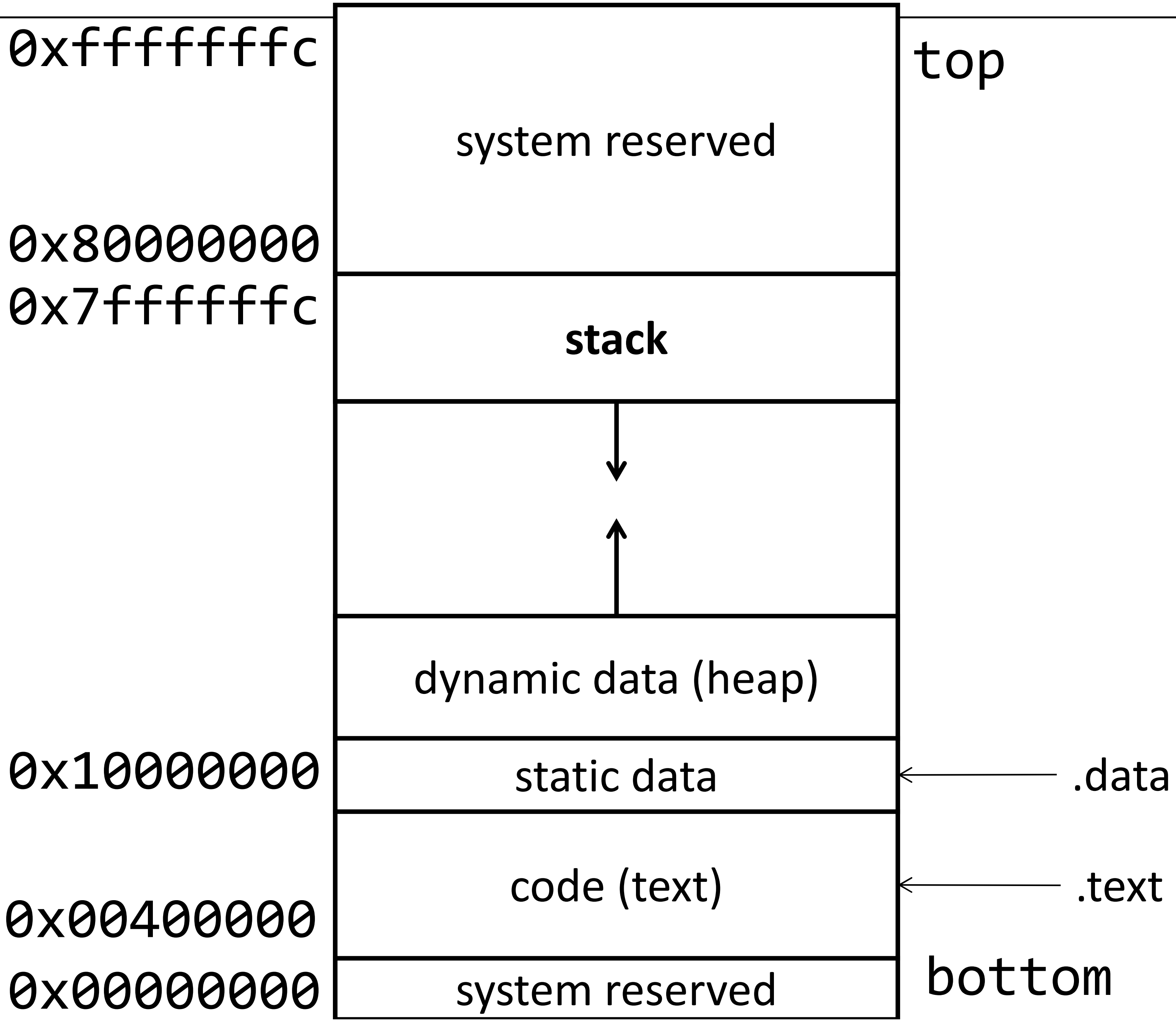
$sp →

Warning: There is no one true MIPS calling convention.
lecture != book != gcc != spim != web

