

Pointers and arrays

Lecture B13



Lecture notes section B1.3

2002-01-29

CSE1303 Part B lecture notes

1

Last time

- C goto
- MIPS branch and jump instructions
- Selection
 - if-else
- Iteration (loops)
 - while
 - for
 - do-while

2002-01-29

CSE1303 Part B lecture notes

2

In this lecture

- Pointers
 - implementation
 - pointer operations
 - dereference (*) operator
 - address (&) operator
- Arrays
 - implementation
 - indices and addresses
 - relationship to pointers

2002-01-29

CSE1303 Part B lecture notes

3

Pointers

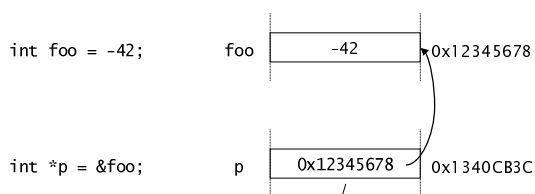
- Pointer is a variable which contains address of some data
 - pointer is same size, no matter how big the data it points to
 - in MIPS, 4 bytes
 - to get address of a variable, use address operator (&)
 - to access data through pointer, use dereference operator (*)

2002-01-29

CSE1303 Part B lecture notes

4

Pointers: & operator



p is a pointer: it contains the address of a piece of data

2002-01-29

CSE1303 Part B lecture notes

5

Pointers: & operator

- In MIPS, address operator (&) is implemented using `la` (load address) instruction
 - `la` computes address of a value in memory, puts result in register
 - doesn't actually fetch contents of value from memory

2002-01-29

CSE1303 Part B lecture notes

6

Pointers: & operator

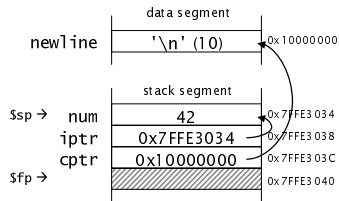
```

/* Example of pointer
operations (part 1). */
char newline = '\n';
int main()
{
    int num = 42;
    int *iptr;
    char *cptr;

    /* Get addresses
of vars. */
    cptr = &newline;
    iptr = &num;

    /* program continues
later ... */
}

```



2002-01-29

CSE1303 Part B lecture notes

7

Pointers: & operator

```

/* Example of pointer
operations (part 1). */
char newline = '\n';
int main()
{
    int num = 42;
    int *iptr;
    char *cptr;

    /* Get addresses
of vars. */
    cptr = &newline;
    iptr = &num;

    /* program continues
later ... */
}

```

```

.data
newline: .byte '\n'

.text
main:
    # 12 bytes of locals.
    move $fp, $sp
    subu $sp, $sp, 12

    # Initialize num = 42.
    li $v0, 42
    sw $v0, -12($fp) # num

    # cptr = &newline;
    la $t0, newline
    sw $t0, -4($fp) # cptr

    # iptr = &num;
    la $t0, -12($fp) # num
    sw $t0, -8($fp) # iptr

# more...

```

2002-01-29

CSE1303 Part B lecture notes

8

Pointers: & operator

```

/* Example of pointer
operations (part 1). */
char newline = '\n';
int main()
{
    int num = 42;
    int *iptr;
    char *cptr;

    /* Get addresses
of vars. */
    cptr = &newline;
    iptr = &num;

    /* program continues
later ... */
}

```

```

.data
newline: # allocate one char.
        .byte '\n'

.text
main:
    # 12 bytes of locals.
    move $fp, $sp
    subu $sp, $sp, 12

    # Initialize num = 42.
    li $v0, 42
    sw $v0, -12($fp) # num

    # cptr = &newline;
    la $t0, newline
    sw $t0, -4($fp) # cptr

    # iptr = &num;
    la $t0, -12($fp) # num
    sw $t0, -8($fp) # iptr

# more...

