Chapter 5
Data Organization: The Load and Store Instructions

Arthur B. Maccabe
Department of Computer Science
The University of New Mexico

Slide 1

Overview

- Assembler directives

  the location counter; the symbol table, allocation, initialization and alignment; assembler segments; and constant expressions

- SPARC load and store instructions

  operand sizes; memory addresses; and encoding load and store instructions

- Data organization

  pointers; arrays; structures; and strings

- Addresses in control: switch
- Addresses and integers
- Addressing modes on the HP-PA

Assembly Language Conventions

Slide 2

Space any number of spaces or tabs; cannot appear in a number or identifier; optional, unless specifically required or prohibited; readability

Comments "!" and everything following on the line.

Identifier an underscore ("_") or a letter followed by any number of underscores, letters, and digits.

Label definition an identifier followed by a colon (":").

Instruction an operation name followed by a list of operands; there must be a space between the operation name and the operand list

Operand list a comma-separated list of operands; number of operands depends on the operation.

Line empty (i.e., white space); a label definition; an instruction; or a label definition followed by an instruction.

Program a sequence of lines.

Assembler Directives—Overview

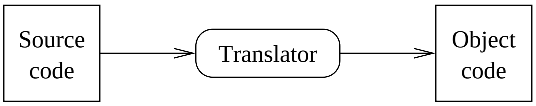
Slide 3

- Sometimes called pseudo-operations
- Like an instruction

  specified on a single line; optional label; name followed by parameters
- Convention: directive names start with '.'
- Translation, the location counter, and the symbol table
- Allocation, initialization, and alignment
- Assembler segments
- Constant expressions

The Translation Process

Slide 4

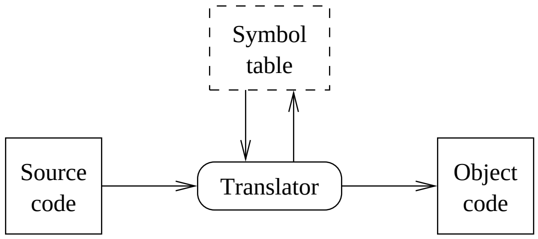


The Location Counter

Slide 5

- Starts at zero; is incremented (by instruction size) as each instruction is translated
- Logical address of the instruction
- Contrast to program counter
  - translation time versus execution time
  - location counter is strictly increasing, program counter can decrease (loops)

The Symbol Table



Slide 6

- Operations: insert and lookup
- Labels

  ⟨*label*, *address*⟩; uses the location counter; forward references are OK

- Symbolic constants

  e.g., .equiv const, 15; ⟨*name*, *value*⟩; uses explicitly supplied value; no forward references; avoid the use of magic numbers

Allocation and Initialization

- Sizes:

Type	Bytes	Bits
char	1	8
short int	2	16
int	4	32
long int	8	64

Slide 7

- Allocation: .skip; e.g., .skip 2
- Initialization:

name	size
.byte	1 byte
.hword	2 bytes
.word	4 bytes
.xword	8 bytes

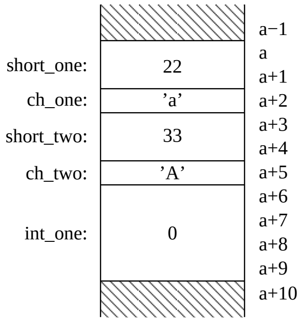
  e.g., .byte 'a', 'b', 'c'

Example

short int short_one = 22;
char ch_one = `'a'`;
short int short_two = 33;
char ch_two = `'A'`;
int int_one = 0;

Slide 8

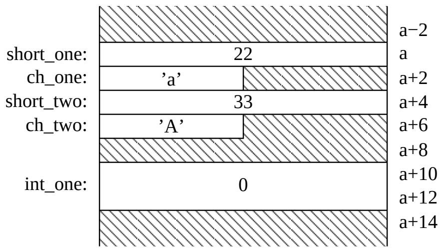
short_one:	.hword	22
ch_one:	.byte	'a'
short_two:	.hword	33
ch_two:	.byte	'A'
int_one:	.word	0