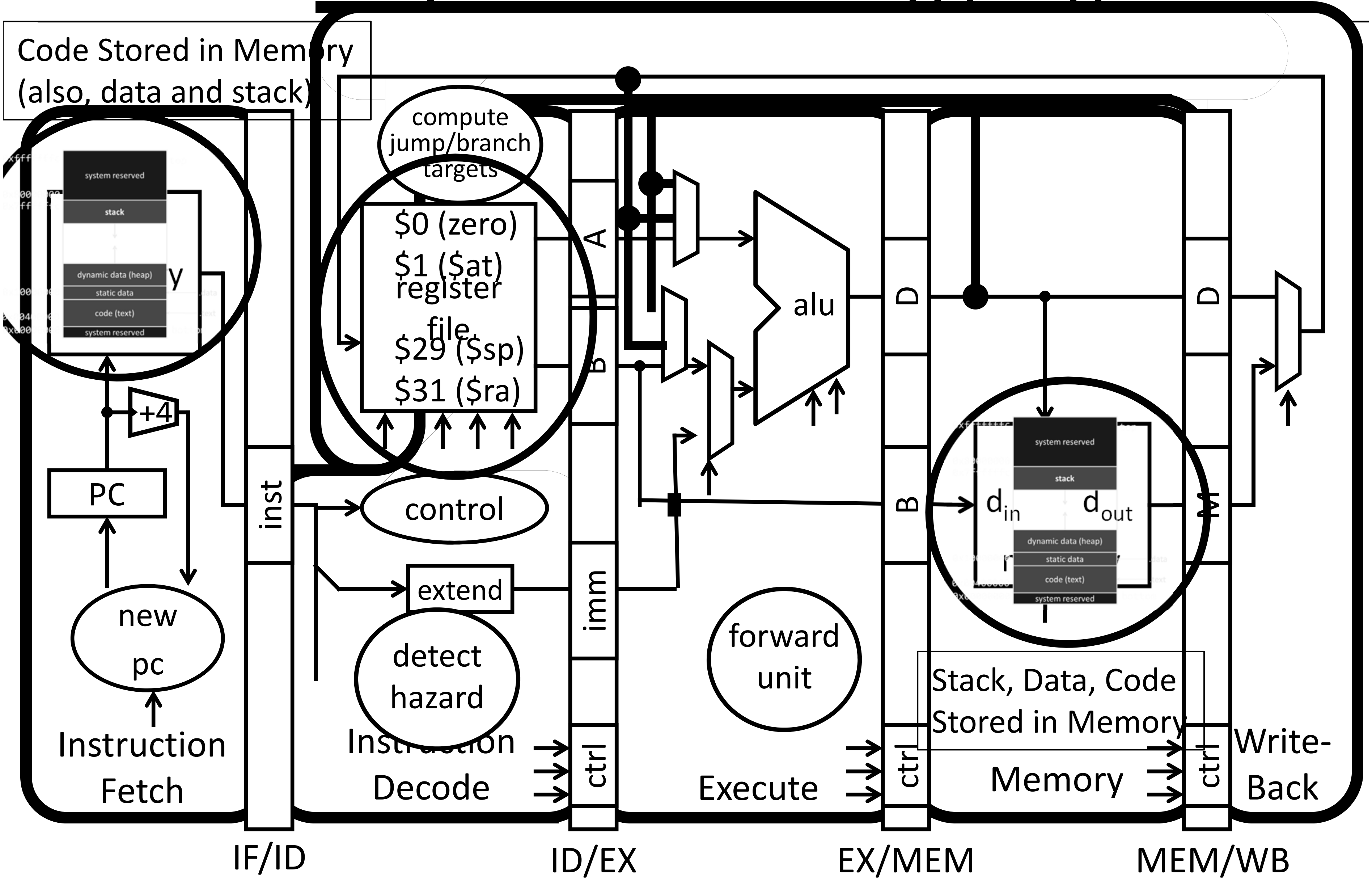
MIPS Register Conventions

r0	$zero	zero	r16	$s0	**saved (callee save)**
r1	$at	assembler temp	r17	$s1	
r2	$v0	function return values	r18	$s2	
r3	$v1		r19	$s3	
r4	$a0	function arguments	r20	$s4	
r5	$a1		r21	$s5	
r6	$a2		r22	$s6	
r7	$a3		r23	$s7	
r8	$t0	**temps (caller save)**	r24	$t8	**more temps (caller save)**
r9	$t1		r25	$t9	
r10	$t2		r26	$k0	reserved for kernel
r11	$t3		r27	$k1	
r12	$t4		r28	$gp	global data pointer
r13	$t5		r29	$sp	stack pointer
r14	$t6		r30	$fp	frame pointer
r15	$t7		r31	$ra	return address

Anatomy of an executing program



Anatomy of an executing program


Code Stored in Memory (also, data and stack)

Stack, Data, Code Stored in Memory

Takeaway

We need a calling convention to coordinate use of registers and memory. Registers exist in the Register File. Stack, Code, and Data exist in memory. Both instruction memory and data memory accessed through cache (modified harvard architecture) and a shared bus to memory (Von Neumann).

