MIPS ASSEMBLY PROGRAMMING LANGUAGE PART IV

Ayman Hajja, PhD

REVIEW; TYPES OF BRANCHES

- Conditional Branch: change control flow depending on outcome of comparison
  - branch on equal (beq) or branch on not equal (bne)
- Unconditional Branch: always branch
  - MIPS instruction for this: jump
    - j Label

EXAMPLE IF STATEMENT

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

```
if (i == j)
    f = g + h;
g = f + h;
```

```
    bne   $s3, $s4, Exit
    add   $s0, $s1, $s2
Exit: add  $s1, $s0, $s2
```

EXAMPLE IF/ELSE STATEMENT

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

```
if (i == j)                    bne  $s3, $s4, Else
    f = g + h;                 add  $s0, $s1, $s2
else                           j Exit
    f = g - h;                 Else: sub $s0, $s1, $s2
i = j + h;                     Exit: add $s3, $s4, $s2
```

'SET LESS THAN' INSTRUCTION

- Set less than instruction: slt register_a, register_b, register_c
  - If register_b is less than register_c, assign register_a the value 1, otherwise assign register_a the value 0

- Examples:
  - addi $s0, $zero, 10
  - addi $s1, $zero, 15
  - slt $t0, $s0, $s1 # what's the value in $t0?
  - slt $t0, $s1, $s0 # what's the value in $t0 now?

'SET LESS THAN' INSTRUCTION

- Set less than instruction: slt register_a, register_b, register_c
  - If register_b is less than register_c, assign register_a the value 1, otherwise assign register_a the value 0

- Examples:
  - addi $s0, $zero, 10
  - addi $s1, $zero, 15
  - slt $t0, $s0, $s1 # what's the value in $t0? Answer is 1
  - slt $t0, $s1, $s0 # what's the value in $t0 now? Answer is 0

INEQUALITIES IN MIPS

- Assume that v_s0 maps to $s0, v_s1 maps to $s1
- How do we do the following:
  - if (v_s0 < v_s1), do something; otherwise, do something else
- Answer:

INEQUALITIES IN MIPS

- Assume that v_s0 maps to $s0, v_s1 maps to $s1
- How do we do the following:
  - if (v_s0 < v_s1), do something; otherwise, do something else
- Answer:

```
slt  $t0, $s0, $s1          # $t0 = 1 if g < h, otherwise $t0 = 0
```

INEQUALITIES IN MIPS

- Assume that v_s0 maps to $s0, v_s1 maps to $s1
- How do we do the following:
  - if (v_s0 < v_s1), do something; otherwise, do something else
- Answer:

```
slt  $t0, $s0, $s1          # $t0 = 1 if g < h, otherwise $t0 = 0
beq $t0, $zero, Else        # if $t0 = 0, go to Else
```

INEQUALITIES IN MIPS; EXAMPLE

C variable	s0	s1	s2	s3	s4
Register	$s0	$s1	$s2	$s3	$s4

	1st command	2nd command
if (s3 < s4)	# slt $t0, $s3, $s4 |	beq $t0, $zero, Else
s0 = s1 + s2;	# add $s0, $s1, $s2	
else	# j Exit	
s2 = s1 - s2;	# Else: sub $s2, $s1, $s2	
s3 = s4 + s2;	# Exit: add $s3, $s4, $s2	

QUESTION

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

```
if (i >= j)
        f = g + h;
else
        f = g - h;
i = j + h;
```

1

```
slt  $t0, j, i
beq  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

2

```
slt  $t0, j, i
bne  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

3

```
slt  $t0, i, j
bne  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

4

```
slt  $t0, i, j
beq  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

QUESTION

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

```
if (i >= j)
        f = g + h;
else
        f = g - h;
i = j + h;
```

1

```
slt  $t0, j, i
beq  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

2

```
slt  $t0, j, i
bne  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

3

```
slt  $t0, i, j
bne  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

4