Alignment

Slide 9

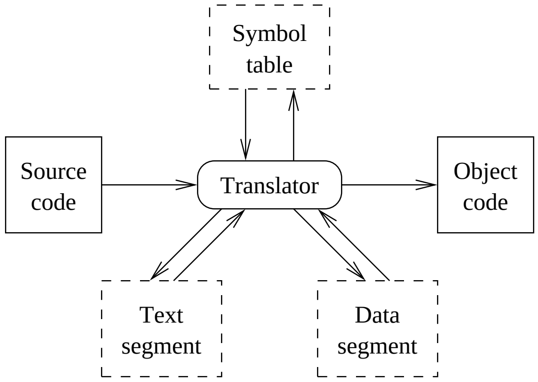
	.align	2
short_one:	.hword	22
ch_one:	.byte	'a'
	.align	2
short_two:	.hword	33
ch_two:	.byte	'A'
	.align	4
int_one:	.word	0



assume a is divisible by 2, but not divisible by 4

Assembler Segments

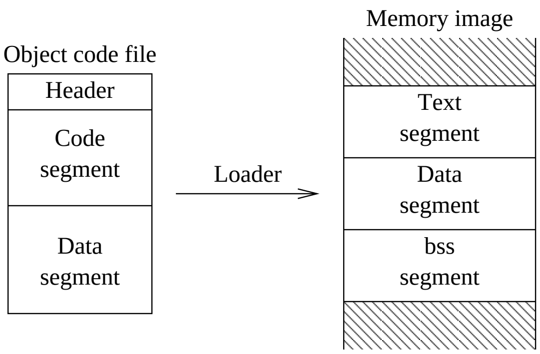
Slide 10



Explicit segment identification: e.g., .seg "data"

The bss Segment

Slide 11



Constant Expressions

Slide 12

- Let the assembler do the math!
- e.g.:

```
.equiv   CarSize, 14

.skip    20 * CarSize
```

- Non relocatable expression

SPARC Load and Store Instructions—Overview

Slide 13

- Store instructions
- Big endian
- Load instructions
- Addressing modes
  - register indirect
  - register indirect with displacement
  - register indirect with index
- Encoding

SPARC Store Instructions

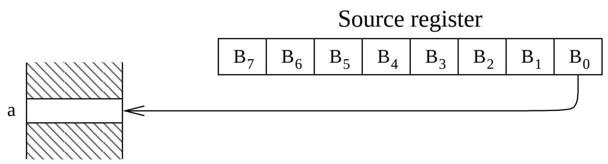
Slide 14

Mnemonic	Operation	Synonyms
stb	Store byte	stub, stsb
sth	Store halfword	stuh, stsh
stw	Store word	st, stuw, stsw
stx	Store extended (double) word	

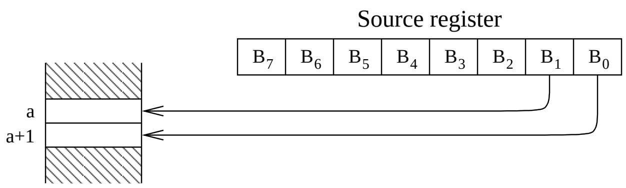
Sizes

Slide 15

- Byte:



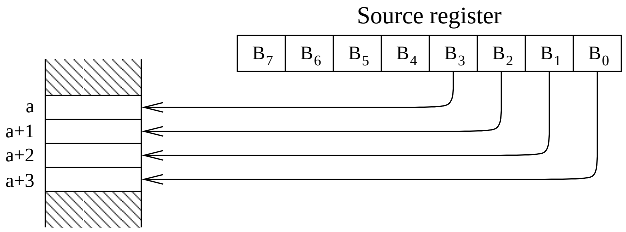
- Halfword:



Sizes (continued)

Slide 16

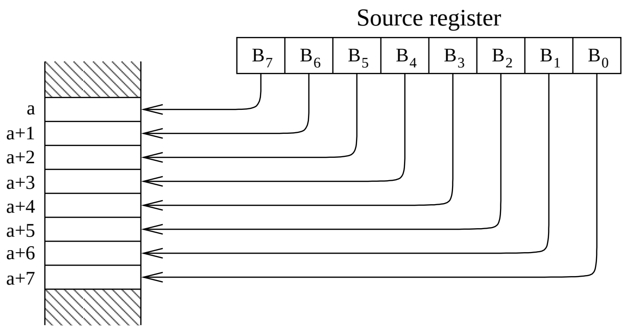
- Word:



Sizes (continued)

Slide 17

- Extended word:



Slide 18

SPARC Load Instructions

Mnemonic	Operation	Synonyms
ldsb	Load signed byte	
ldsh	Load signed halfword	
ldsw	Load signed word	
ldub	Load unsigned byte	
lduh	Load unsigned halfword	
lduw	Load unsigned word	ld
ldx	Load extended (double) word	

Slide 19

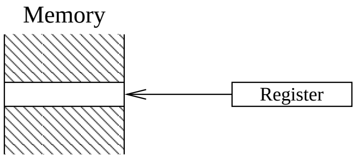
Memory Addresses

- Address description
- Effective address calculation
- Three modes
  - register indirect
  - register indirect with displacement
  - register indirect with index

Register Indirect

Slide 20

- Syntax: register in square brackets (e.g., [%r2])
- Graphical interpretation:



- Code example:

```
lduw  [%r4], %r5
inc   %r5
stw   %r5, [%r4]
```

Using Symbolic Addresses

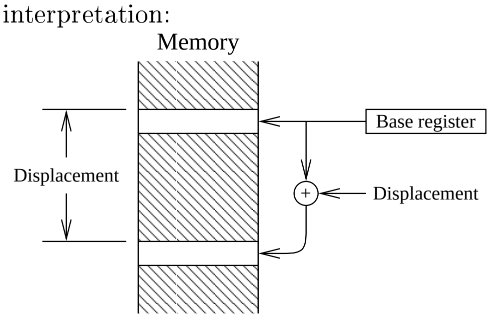
Slide 21

```
    .seg  "data"          ! switch to the data segment
x:  .skip 2               ! allocate 2 bytes

    .seg  "text"          ! switch to the text segment
    setx  x, %r3, %r4     ! put the address of x into %r4
    ldsh  [%r4], %r5
    inc   %r5
    sth   %r5, [%r4]
```

Register Indirect with Displacement

Slide 22

- Syntax: register '+' small constant (e.g., [%r4 + 24])
- Graphical interpretation:



Expanding the "setx"

Slide 23

```
sethi   %uhi(x), %r3        ! start of setx
or      %r3, %ulo(x), %r3
sllx    %r3, 32, %r3
sethi   %hi(x), %r4
or      %r4, %r3, %r4
or      %r4, %lo(x), %r4    ! end of setx
ldsh    [%r4], %r5
inc     %r5
sth     %r5, [%r4]
```

Using Displacement Addressing

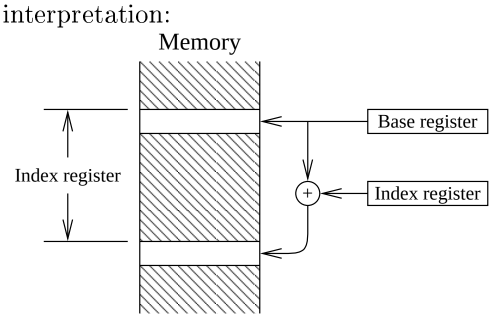
Slide 24

```
sethi  %uhi(x), %r3
or     %r3, %ulo(x), %r3
sllx   %r3, 32, %r3
sethi  %hi(x), %r4
or     %r4, %r3, %r4
ldsh   [%r4+%lo(x)], %r5
inc    %r5
sth    %r5, [%r4+%lo(x)]
```

Register Indirect with Index

Slide 25

- Syntax: base register '+' index register (e.g., [%r4 + %r5])
- Graphical interpretation:



Assuming 32-bit Addresses

Slide 26

```
sethi  %hi(x), %r4
ldsh   [%r4+%lo(x)], %r5
inc    %r5
sth    %r5, [%r4+%lo(x)]
```

Encoding Load and Store Instructions

Slide 27

31 30	29 25	24 19	18 14	13	12 5	4 0
11	rd	op_3	rs_1	0	00000000	rs_2

11	rd	op_3	rs_1	1	simm13

Field	Meaning
rd	Destination register for load instructions, source register for store instructions
op_3	Op code
rs_1	Base register
rs_2	Index register
simm13	Displacement value in 13-bit 2's complement representation

Slide 28

Opcodes

Operation name	op_3	Operation
ldsb	00 1001	Load signed byte
ldsh	00 1010	Load signed halfword
ldsw	00 1000	Load signed word
ldub	00 0001	Load unsigned byte
lduh	00 0010	Load unsigned halfword
lduw	00 0000	Load unsigned word
ldx	00 1011	Load extended word
stb	00 0101	Store byte
sth	00 0110	Store halfword
stw	00 0100	Store word
stx	00 0111	Store extended word