Next Goal

Given a running program (a process), how do we know what is going on (what function is executing, what arguments were passed to where, where is the stack and current stack frame, where is the code and data, etc)?

Activity #1: Debugging

init():	0x400000
printf(s, …):	0x4002B4
vnorm(a,b):	0x40107C
main(a,b):	0x4010A0
pi:	0x10000000
str1:	0x10000004

CPU:
$pc=0x004003C0
$sp=0x7FFFFFAC
$ra=0x00401090

	0x00000000
	0x0040010c
	0x7FFFFFF4
	0x00000000
	0x00000000
	0x00000000
	0x00000000
	0x004010c4
	0x7FFFFFDC
	0x00000000
	0x00000000
	0x00000015
0x7FFFFFB0	0x10000004
	0x00401090

What func is running?

Who called it?

Has it called anything?

Will it?

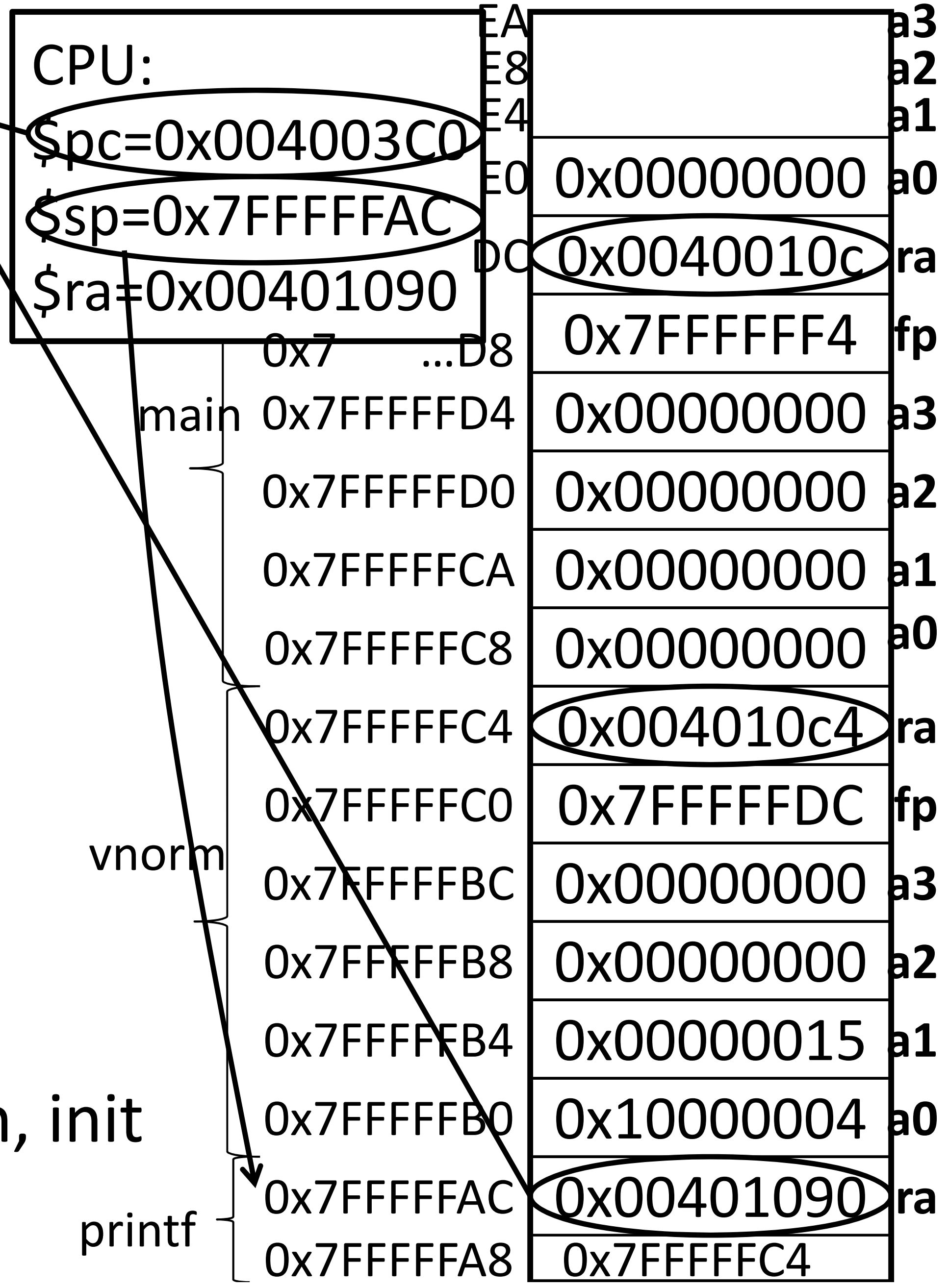
Args?