```
slt  $t0, i, j
beq  $t0, $zero, Else
f = g + h
j Exit
Else: f = g - h
Exit: i = j + h
```

SUMMARY OF CONVERTING A CONDITION TO MIPS

- if (s0 == s1) becomes:
  - bne s0, s1, ELSE (or branch after loop if there's no else)
- if (s0 != s1) becomes:
  - beq s0, s1, ELSE (or branch after loop if there's no else)
- if (s0 < s1) becomes
  - slt $t0, $s0, $s1
  - beq $t0, $0, ELSE (or branch after loop if there's no else)
- if (s0 > s1), treat it as (s1 < s0) and do above

SUMMARY OF CONVERTING A CONDITION TO MIPS

- if (s0 <= s1), first we convert condition to: ! (s1 < s0)

SUMMARY OF CONVERTING A CONDITION TO MIPS

If this is true, that means original condition is false and we should go to ELSE

- if (s0 <= s1), first we convert condition to: ! (s1 < s0)

SUMMARY OF CONVERTING A CONDITION TO MIPS

If this is true, that means original condition is false and we should go to ELSE

- if (s0 <= s1), first we convert condition to: ! (s1 < s0)
  - slt $t0, $s1, $s0 # Here, we check opposite condition

If $s1 < $s0, the value of $t will be 1

SUMMARY OF CONVERTING A CONDITION TO MIPS

If this is true, that means original condition is false and we should go to ELSE

- if (s0 <= s1), first we convert condition to: ! (s1 < s0)
  - slt $t0, $s1, $s0 # Here, we check opposite condition
  - bne $t0, $0, ELSE # Or branch after loop if there's no else

If $s1 < $s0, the value of $t will be 1

If $t0 not equal to 0 (must be equal to 1), go to ELSE

IMMEDIATES IN INEQUALITIES

- slti: set less than immediate; used to compare registers to constants;

```
if (a < 100)
{
        line 1
        line 2
}
```

```
slti  $t0, $s0, 100        # $t0 = 1 if $s0 < 100
beq  $t0, $zero, Next      # $s0 == 1, go to Next
Mips line 1
Mips line 2
Next: …
```

MIPS REPETITION STRUCTURES

- To translate loops, it's easier to convert the loops to if-statements with goto statements, prior to translation.

```
while ( i != j )
{
    k = k + 1;
    i = i * 2;
}
```

MIPS REPETITION STRUCTURES

- To translate loops, it's easier to convert the loops to if-statements with goto statements, prior to translation.

```
while ( v_s0 != v_s1 )
{
    v_s2 = v_s2 + 1;
    v_s0 = v_s0 * 2;
}
```

```
L1:
if (v_s0 !=v_s1)
{
    v_s2 = v_s2 + 1;
    v_s0 = v_s0 * 2;
    goto L1;
}
```

MIPS REPETITION STRUCTURES

- To translate loops, it's easier to convert the loops to if-statements with goto statements, prior to translation.

```
L1:
if (v_s0 !=v_s1)
{
        v_s2 = v_s2 + 1;
        v_s0 = v_s0 * 2;
        goto L1;
}
```