Slide 29

Example Encoding 1

- SPARC instruction: ldsh [%r1+2], %r5
- Format

31 30	29 25	24 19	18 14	13	12 0
11	rd	op_3	rs_1	1	simm13

- Field values

Field	Value	Encoded value
rd	%r5	00101
op_3	ldsh	001010
rs_1	%r1	00001
$simm_{13}$	2	0000000000010

- Encoding: 11 00101 001010 00001 1 0000000000010, or 0xCA506002

Example Encoding 2

Slide 30

- SPARC instruction: ldsh [%r1+%r3], %r5
- Format

31 30	29 25	24 19	18 14	13	12 5	4 0
11	rd	op_3	rs_1	0	00000000	rs_2

- Field values

Field	Value	Encoded value
rd	%dr5	00101
op_3	ldsh	001010
rs_1	%r1	00001
rs_2	%r3	00011

- Encoding: 11 00101 001010 00001 0 00000000 00011, or 0xCA504003

Example Encoding 3

Slide 31

- SPARC instruction: stw %r7, [%r1-12]
- Format

31 30	29 25	24 19	18 14	13	12 0
11	rd	op_3	rs_1	1	simm13

- Field values

Field	Value	Encoded value
rd	%r7	00111
op_3	stw	000100
rs_1	%r1	00001
$simm_{13}$	-12	1111111110100

- Encoding: 11 00111 000100 00001 1 1111111110100, or 0xCE207FF4

Data Organization—Overview

Slide 32

- How is data organized in memory?
- Pointers
- Arrays
- Structures
- Strings

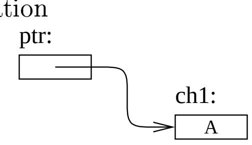
Pointers

Slide 33

- "pointer to" and "address of" (usually "pointer to" is associated with a type)
- Example

  char ch1, ∗ptr;

  ptr = &ch;
  ∗ptr = `'A'`;
- Graphical interpretation



Pointers in Assembly Language

Slide 34

- C code

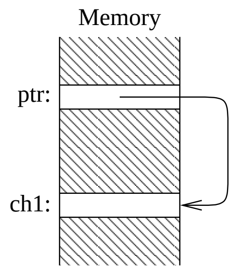
```
char ch1, *ptr;

ptr = &ch1;
```

- SPARC code

```
        .seg    "data"
        .align  8
ptr:    .skip   8                       ! an address is 64 bits
ch1:    .skip   1                       ! a character is one byte
        ⋮
        .seg    "text"
        set     ch1, %r2                ! %r2 = &ch1
        sethi   %hi(ptr), %r3
        stx     %r2, [%r3+%lo(ptr)]     ! store the value in ptr
```

Graphical Interpretation

Slide 35



Pointer Dereferencing

- C code

$*$ptr = 'a';

Slide 36

- SPARC code

```
sethi  %hi(ptr), %r2
ldx    [%r2+%lo(ptr)], %r3   ! load the pointer value
set    'a', %r4
stb    %r4, [%r3]            ! use register indirect addressing
```

Array Layout and Indexing

Slide 37

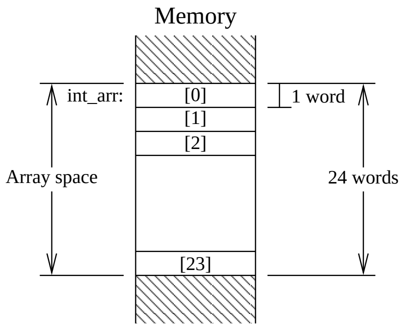
- Basics
- Iterating through an array
- Nonzero-based arrays
- Multidimensional arrays

Arrays: Layout

- Allocation
  - C code:
    int int_arr[24];
  - SPARC directive:
    .reserve int_arr, $24 * 4$, "bss" ! *4 bytes per integer*

Slide 38

- Graphical interpretation



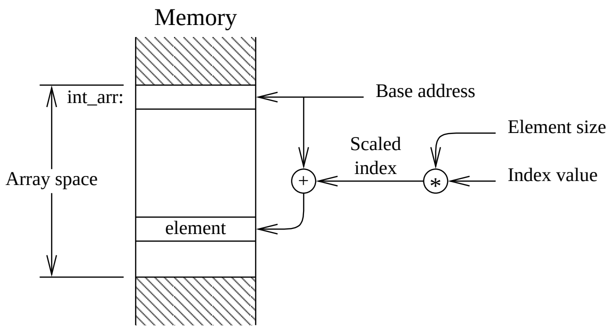
Arrays: Indexing

- Scaled addition

$$\text{addr} = \text{base} + \text{elem\_size} * \text{index}$$

Slide 39

- Graphical interpretation



Indexing Example 1

Slide 40

- C code

  int_arr[5] = 47;