Stack depth?

Call trace?

Activity #1: Debugging

init():	0x400000
printf(s, …):	0x4002B4
vnorm(a,b):	0x40107C
main(a,b):	0x4010A0
pi:	0x10000000
str1:	0x10000004



CPU:

$pc=0x004003C0

$sp=0x7FFFFFAC

$ra=0x00401090

Memory

…F4		ra
…F0		fp
EA		a3
E8		a2
E4		a1
E0	0x00000000	a0
DC	0x0040010c	ra
0x7 …D8	0x7FFFFFF4	fp
0x7FFFFFD4	0x00000000	a3
0x7FFFFFD0	0x00000000	a2
0x7FFFFFCA	0x00000000	a1
0x7FFFFFC8	0x00000000	a0
0x7FFFFFC4	0x004010c4	ra
0x7FFFFFC0	0x7FFFFFDC	fp
0x7FFFFFBC	0x00000000	a3
0x7FFFFFB8	0x00000000	a2
0x7FFFFFB4	0x00000015	a1
0x7FFFFFB0	0x10000004	a0
0x7FFFFFAC	0x00401090	ra
0x7FFFFFA8	0x7FFFFFC4	

What func is running? printf

Who called it? vnorm

Has it called anything? no

Will it? no b/c no space for outgoing args

Args? Str1 and 0x15

Stack depth? 4

Call trace? printf, vnorm, main, init

Compilers and Assemblers

Next Goal

How do we compile a program from source to assembly to machine object code?

Big Picture

Compiler output is assembly files

Assembler output is obj files

Linker joins object files into one executable

Loader brings it into memory and starts execution

Example: Add 1 to 100

```
int n = 100;

int main (int argc, char* argv[ ]) {
                int i;
                int m = n;
                int sum = 0;

                for (i = 1; i <= m; i++)
                   sum += i;

                printf ("Sum 1 to %d is %d\n", n, sum);
}
```

```
export PATH=${PATH}:/courses/cs3410/mipsel-linux/bin:/courses/cs3410/mips-sim/bin
or
setenv PATH ${PATH}:/courses/cs3410/mipsel-linux/bin:/courses/cs3410/mips-sim/bin
```

```
# Assemble
[csug03] mipsel-linux-gcc –S add1To100.c
```

Example: Add 1 to 100

```
        .data
        .globl  n
        .align  2
n:      .word   100
        .rdata
        .align  2
$str0: .asciiz
           "Sum 1 to %d is %d\n"
        .text
        .align  2
        .globl  main
main:   addiu   $sp,$sp,-48
        sw      $31,44($sp)
        sw      $fp,40($sp)
        move    $fp,$sp
        sw      $4,48($fp)
        sw      $5,52($fp)
        la      $2,n
        lw      $2,0($2)
        sw      $2,28($fp)
        sw      $0,32($fp)
        li      $2,1
        sw      $2,24($fp)
```

```
$L2:    lw      $2,24($fp)
        lw      $3,28($fp)
        slt     $2,$3,$2
        bne     $2,$0,$L3
        lw      $3,32($fp)
        lw      $2,24($fp)
        addu    $2,$3,$2
        sw      $2,32($fp)
        lw      $2,24($fp)
        addiu   $2,$2,1
        sw      $2,24($fp)
        b       $L2
$L3:    la      $4,$str0
        lw      $5,28($fp)
        lw      $6,32($fp)
        jal     printf
        move    $sp,$fp
        lw      $31,44($sp)
        lw      $fp,40($sp)
        addiu   $sp,$sp,48
        j       $31
```