```
L1:
beq  $s0, $s1, EXIT
addi $s2, $s2, 1
add  $s0, $s0, $s0
j L1                          # Go to top of loop
EXIT:
```

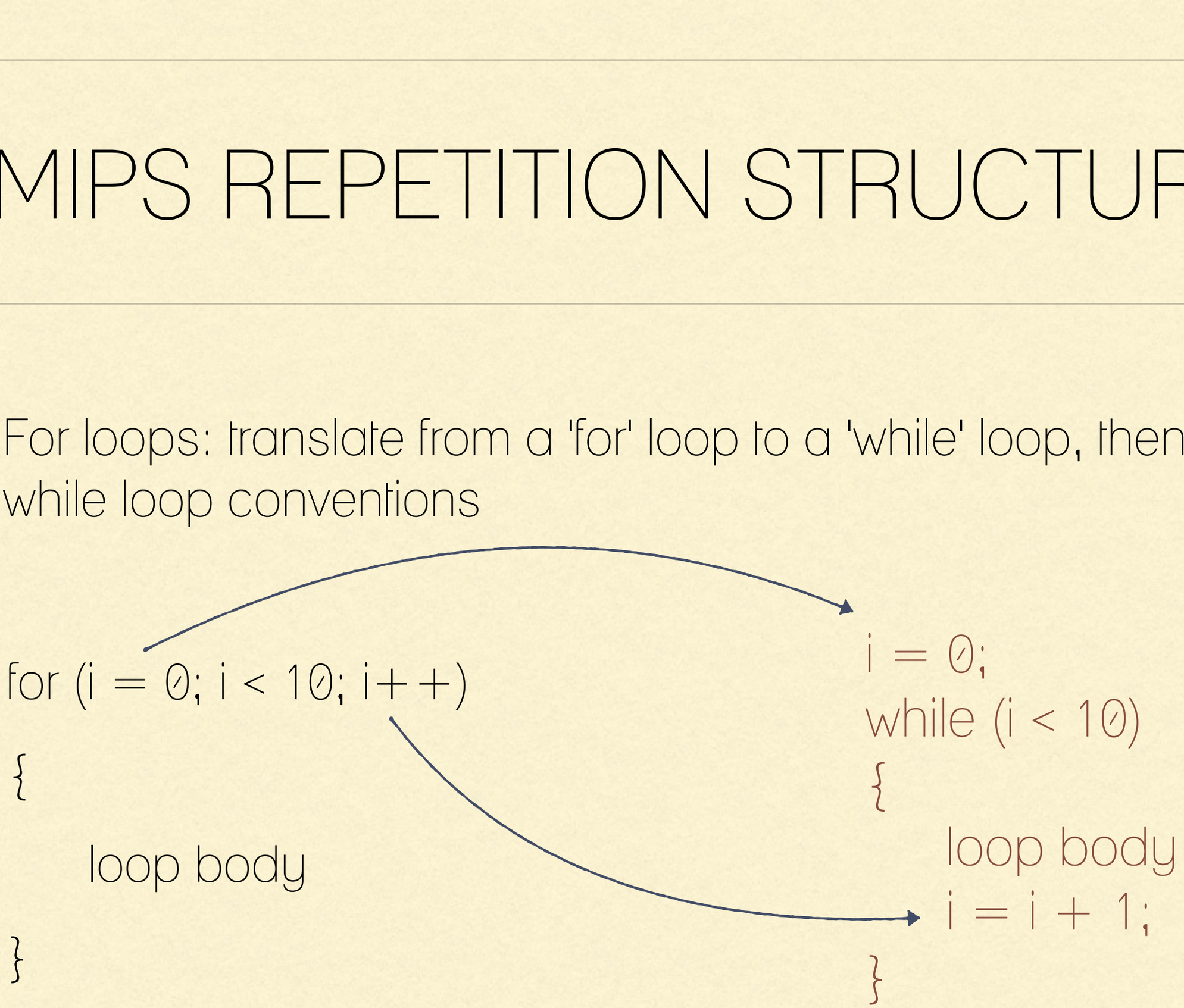
MIPS REPETITION STRUCTURES

- For loops: translate from a 'for' loop to a 'while' loop, then use while loop conventions