- SPARC code

```
set   int_arr+5*4, %r2   ! the indexed address calculation
set   47, %r3            ! put the value 47 in a register
stw   %r3, [%r2]         ! the store instruction
```

Indexing Example 2

Slide 41

- C code

  int_arr[i] = 47;

- SPARC code

```
set    int_arr, %r2          ! put base address in a register
sethi  %hi(i), %r3
ldsw   [%r3+%lo(i)], %r3     ! load the value of i
sll    %r3, 2, %r3           ! scale the index
set    47, %r4               ! put the value 47 in a register
stw    %r4, [%r2+%r3]        ! store the value
```

Sum Array Elements

Slide 42

```
int arr[20];
int sum, i;

sum = 0;
for( i = 0 ; i < 20 ; i++ ) {
    sum = sum + arr[i];
}
```

Using Simple Control Transfers

Slide 43

```
    int arr[20];
    int sum, i;

    sum = 0;
    i = 0;
top:
    sum = sum + arr[i];
    i = i + 1;
if( i < 20 ) goto top
```

SPARC code

Slide 44

```
        .reserve  arr, 20*4, "bss"
        .seg      "data"
sum:    .skip     4

        .seg      "text"
        clr       %r2                ! sum = 0;
        clr       %r3                ! i = 0;
        set       arr, %r4           ! %r4 holds the base address
```

SPARC code (continued)

Slide 45

```
top:
        ldsw      [%r4+%r3], %r5     ! arr[i]
        add       %r5, %r2, %r2      ! sum = sum + arr[i];
        inc       4, %r3             ! i = i + 1;
        cmp       %r3, 80            ! i < 20
        bl        top
        nop

        sethi     %hi(sum), %r5
        stw       %r2, [%r5+%lo(sum)]
```

Nonzero Arrays

Slide 46

- Basic indexing expression

$$\text{addr} = \text{base} + \text{elem\_size} * (\text{index} - \text{lower})$$

- Gathering constants

$$\text{addr} = (\text{base} - \text{elem\_size} * \text{lower}) + \text{elem\_size} * \text{index}$$

- Savings:
  - when from: subtract, multiply, add
  - to: multiply, add

Example: Allocation and Definitions

Slide 47

- Context
  - an array of integers
  - upper bound 34
  - lower bound 15
- SPARC code

```
   .seg       "data"
   .align     4
i: .skip      4
   .reserve   real_base, (34-15+1)*4, "bss"   ! 4 bytes per integer
   .equiv     img_base, real_base - 15*4
```

Example: Indexing

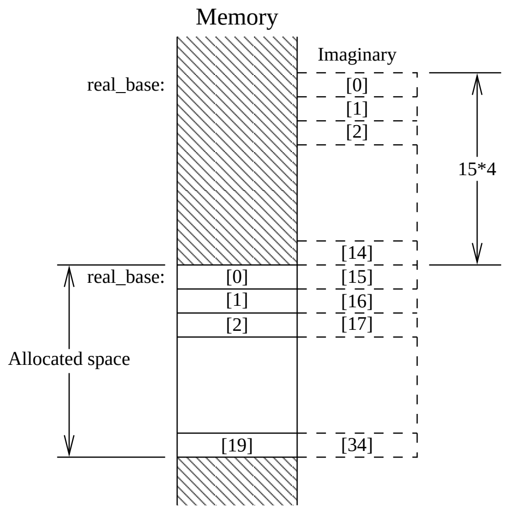
Slide 48

- set element with index i to 432
- SPARC code

```
.seg    "text"
set     img_base, %r2          ! put base address in a register
sethi   %hi(i), %r3
ldsw    [%r3+%lo(i)], %r3      ! load the value of i
sll     %r3, 2, %r3            ! scale the index
set     432, %r4
stw     %r4, [%r2+%r3]         ! store the value
```