Example: Add 1 to 100

```
        .data
        .globl  n√
        .align  2
n:      .word   100
        .rdata
        .align  2
$str0:  .asciiz
           "Sum 1 to %d is %d\n"
        .text
        .align  2
        .globl  main√
main:   addiu        $sp,$sp,-48
        sw           $31,44($sp)
prolog  sw           $fp,40($sp)
        move         $fp,$sp
        sw      $a0  $4,48($fp)
        sw      $a1  $5,52($fp)
        la      $v0  $2,n
        lw           $2,0($2)   $v0=100
        sw           $2,28($fp) m=100
        sw           $0,32($fp) sum=0
        li           $2,1
        sw           $2,24($fp) i=1
```

```
$L2:    lw      $v0  $2,24($fp)  i=1
        lw      $v1  $3,28($fp)  m=100
        slt          $2,$3,$2    if(m < i)
        bne          $2,$0,$L3   100 < 1
        lw           $3,32($fp)  v1=0(su
        lw           $2,24($fp)  v0=1(i)
        addu         $2,$3,$2    v0=1(0+1)
        sw           $2,32($fp)  sum=1
        lw           $2,24($fp)  i=1
        addiu        $2,$2,1     i=2 (1+1)
        sw           $2,24($fp)  i=2
        b            $L2
$L3:    la      $a0  $4,$str0    str
printf  lw      $a1  $5,28($fp)  m=100
        lw      $a2  $6,32($fp)  sum
        jal          printf
        move         $sp,$fp
epilog  lw           $31,44($sp)
        lw           $fp,40($sp)
        addiu        $sp,$sp,48
        j            $31
```

Example: Add 1 to 100

```
# Assemble
[csug01] mipsel-linux-gcc –c add1To100.s

# Link
[csug01] mipsel-linux-gcc –o add1To100 add1To100.o
${LINKFLAGS}
# -nostartfiles –nodefaultlibs
# -static -mno-xgot -mno-embedded-pic
-mno-abicalls -G 0 -DMIPS -Wall

# Load
[csug01] simulate add1To100
Sum 1 to 100 is 5050
MIPS program exits with status 0 (approx. 2007
instructions in 143000 nsec at 14.14034 MHz)
```

Globals and Locals

Variables	Visibility	Lifetime	Location
Function-Local i, m, sum	w/in func	func invocation	stack
Global n, str	whole prgm	prgm execution	.data
Dynamic A	? Anywhere that has a ptr	b/w malloc and free	heap

```
int n = 100;
int main (int argc, char* argv[ ]) {
        int i, m = n, sum = 0, *A = malloc(4*m + 4);
        for (i = 1; i <= m; i++) { sum += i; A[i] = sum; }
        printf ("Sum 1 to %d is %d\n", n, sum);
}
```

Globals and Locals

Variables	Visibility	Lifetime	Location
Function-Local i, m, sum	w/in func	func invocation	stack
Global n, str	whole prgm	prgm execution	.data
Dynamic A **C Pointers can be trouble**	Anywhere that has a ptr	b/w malloc and free	heap

Globals and Locals

Variables	Visibility	Lifetime	Location
Function-Local i, m, sum	w/in func	func invocation	stack
Global n, str	whole prgm	prgm execution	.data
Dynamic A	Anywhere that has a ptr	b/w malloc and free	heap

C Pointers can be trouble

```
int *trouble()
{ int a; …; return &a; }
```

“addr of” something on the stack!

Invalid after return

```
char *evil()
{ char s[20]; gets(s); return s; }
```

Buffer overflow

```
int *bad()
{ s = malloc(20); … free(s); … return s; }
```

Allocated on the heap

But freed (i.e. a dangling ptr)

(Can’t do this in Java, C#, ...)