```
for (i = 0; i < 10; i++)
{
    loop body
}
```

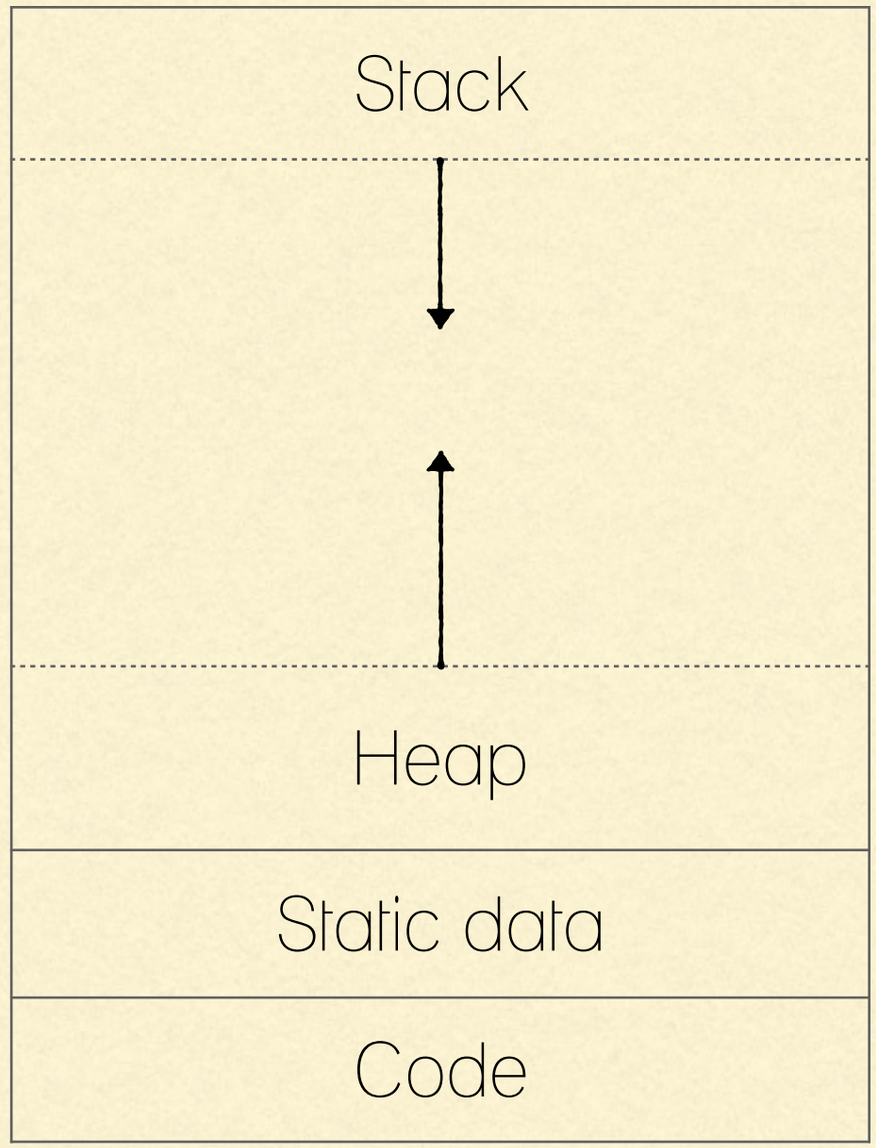
MIPS REPETITION STRUCTURES

- For loops: translate from a 'for' loop to a 'while' loop, then use while loop conventions



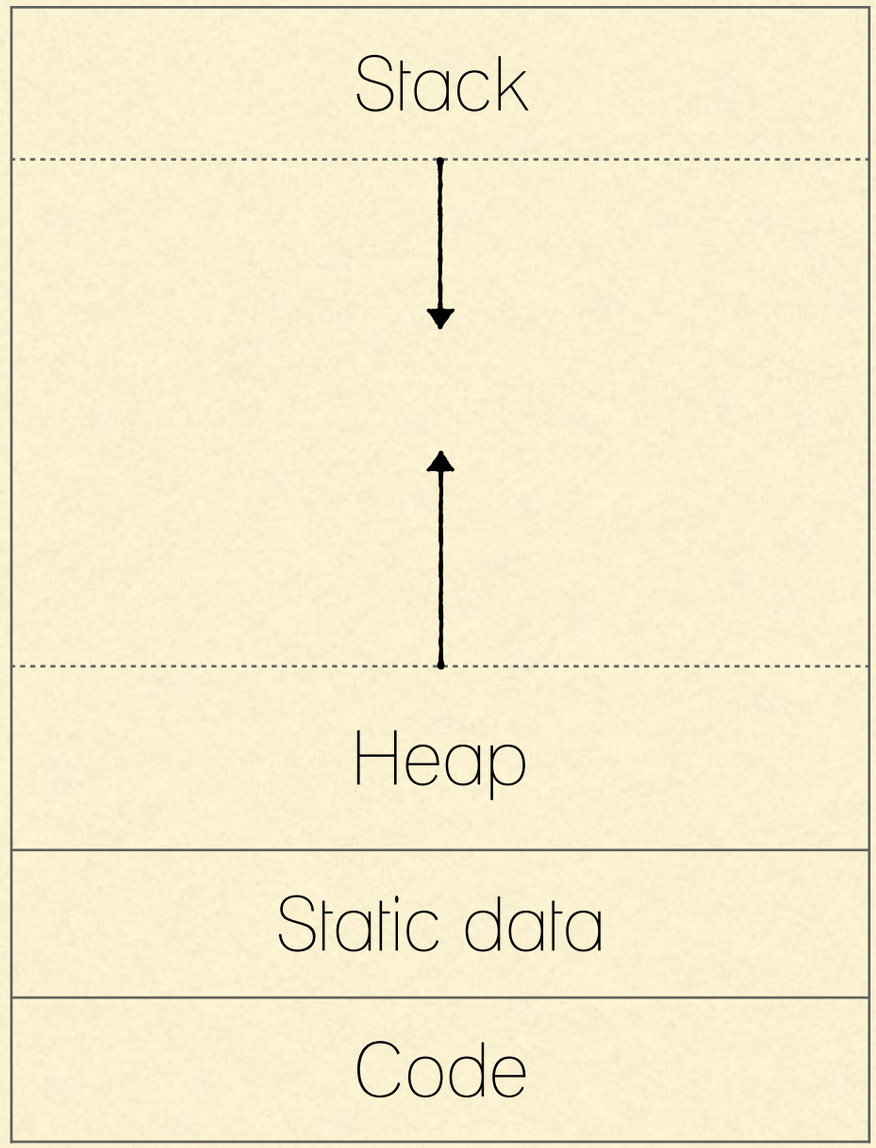
C/MIPS MEMORY MANAGEMENT

- Programs's address space contains 4 regions;