Graphical Interpretation

Slide 49



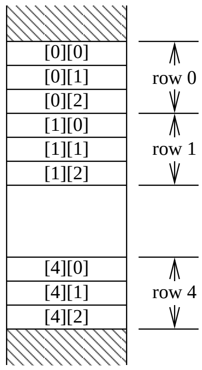
Multidimensional Arrays

Slide 50

- Array of arrays
- C declaration: int two_arr[5][3];
- Row major layout (FORTRAN uses column major layout)

Graphical Interpretation

Slide 51



Example: Declarations

Slide 52

- C code

```
int i, j;
int two_arr[5][3];
```

- SPARC code

```
       .seg      "data"
       .align    4
    i: .skip     4
    j: .skip     4
       .reserve  two_arr, (5*3)*4, "bss"   ! 5 arrays of 3 elements
                                           ! of 4 bytes
```

Example: Access

Slide 53

- C code

```
two_arr[i][j] = 47;
```

- SPARC code

```
    .seg    "text"
    set     two_arr, %r2          ! %r2 holds the array base address
    sethi   %hi(i), %r3
    ldsw    [%r3+%lo(i)], %r3     ! load i
    mulx    %r3, 3*4, %r3         ! scale by the size of a row
    add     %r2, %r3, %r2         ! %r2 holds the row base address
    sethi   %hi(j), %r3
    ldsw    [%r3+%lo(j)], %r3     ! load j
    sll     %r3, 2, %r3           ! scale by the element size
    set     47, %r4               ! the base address
    stw     %r4, [%r2+%r3]
```

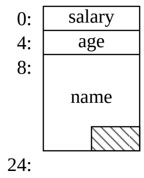
Structures

Slide 54

- C declarations

```
struct person            /* declare a structure */
    char name[14];
    int salary, age;
};
```

- Template



- Padding introduced to support tiling (following one after the other with no alignment problems)

SPARC Template

Slide 55

```
! defines for the structure person
.equiv   SALARY, 0   ! the salary member starts at offset 0
.equiv   AGE, 4      ! the age member starts at offset 4
.equiv   NAME, 8     ! the name member starts at offset 8
.equiv   P_SIZE, 24  ! each structure is 24 bytes
```

Access: Using '.'

Slide 56

- C code

```
struct person joe;        /* joe is a structure variable */

joe.age = 25;             /* joe is 25 */
```

- SPARC code

```
        .seg    "data"
        .align  4
joe:    .skip   P_SIZE                      ! joe is a structure

        .seg    "text"
        set     25, %r2
        sethi   %hi(joe+AGE), %r3
        stw     %r2, [%r3+%lo(joe+AGE)]     ! joe.age = 25
```

Access: Using '→'

Slide 57

- C code

```
struct person *sam;       /* sam is a pointer to a structure */

sam→age = 32;             /* sam is 32 */
```

- SPARC code

```
        .seg    "data"
        .align  8
sam:    .skip   8                           ! sam is a pointer

        .seg    "text"
        set     32, %r2
        sethi   %hi(sam), %r3
        ldx     [%r3+%lo(sam)], %r4         ! %r4 points to the structure
        stw     %r2, [%r4+AGE]              ! sam→age = 32
```

Strings

Slide 58

- Directives: .ascii and .asciz
- Representations
  - Length plus value
  - NULL terminated
- An Example

Length Plus Value

Slide 59

- Length in first byte
- Value in remaining bytes
- Example

my_str:	.byte	last_ch − first_ch	! *just the length*
first_ch:	.ascii	"Hello, world."	
last_ch:			! *marking the end*

- Size of length establishes maximum string length

NULL Terminated

Slide 60

- Value followed by terminating character
- Example

  my_str: .asciz "Hello, world."

- Inefficiency of string length and concatenation

Calculating String Length

Slide 61

```
char str[128];
int len;

len = 0;
while( str[len] ≠ 0 ) {
    len = len + 1;
}
```

Simple Control Transfers

Slide 62

```
len = 0;
goto test;

top:
    len = len + 1;
test:
if( str[len] ≠ 0 ) goto top;
```

SPARC Code

Slide 63

```
        .seg        "data"
        .align      8
        .reserve    str, 128, "bss"
len:    .skip 4
```

SPARC Code (continued)

Slide 64

```
        .seg    "text"
        clr     %r2                     ! len = 0;
        set     str, %r3
        ba      %xcc, test              ! goto test
        nop

top:
        inc     %r2                     ! len = len + 1;
test:
        ldub    [%r3+%r2], %r4          ! load str[len]
        brnz    %r4, top                ! if( str[len] != 0 ) goto top
        nop

        sethi   %hi(len), %r4
        stw     %r2, [%r4+%lo(len)]     ! store len
```