```

2002-01-29

CSE1303 Part B lecture notes

9

Pointers: * operator

- In MIPS, dereference operator (*) implemented using load instructions
 - lb, lh, lw, depending on size of thing being pointed to
 - or store instructions (sb, sh, sw) if dereference is on left side of assignment
- Load pointer value (the address) into a register \$reg with lw
- Load using address 0(\$reg)
 - i.e., address is \$reg + 0
 - memory in address pointed to by \$reg (the pointer) is loaded

2002-01-29

CSE1303 Part B lecture notes

10

Pointers: * operator

```

/* Pointers, part 2. */
/* This doesn't use
pointers (yet): */
/* Print num then newline. */
printf("%d", num);
putc(newline);
/* more later ... */

```

```

# continued...
# printf("%d", num);
li $v0, 1 # print int
lw $t0, -12($fp) # num
syscall

# Syscall 11 prints a
# character (in $a0).

# putc(newline);
li $v0, 11 # print char
lbu $a0, newline
syscall

# more...

```

2002-01-29

CSE1303 Part B lecture notes

11

Pointers: * operator

```

/* Pointers, part 2. */
/* This doesn't use
pointers (yet): */
/* Print num then newline. */
printf("%d", num);
putc(newline);
/* more later ... */

```

```

# continued...
# printf("%d", num);
li $v0, 1 # print int
lw $t0, -12($fp) # num
syscall

# Syscall 11 prints a
# character (in $a0).

# putc(newline);
li $v0, 11 # print char
lbu $a0, newline
syscall

# more...

```

2002-01-29

CSE1303 Part B lecture notes

12

Pointers: * operator

```

/* Pointers, part 3. */
/* Change num to 87
   through iptr. */
*iptr = 87;

/* Print num through iptr. */
printf("%d", *iptr);

/* Print newline
   through cptr. */
putc(*cptr);

exit(0);
}

```

```

# continued...

# *iptr = 87;
li $t0, 87
lw $t1, -8($fp) # iptr
sw $t0, 0($t1) # *iptr

# printf("%d", *iptr);
li $v0, 1 # print int
lw $t0, -8($fp) # iptr
lw $a0, 0($t0) # *iptr
syscall

# putc(*cptr);
li $v0, 11 # print char
lw $t0, -4($fp) # cptr
lbu $a0, 0($t0) # *cptr
syscall

li $v0, 10 # exit
syscall

```

2002-01-29

CSEI 303 Part B lecture notes

13

Pointers: * operator

```

/* Pointers, part 3. */
/* Change num to 87
   through iptr. */
*iptr = 87;

/* Print num through iptr. */
printf("%d", *iptr);

/* Print newline
   through cptr. */
putc(*cptr);

exit(0);
}

```

```

# continued...

# *iptr = 87;
li $t0, 87
lw $t1, -8($fp) # iptr
sw $t0, 0($t1) # *iptr

# printf("%d", *iptr);
li $v0, 1 # print int
lw $t0, -8($fp) # iptr
lw $a0, 0($t0) # *iptr
syscall

# putc(*cptr);
li $v0, 11 # print char
lw $t0, -4($fp) # cptr
lbu $a0, 0($t0) # *cptr
syscall

li $v0, 10 # exit
syscall

```

2002-01-29

CSEI 303 Part B lecture notes

14

Why use pointers in MIPS?

- All the same reasons to use pointers in C
 - to pass values to functions by reference
 - for efficiency
 - to have them modified
 - to dynamically allocate memory
 - e.g., with `malloc()`
 - to implement self-referential data structures
 - e.g., lists, trees
 - to implement arrays
 - by pointer/array duality
 - and, by extension, strings (arrays of char)

2002-01-29

CSEI 303 Part B lecture notes

15

Arrays

The programmer's view: array is accessed through indices 0, 1, 2, ...

a[0]	0
a[1]	-1
a[2]	4
a[3]	-9
a[4]	16

2002-01-29

CSEI 303 Part B lecture notes

16

Arrays

The computer's view: array is part of memory, accessed through addresses 0x10002FC0, 0x10002FC4, ...

