Assembler Directives

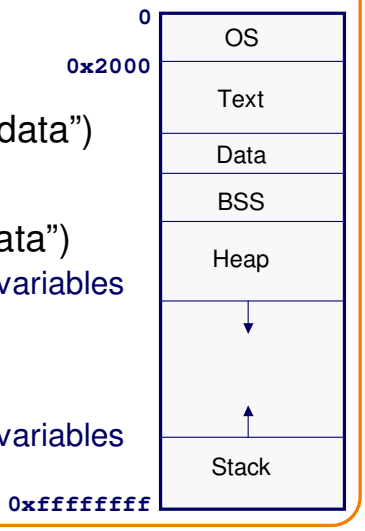
CS 217

Assembler Directives

- Identify sections
- Allocate/initialize memory
- Make symbols externally visible

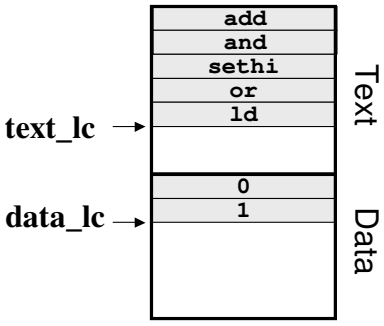
Identifying Sections

- Text (.section ".text")
  - Contains code (instructions)
  - Default section
- Read-Only Data (.section ".rodata")
  - Contains constants
- Read-Write Data (.section ".data")
  - Contains user-initialized global variables
- BSS (.section ".bss")
  - Block starting symbol
  - Contains zero-initialized global variables



Sections (cont)

- Each section has own location counter
  - Location counter is updated when assembler processes directive or instruction



Allocating memory

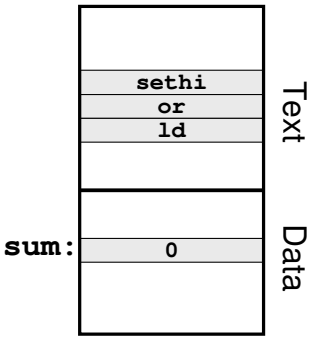
- Increment location counter by nbytes
  - `.skip nbytes`

```
        .section ".bss"
var1:   .skip 16

        .section ".data"
var2:   .skip 4
```

Initializing memory

- Increment location counter and initialize data
  - `.byte  byteval1 [, byteval2 ...]`
  - `.half  halfval1 [, halfval2 ...]`
  - `.word  wordval1 [, wordval2 ...]`

```
        .section ".data"
sum:    .word 0

        .section ".text"
        set sum, %o0
        ld [%o0], %i1
```



Initializing ASCII Data

- Special directives for ascii data

```
.byte  150, 145, 154, 154, 157, 0

.ascii "hello"
.byte 0

.asciz "hello"
```

Making Symbols Externally Visible

- Mark variables as global
  - `.global`

```
        .section ".data"
        .align 4
        .global month
month:  .word jan, feb, mar, apr, may, jun
        .word jul, aug, sep, oct, nov, dec
  jan:  .asciz "January"
  feb:  .asciz "February"
  mar:  .asciz "March"
  apr:  .asciz "April"
  may:  .asciz "May"
  jun:  .asciz "June"
  jul:  .asciz "July"
  ...
```

Making Symbols Externally Visible

- Mark functions as global
  - `.global`

```
        .section ".rodata"
  fmt:  .asciz "Hello, world\n"

        .section ".text"
        .align 4
        .global main
 main:  save %sp, -96, %sp
        set fmt, %o0
        call printf
        nop
        mov 1, %g1
        ta 0
        ret
        restore
```

Example 1

```
int a[100];

main()
{
…
}
```

```
        .section ".bss"
    a:  .skip 4 * 100

        .section ".text"
        .global main
 main:  save %sp, -96, %sp

        clr %10
   L1:  cmp %10, %11
        bge L2; nop
        ...
        sll %10, 2, %12
        ld [a + %12], %13
        ...
        inc %10
        ba L1; nop

   L2:
        mov 1, %g1
        ta 0
        ret
        restore
```

Example 2

```
int a[100];

void swap(int *x, int *y)
{
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}

main()
{
  …
  swap(&a[1],&a[2]);
  …
}
```

```
        .section ".bss"
a:      .skip 4 * 100

        .section ".text"
        .global swap
swap:   ld [%o0], %o2
        ld [%o1], %o3
        st %o2, [%o1]
        retl
        st %o3, [%o0]

        .global main
main:   save %sp, -96, %sp
        …
        set a, %l0
        add %l0, 4, %o0
        call swap
        add %l0, 8, %o1
        …
        mov 1, %g1
        ta 0
        ret
        restore
```