Addresses in Control—switch

Slide 65

- Sample switch statement

```
switch (i) {
case 1:
    /* statement 1 */
    break;
case 2:
case 4:
    /* statement 2 */
    break;
case 6:
    /* statement 3 */
}
```

- Cascaded comparisons
- Branch table

Cascaded Comparisons

Slide 66

```
        .seg    "text"
        sethi   %hi(i), %r2
        ldw     [%r2+%lo(i)], %r3
        cmp     %r3, 1
        bne     %xcc, try2
        nop                             ! branch delay slot
        ⋮                               ! code for stmt1
        bra     %xcc, last
        nop                             ! branch delay slot
```

Cascaded Comparisons (continued)

Slide 67

```
try2:
        cmp     %r3, 2
        bre     %xcc, lab2
        nop                             ! branch delay slot
        cmp     %r3, 4
        bne     try3
        nop                             ! branch delay slot
lab2:
        ⋮                               ! code for stmt2
        bra     %xcc, last
        nop                             ! branch delay slot
```

Cascaded Comparisons (continued)

Slide 68

```
try3:
        cmp     %r3, 6
        bne     %xcc, last
        nop                         ! branch delay slot
        ⋮                           ! code for stmt3
last:
```

Branch Table (the table)

Slide 69

```
        .seg    "data"
tab:                        ; initialize the branch table
        .word   lab1        ; (i = 1) offset 0 → stmt1
        .word   lab2        ; (i = 2) offset 1 → stmt2
        .word   last        ; (i = 3) offset 2 → last
        .word   lab2        ; (i = 4) offset 3 → stmt2
        .word   last        ; (i = 5) offset 4 → last
        .word   lab3        ; (i = 6) offset 5 → stmt3
```

Branch Table (branching)

Slide 70

```
.seg    "text"
sethi   %hi(i), %r2
ldsw    [%r2+%lo(i)], %r3
cmp     %r3, 1
blt     %xcc, last          ! nothing to do
cmp     %r3, 6
bgt     last                ! nothing to do
set     tab−4, %r2          ! normalized table base address
sll     %r3, 2, %r3         ! scale the index
ldw     [%r2+%r3], %r4      ! %r4 holds target address
jmp     %r4
nop                         ! branch delay slot
```

Branch Table (the statements)

Slide 71

```
lab1:
        ⋮                       ! code for stmt1
        bra     %xcc, last
        nop                     ! branch delay slot
lab2:
        ⋮                       ! code for stmt2
        bra     %xcc, last
        nop                     ! branch delay slot
lab3:
        ⋮                       ! code for stmt3
last:
```

Addresses and Integers

Slide 72

- Add an integer to an address
- Subtract and integer from an address
- Subtract two addresses

Addressing Modes on the HP PA

Slide 73

- Displacement

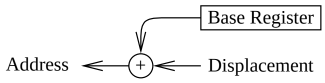
  pre and post modification

- Indexed
  - post modification
  - scaling

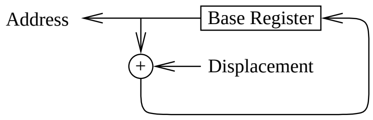
Displacement Addressing on the HP PA

Slide 74

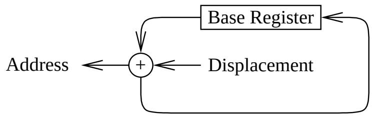
- Base register not modified



- Base register modified, post-displacement



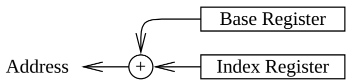
- Base register modified, pre-displacement



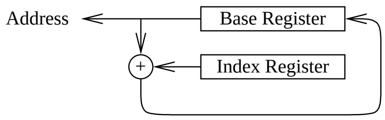
Indexed Addressing on the HP PA

Slide 75

- Simple indexing



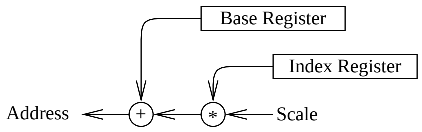
- Base register modification



Indexed Addressing on the HP PA (continued)

Slide 76

- Indexing with scaling



- Base register modification with scaling