0	0x10002FC0
-1	0x10002FC4
4	0x10002FC8
-9	0x10002FCC
16	0x10002FD0

2002-01-29

CSEI 303 Part B lecture notes

17

Arrays

To program arrays in assembly language, need to understand relationship between indices and addresses, and need to convert between them.

a[0]	←→	0x10002FC0
a[1]	←→	0x10002FC4
a[2]	←→	0x10002FC8
a[3]	←→	0x10002FCC
a[4]	←→	0x10002FD0

2002-01-29

CSEI 303 Part B lecture notes

18

Arrays

arrays: first index is always 0

addresses: for a given array, address of first element is constant (here, 0x10002FC0)

a[0]	←→	0x10002FC0
a[1]	←→	0x10002FC4
a[2]	←→	0x10002FC8
a[3]	←→	0x10002FCC
a[4]	←→	0x10002FD0

2002-01-29 CSEI 303 Part B lecture notes 19

Arrays

arrays: adjacent indices differ by 1

addresses: addresses differ by size of array element type (here, 4 bytes for int)

	←→	0x10002FC0	
+1	←→	0x10002FC4	+4
+1	←→	0x10002FC8	+4
+1	←→	0x10002FCC	+4
+1	←→	0x10002FD0	+4

2002-01-29 CSEI 303 Part B lecture notes 20

Arrays

This is a linear relationship; to calculate any y value, need y-intercept, slope and an x value

slope ("m") is size of one element

y-intercept ("c") is address of a[0]

	←→	0x10002FC0	
+1	←→	0x10002FC4	+4
+1	←→	0x10002FC8	+4
+1	←→	0x10002FCC	+4
+1	←→	0x10002FD0	+4

x value is the desired index

now calculate $y = mx + c$

2002-01-29 CSEI 303 Part B lecture notes 21

Arrays

- To compute address of a[i]
 - determine start address of array
 - address of a[0]
 - numerically smallest address of entire array
 - determine size of one element of a
 - in bytes
 - compute i
 - can be an arbitrary integer expression
 - address is start + size * i
- Need load/store to access a[i]'s data
 - since above calculation only computes address of (i.e., pointer to) a[i]

2002-01-29 CSEI 303 Part B lecture notes 22

Arrays

```

/* Array example. */
int g[5] = {
};
45, -3, 0, 16, -23
int main()
{
  int i;
  int total[6];

  /* Get an array of running
  totals. */
  total[0] = 0;
  for (i = 0; i < 5; i++)
  {
    total[i + 1] =
    total[i] + g[i];
  }

  /* Print final total. */
  printf("%d", total[5]);
}

```

	\$sp →		i			
		total	[0]	6	0x7FFD14D8	
			[1]	0	0x7FFD14DC	
			[2]	45	0x7FFD14E0	
			[3]	42	0x7FFD14E4	
			[4]	42	0x7FFD14E8	
			[5]	58	0x7FFD14EC	
				35	0x7FFD14F0	
				[]	0x7FFD14F4	

2002-01-29 CSEI 303 Part B lecture notes 23

Arrays

```

/* Array example. */
int g[5] = {
};
45, -3, 0, 16, -23
int main()
{
  int i;
  int total[6];

  /* Get an array of running
  totals. */
  total[0] = 0;
  for (i = 0; i < 5; i++)
  {
    total[i + 1] =
    total[i] + g[i];
  }

  /* Print final total. */
  printf("%d", total[5]);
}

```

```

g:      .data
        .word 45, -3, 0, 16, -23
main:   # local variables:
        # i * int (i)
        # + 6 * int (total)
        # = 28 bytes
        move $fp, $sp
        subu $sp, $sp, 28

        # total[0] = 0;
        la $t0, -24($fp) # total
        sw $0, 0($t0)

        # i = 0;
        sw $0, -28($fp) # i

loop:   # i < 5
        lw $t0, -28($fp) # i
        bge $t0, 5, end

# continued ...