Example #2: Review of Program Layout

calc.c

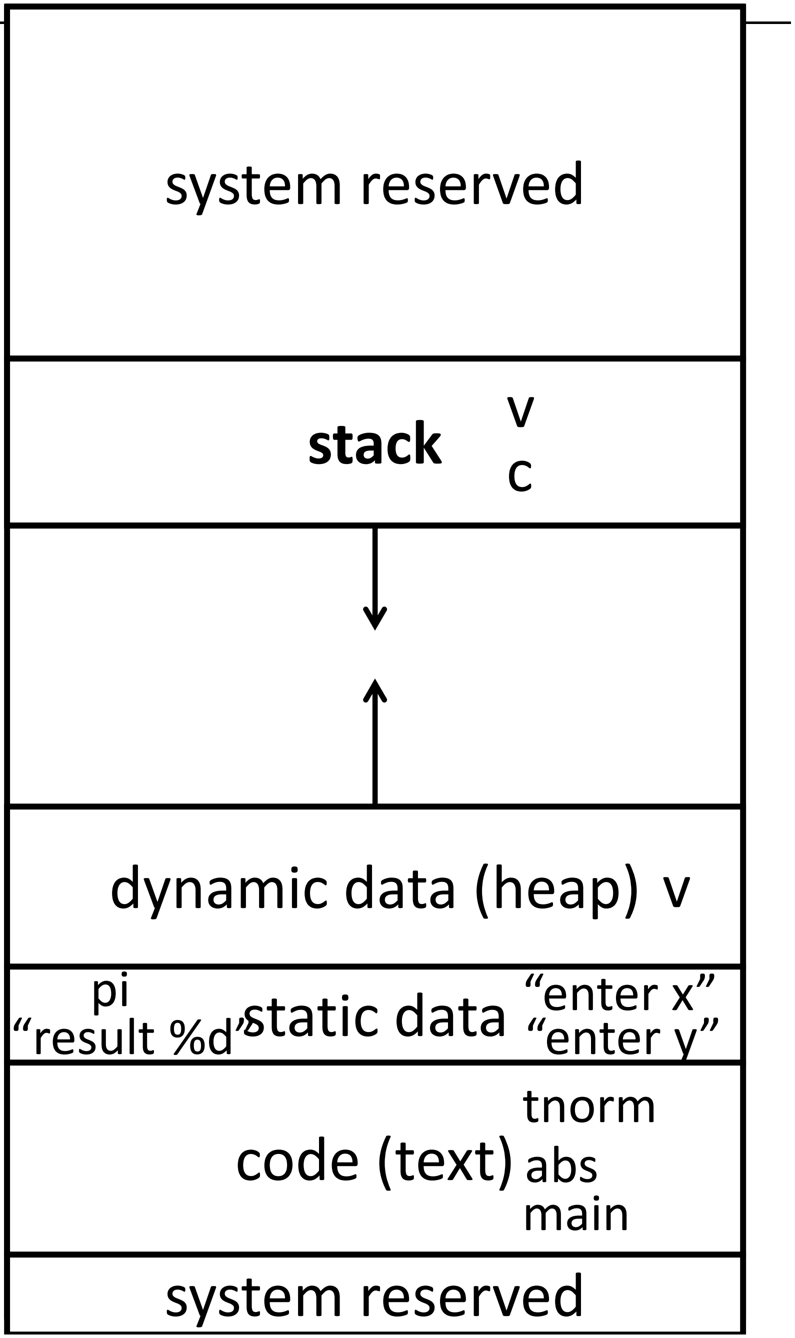
```
vector* v = malloc(8);
v->x = prompt(“enter x”);
v->y = prompt(“enter y”);
int c = pi + tnorm(v);
print(“result %d”, c);
```

math.c

```
int tnorm(vector* v) {
 return abs(v->x)+abs(v->y);
}
```

lib3410.o

```
  global variable: pi
  entry point: prompt
  entry point: print
  entry point: malloc
```