  - Stack; local variables inside functions, grows downward
  - Heap; space requested for dynamic data
  - Static data; variables declared outside functions, does not grow or shrink
  - Code; loaded when program starts — does not change



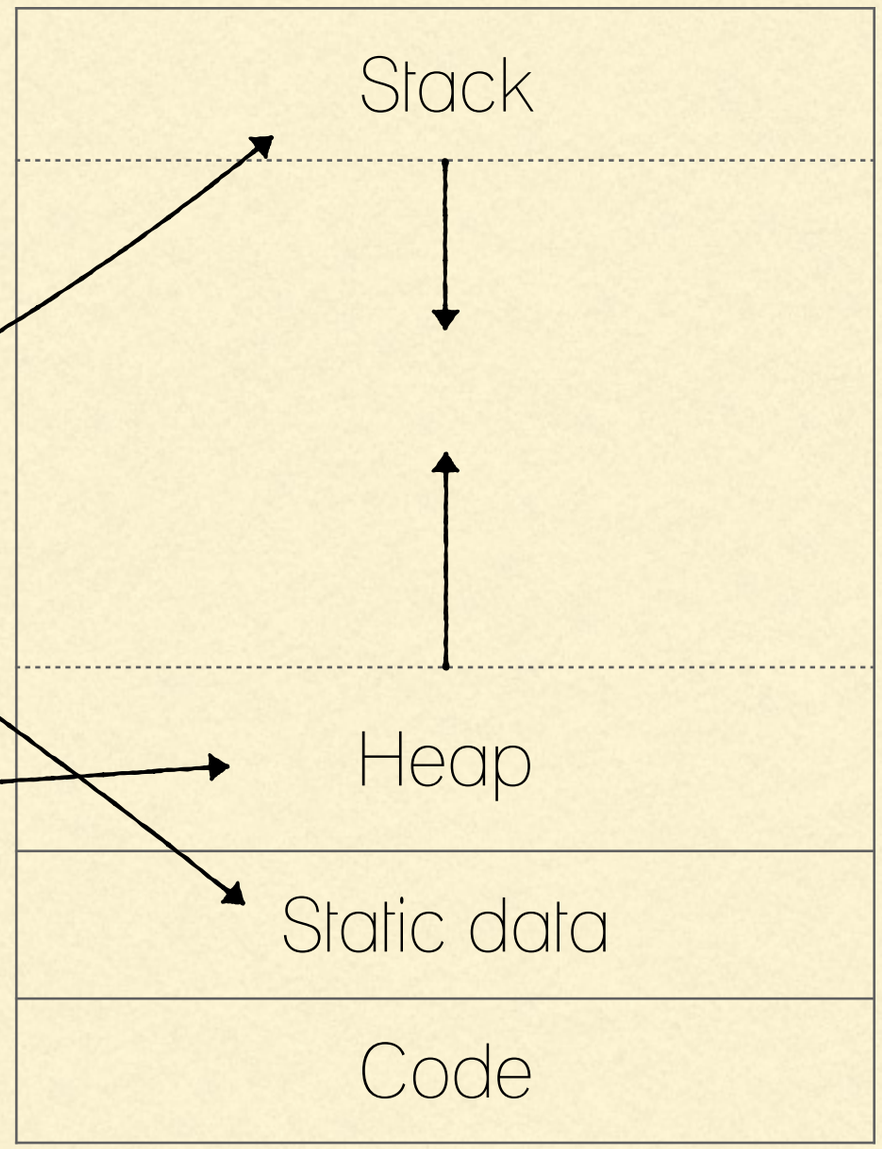
C/MIPS MEMORY MANAGEMENT