```

2002-01-29 CSEI 303 Part B lecture notes 24

Arrays

```

/* Array example. */
int g[5] = {
    45, -3, 0, 16, -23
};

int main()
{
    int i;
    int total[6];

    /* Get an array of running
    totals. */
    total[0] = 0;
    for (i = 0; i < 5; i++)
    {
        total[i + 1] =
            total[i] + g[i];
    }

    /* Print final total. */
    printf("%d", total[5]);
}

```

```

.data
g: .word 45, -3, 0, 16, -23

.text
main:
    # local variables:
    # 1 * int (i)
    # + 6 * int (total)
    # = 28 bytes
    move $fp, $sp
    subu $sp, $sp, 28

    # total[0] = 0;
    la $t0, -24($fp) # total
    sw $0, 0($t0)

    # i = 0;
    sw $0, -28($fp) # i

    # i < 5
    lw $t0, -28($fp) # i
    bge $t0, 5, end

    # continued ...

```

2002-01-29

CSEI 303 Part B lecture notes

25

Arrays

```

/* Array example. */
int g[5] = {
    45, -3, 0, 16, -23
};

int main()
{
    int i;
    int total[6];

    /* Get an array of running
    totals. */
    total[0] = 0;
    for (i = 0; i < 5; i++)
    {
        total[i + 1] =
            total[i] + g[i];
    }

    /* Print final total. */
    printf("%d", total[5]);
}

```

```

# ... continued (part 2)

    # shift left 2 used here
    # to scale by sizeof(int)

    # total[i]
    la $t0, -24($fp) # total
    lw $t1, -28($fp) # i
    lw $t2, -28($fp) # g
    sll $t1, $t1, 2 # * 4
    add $t0, $t0, $t1
    add $t0, $t0, $t2
    lw $t2, 0($t0)

    # Add two array elements.
    add $t2, $t2, $t3

    # continued ...

```

2002-01-29

CSEI 303 Part B lecture notes

26

Arrays

```

/* Array example. */
int g[5] = {
    45, -3, 0, 16, -23
};

int main()
{
    int i;
    int total[6];

    /* Get an array of running
    totals. */
    total[0] = 0;
    for (i = 0; i < 5; i++)
    {
        total[i + 1] =
            total[i] + g[i];
    }

    /* Print final total. */
    printf("%d", total[5]);
}

```

```

# ... continued (part 2)

    # shift left 2 used here
    # to scale by sizeof(int)

    # total[i]
    la $t0, -24($fp) # total
    lw $t1, -28($fp) # i
    sll $t1, $t1, 2 # * 4
    add $t0, $t0, $t1
    lw $t2, 0($t0)

    # g[i]
    la $t0, g # g
    lw $t1, -28($fp) # i
    sll $t1, $t1, 2 # * 4
    add $t0, $t0, $t1
    lw $t3, 0($t0)

    # Add two array elements.
    add $t2, $t2, $t3

    # continued ...

```

2002-01-29

CSEI 303 Part B lecture notes

27

Arrays

```

/* Array example. */
int g[5] = {
    45, -3, 0, 16, -23
};

int main()
{
    int i;
    int total[6];

    /* Get an array of running
    totals. */
    total[0] = 0;
    for (i = 0; i < 5; i++)
    {
        total[i + 1] =
            total[i] + g[i];
    }

    /* Print final total. */
    printf("%d", total[5]);
}

```

```

# ... continued (part 3)

    # total[i + 1] = ...
    la $t0, -24($fp) # total
    lw $t1, -28($fp) # i
    add $t1, $t1, 1 # + 1
    sll $t1, $t1, 2 # * 4
    add $t0, $t0, $t1
    sw $t2, 0($t0)

    # i++
    lw $t0, -28($fp) # i
    add $t0, $t0, 1
    sw $t0, -28($fp) # i

    # Repeat loop.
    j loop

    # continued ...