system reserved

stack v c

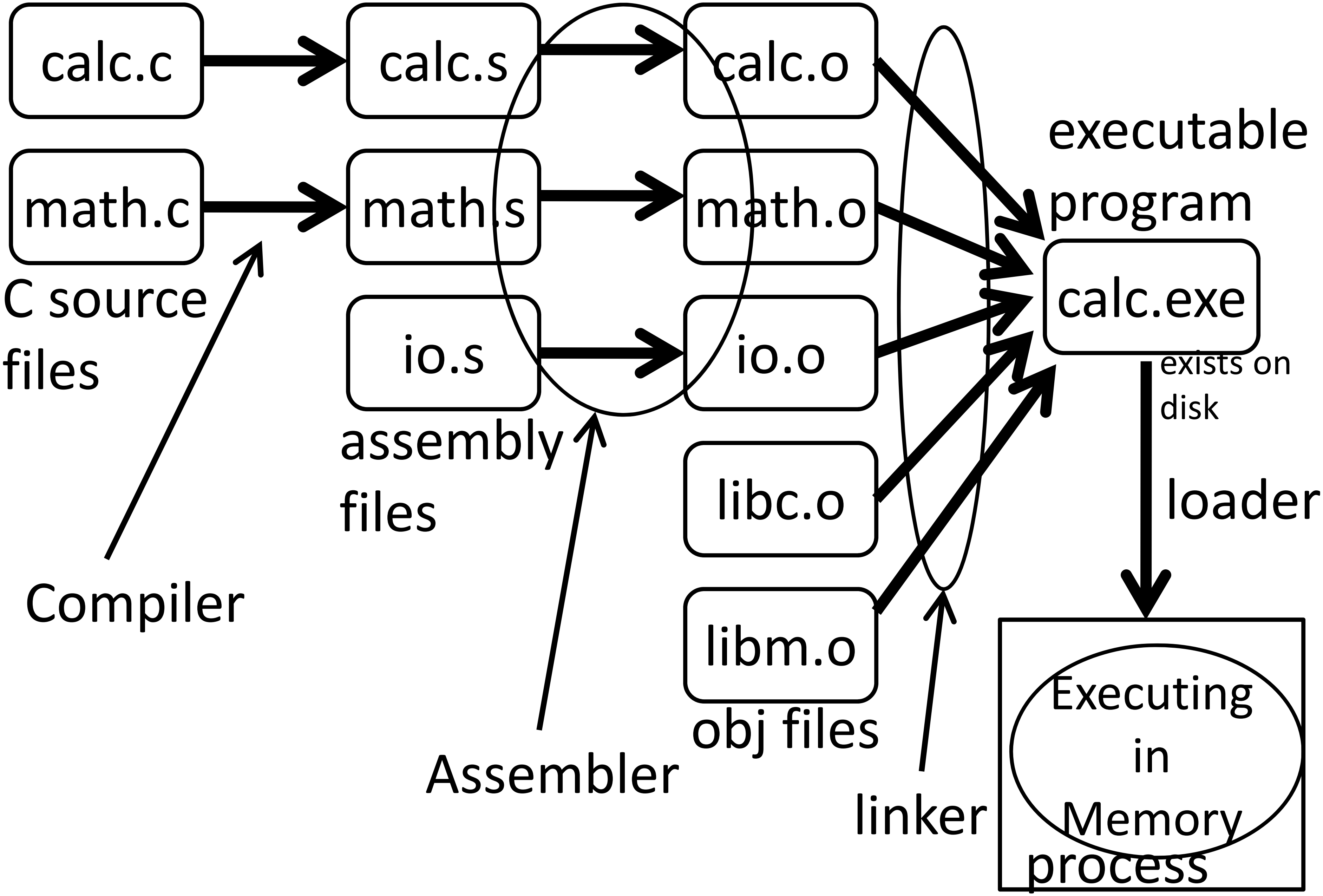
dynamic data (heap) v

pi “result %d” static data “enter x” “enter y”

code (text) tnorm abs main

system reserved

Assembler



calc.c

math.c

C source files

calc.s

math.s

io.s

assembly files

calc.o

math.o

io.o

libc.o

libm.o

obj files

executable program

calc.exe

exists on disk

loader

Executing in Memory

process

Compiler

Assembler

linker

Recap

Compiler output is assembly files

Assembler output is obj files

Next Time

Linker joins object files into one executable

Loader brings it into memory and starts execution

Administrivia

Upcoming agenda

- Schedule PA2 Design Doc Mtg for ***next*** Monday, Mar 11th
- HW3 due next Wednesday, March 13th
- PA2 Work-in-Progress circuit due ***before*** spring break

- Spring break: Saturday, March 16th to Sunday, March 24th

- Prelim2 Thursday, March 28th, right after spring break

- PA2 due Thursday, April 4th