Example 3

```
struct example {
  int a, b;
  char d;
  short x, y;
  int u, v;
};

struct example a =
{
  1, 2,
  'C',
  4, 5,
  6, 7
};

main()
{
…
}
```

```
        .section ".data"
a:      .word 1, 2
        .byte `C'
        .align 2
        .half 3, 4
        .align 4
        .word 6, 7

        .section ".text"
        .align 4
        .global main
main:   save %sp, -96, %sp

        set a, %l0
        ld [%l0 + 0], %l1
        ld [%l0 + 4], %l2
        ldub [%l0 + 8], %l3
        ldsh [%l0 + 10], %l4

        mov 1, %g1
        ta 0
        ret
        restore
```

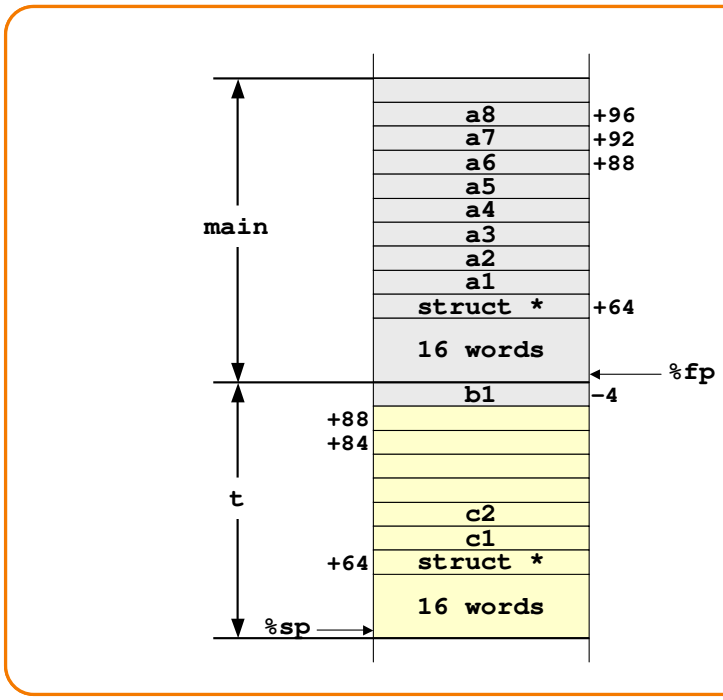
Example 4

```
main() {
   t(1,2,3,4,5,6,7,8);
}

int t(int a1, int a2, int a3, int a4,
  int a5, int a6, int a7, int a8) {
    int b1 = a1;
    return s(b1,a8);
}

static int s(int c1, int c2) {
   return c1 + c2;
}
```

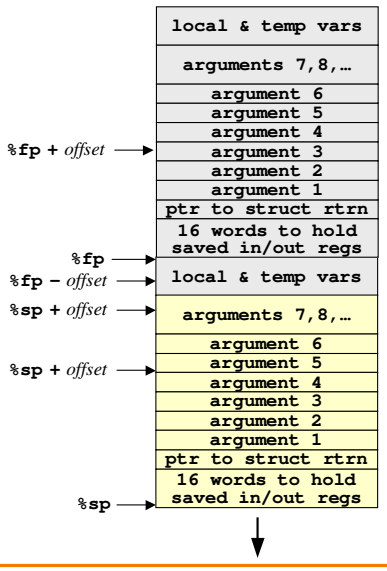
Example 4 (cont)

```
        .global main
main:   save %sp,-104,%sp
        set 1,%o0
        set 2,%o1
        set 3,%o2
        set 4,%o3
        set 5,%o4
        set 6,%o5
        set 7,%i5
        st %i5,[%sp+4*6+68]
        set 8,%i5
        st %i5,[%sp+4*7+68]
        call t; nop
        mov 1, %g1
        ta 0
        ret;
        restore
```

```
        .global t
t:  save %sp,-96,%sp
        st %i0,[%fp-4]
        ld [%fp-4],%o0
        ld [%fp+4*7+68],%o1
        call s; nop
        mov %o0,%i0
        ret; restore

s:  add %o0,%01,%o0
        retl; nop
```


main
t
a8
a7
a6
a5
a4
a3
a2
a1
struct *
16 words
b1
c2
c1
struct *
16 words
+96
+92
+88
+64
%fp
-4
+88
+84
+64
%sp

Stack Frame


local & temp vars
arguments 7,8,…
argument 6
argument 5
argument 4
argument 3
argument 2
argument 1
ptr to struct rtrn
16 words to hold saved in/out regs
local & temp vars
arguments 7,8,…
argument 6
argument 5
argument 4
argument 3
argument 2
argument 1
ptr to struct rtrn
16 words to hold saved in/out regs
%fp + offset
%fp
%fp – offset
%sp + offset
%sp + offset
%sp