```

2002-01-29

CSEI 303 Part B lecture notes

28

Arrays

```

/* Array example. */
int g[5] = {
    45, -3, 0, 16, -23
};

int main()
{
    int i;
    int total[6];

    /* Get an array of running
    totals. */
    total[0] = 0;
    for (i = 0; i < 5; i++)
    {
        total[i + 1] =
            total[i] + g[i];
    }

    /* Print final total. */
    printf("%d", total[5]);
}

```

```

# ... continued (part 3)

    # total[i + 1] = ...
    la $t0, -24($fp) # total
    lw $t1, -28($fp) # i
    add $t1, $t1, 1 # + 1
    sll $t1, $t1, 2 # * 4
    add $t0, $t0, $t1
    sw $t2, 0($t0)

    # i++
    lw $t0, -28($fp) # i
    add $t0, $t0, 1
    sw $t0, -28($fp) # i

    # Repeat loop.
    j loop

    # continued ...

```

2002-01-29

CSEI 303 Part B lecture notes

29

Arrays

```

/* Array example. */
int g[5] = {
    45, -3, 0, 16, -23
};

int main()
{
    int i;
    int total[6];

    /* Get an array of running
    totals. */
    total[0] = 0;
    for (i = 0; i < 5; i++)
    {
        total[i + 1] =
            total[i] + g[i];
    }

    /* Print final total. */
    printf("%d", total[5]);
}

```

```

# ... continued (part 4)

end:
    # Print total[5]
    li $v0, 1 # print int
    # Allowed arbitrary
    # expression provided it
    # is constant; could have
    # said -4($fp) too.
    lw $a0, -24+5*4($fp) #
    [5]
    syscall

    # Exit
    li $v0, 10 # exit
    syscall

```

2002-01-29

CSEI 303 Part B lecture notes

30

Arrays

```

/* Array example. */
int g[5] = {
    45, -3, 0, 16, -23
};
int main()
{
    int i;
    int total[6];

    /* Get an array of running
    totals. */
    total[0] = 0;
    for (i = 0; i < 5; i++)
        total[i + 1] =
            total[i] + g[i];

    /* Print final total. */
    printf("%d", total[5]);
}

```

```

# ... continued (part 4)
end:
    # Print total[5]
    li $v0, 1 # print int
    # Allowed arbitrary
    # expression provided it
    # is constant; could have
    # said -4($fp) too.
    lw $a0, -24+5*4($fp) #
    syscall

    # Exit
    li $v0, 10 # exit
    syscall

```

2002-01-29

CSEI 303 Part B lecture notes

31

Pointers and arrays

```

/* Print a string using
array accesses. */
char a[] = "Hello world\n";
int i = 0;
char c;

int main()
{
    /* get element of array,
    store in c, and stop
    if it is the end-of-
    string character. */
    while ((c = a[i]) != '\0')
    {
        /* Print it out. */
        putc(c);
        i++;
    }
}

```

```

.data
.asciiz "Hello world\n"
i:
.word 0
.space 1

.text
main:
# calculate a[i] (size = 1)
loop:
la $t0, a # a
lw $t1, i # i
add $t0, $t0, $t1
lbu $t2, 0($t0) # a[i]
sb $t2, c # store in c
beq $t0, '\0', end

# putc(c):
li $v0, 11 # print char
lbu $a0, c
syscall

# i++;
lw $t0, i
add $t0, $t0, 1
sw $t0, i
j loop # repeat loop

end:
li $v0, 10
syscall

```

2002-01-29

CSEI 303 Part B lecture notes

32

Pointers and arrays

```

/* Print a string using
array accesses. */
char a[] = "Hello world\n";
int i = 0;
char c;

int main()
{
    /* get element of array,
    store in c, and stop
    if it is the end-of-
    string character. */
    while ((c = a[i]) != '\0')
    {
        /* Print it out. */
        putc(c);
        i++;
    }
}