```
int arr_global[2];

int main()
{
    int arr_local[2];
    int *arr_dyn = malloc(sizeof(int) * 2);
}
```



C/MIPS MEMORY MANAGEMENT

```
int arr_global[2];

int main()
{
    int arr_local[2];
    int *arr_dyn = malloc(sizeof(int) * 2);
}
```



ALLOCATING SPACE IN STATIC DATA

```
int arr[] = [2, 5, 10];
void main()
{
    arr[2] = arr[0] + arr[1];
}
```

ALLOCATING SPACE IN STATIC DATA

```
int arr[] = [2, 5, 10];
void main()
{
    arr[2] = arr[0] + arr[1];
}
```

```
.data                       # Static data section
    arr: .word 2, 5, 10     # Space for 3 words
.text                       # Instructions section
```

ALLOCATING SPACE IN STATIC DATA

```
int arr[] = [2, 5, 10];
void main()
{
    arr[2] = arr[0] + arr[1];
}
```

```
.data                          # Static data section
    arr: .word 2, 5, 10        # Space for 3 words
.text                          # Instructions section
lw $t0, arr($zero)     # $t0 = arr[0]
```

ALLOCATING SPACE IN STATIC DATA

```
int arr[] = [2, 5, 10];
void main()
{
    arr[2] = arr[0] + arr[1];
}
```

```
.data                       # Static data section
    arr: .word 2, 5, 10     # Space for 3 words
.text                       # Instructions section
lw $t0, arr($zero)      # $t0 = arr[0]
addi $t1, $zero, 4      # $t1 = 4
lw $t2, arr($t1)        # $t2 = arr[1]
```

ALLOCATING SPACE IN STATIC DATA

```
int arr[] = [2, 5, 10];
void main()
{
    arr[2] = arr[0] + arr[1];
}
```

```
.data                       # Static data section
    arr: .word 2, 5, 10     # Space for 3 words
.text                       # Instructions section
lw $t0, arr($zero).         # $t0 = arr[0]
addi $t1, $zero, 4          # $t1 = 4
lw $t2, arr($t1)            # $t2 = arr[1]
addi $t1, $zero, 8          # $t1 = 8
add $t3, $t0, $t2           # $t3 = arr[0] + arr[1]
```

ALLOCATING SPACE IN STATIC DATA

```
int arr[] = [2, 5, 10];
void main()
{
    arr[2] = arr[0] + arr[1];
}
```

```
.data                       # Static data section
    arr: .word 2, 5, 10     # Space for 3 words
.text                       # Instructions section
lw $t0, arr($zero)      # $t0 = arr[0]
addi $t1, $zero, 4      # $t1 = 4
lw $t2, arr($t1)        # $t2 = arr[1]
addi $t1, $zero, 8      # $t1 = 8
add $t3, $t0, $t2       # $t3 = arr[0] + arr[1]
sw $t3, arr($t1)        # arr[2] = arr[0] + arr[1]
```

ALLOCATING SPACE IN THE STACK

```
void main()
{
    int arr[] = [2, 5, 6];
}
```

ALLOCATING SPACE IN THE STACK

```
void main()
{
    int arr[] = [2, 5, 6];
}
```

```
.data                           # Static data section
.text                           # Instruction section
    addi $sp, $sp, -12          # allocate 12 bytes of storage
    addi $t0, $zero, 2          # t0 = 2
    sw   $t0, 0($sp)            # arr[0] = 2
    addi $t0, $zero, 5          # t0 = 5
    sw   $t0, 4($sp)            # arr[1] = 5
    addi $t0, $zero, 6          # t0 = 6
    sw   $t0, 8($sp)            # arr[2] = 6
```