```

```

.data
.asciiz "Hello world\n"
i:
.word 0
.space 1

.text
main:
# calculate a[i] (size = 1)
loop:
la $t0, a # a
lw $t1, i # i
add $t0, $t0, $t1
lbu $t2, 0($t0) # a[i]
sb $t2, c # store in c
beq $t0, '\0', end

# putc(c):
li $v0, 11 # print char
lbu $a0, c
syscall

# i++;
lw $t0, i
add $t0, $t0, 1
sw $t0, i
j loop # repeat loop

end:
li $v0, 10
syscall

```

2002-01-29

CSEI 303 Part B lecture notes

33

Pointers and arrays

```

/* Print a string using
pointer dereferences. */
char a[] = "Hello world\n";
int i = 0;
char c;

int main()
{
    /* get element of array,
    store in c, and stop
    if it is the end-of-
    string character. */
    while ((c = *(a+i)) != '\0')
    {
        /* Print it out. */
        putc(c);
        i++;
    }
}

```

```

.data
.asciiz "Hello world\n"
i:
.word 0
.space 1

.text
main:
# calculate *(a+i)
loop:
la $t0, a # a
lw $t1, i # i
add $t0, $t0, $t1 # a+i
lbu $t2, 0($t0) # *(a+i)
sb $t2, c # store in c
beq $t0, '\0', end

# putc(c):
li $v0, 11 # print char
lbu $a0, c
syscall

# i++;
lw $t0, i
add $t0, $t0, 1
sw $t0, i
j loop # repeat loop

end:
li $v0, 10
syscall

```

2002-01-29

CSEI 303 Part B lecture notes

34

Pointers and arrays

```

/* Print a string using
pointer dereferences. */
char a[] = "Hello world\n";
int i = 0;
char c;

int main()
{
    /* get element of array,
    store in c, and stop
    if it is the end-of-
    string character. */
    while ((c = *(a+i)) != '\0')
    {
        /* print it out. */
        putc(c);
        i++;
    }
}

```

```

.data
.asciiz "Hello world\n"
i:
.word 0
.space 1

.text
main:
# calculate *(a+i)
loop:
la $t0, a # a
lw $t1, i # i
add $t0, $t0, $t1 # a+i
lbu $t2, 0($t0) # *(a+i)
sb $t2, c # store in c
beq $t0, '\0', end

# putc(c):
li $v0, 11 # print char
lbu $a0, c
syscall

# i++;
lw $t0, i
add $t0, $t0, 1
sw $t0, i
j loop # repeat loop

end:
li $v0, 10
syscall

```

2002-01-29

CSEI 303 Part B lecture notes

35

Pointers and arrays

# pointer version	# array version
la \$t0, a # a	la \$t0, a # a
lw \$t1, i # i	lw \$t1, i # i
add \$t0, \$t0, \$t1 # a+i	add \$t0, \$t0, \$t1 # a+i
lbu \$t2, 0(\$t0) # *(a+i)	lbu \$t2, 0(\$t0) # a[i]
# ...	# ...

code for pointer version is identical to code for array version

This is why
a[i] = *(a+i)
they compile to the same code.

2002-01-29

CSEI 303 Part B lecture notes

36

Covered in this lecture

- **Pointers**
 - implementation
 - pointer operations
 - dereference (*) operator
 - address (&) operator
- **Arrays**
 - implementation
 - indices and addresses
 - relationship to pointers

2002-01-29

CSE1303 Part B lecture notes

37

Going further

- **Two-dimensional arrays**
 - implement as array-of-arrays
 - same technique as for 1-D arrays, applied twice

2002-01-29

CSE1303 Part B lecture notes

38

Next time

- **Functions**
 - calling
 - returning
 - local variables
 - stack frames



Reading:
Lecture notes section B1.4

2002-01-29

CSE1303 Part B lecture notes

39

Copyright

Copyright © 2001 Deborah Pickett.
No part of this presentation may be duplicated without permission from the author.

2002-01-29

CSE1303 Part B lecture notes

40