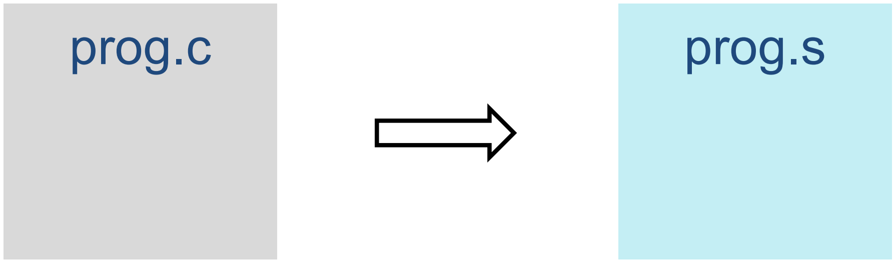
CMSC 216
Introduction to Computer Systems
Assembly MIPS 2

Translating C to assembly



- Static variables mapped to **.data** area
  - name of variable used as label in .data area
- Local variables mapped to **stack**
- While a variable is "**in-use**" (ie, in an expression that is being evaluated), it is mapped to a register
- String literals stored in **.data** area
- Code mapped to **.text** area
- Input-output calls (printf, scanf) mapped to **system calls**
- If-else and loops mapped using **labels** and **gotos** (branches)
- Functions mapped to functions (with the same name)
- Next: examples without local variables or functions

example_1

```
/* file: example_1.c */

int main() {
  printf("result: %d\n", 45 * 29);
  return 0;
}
```

Translate to assembly program:

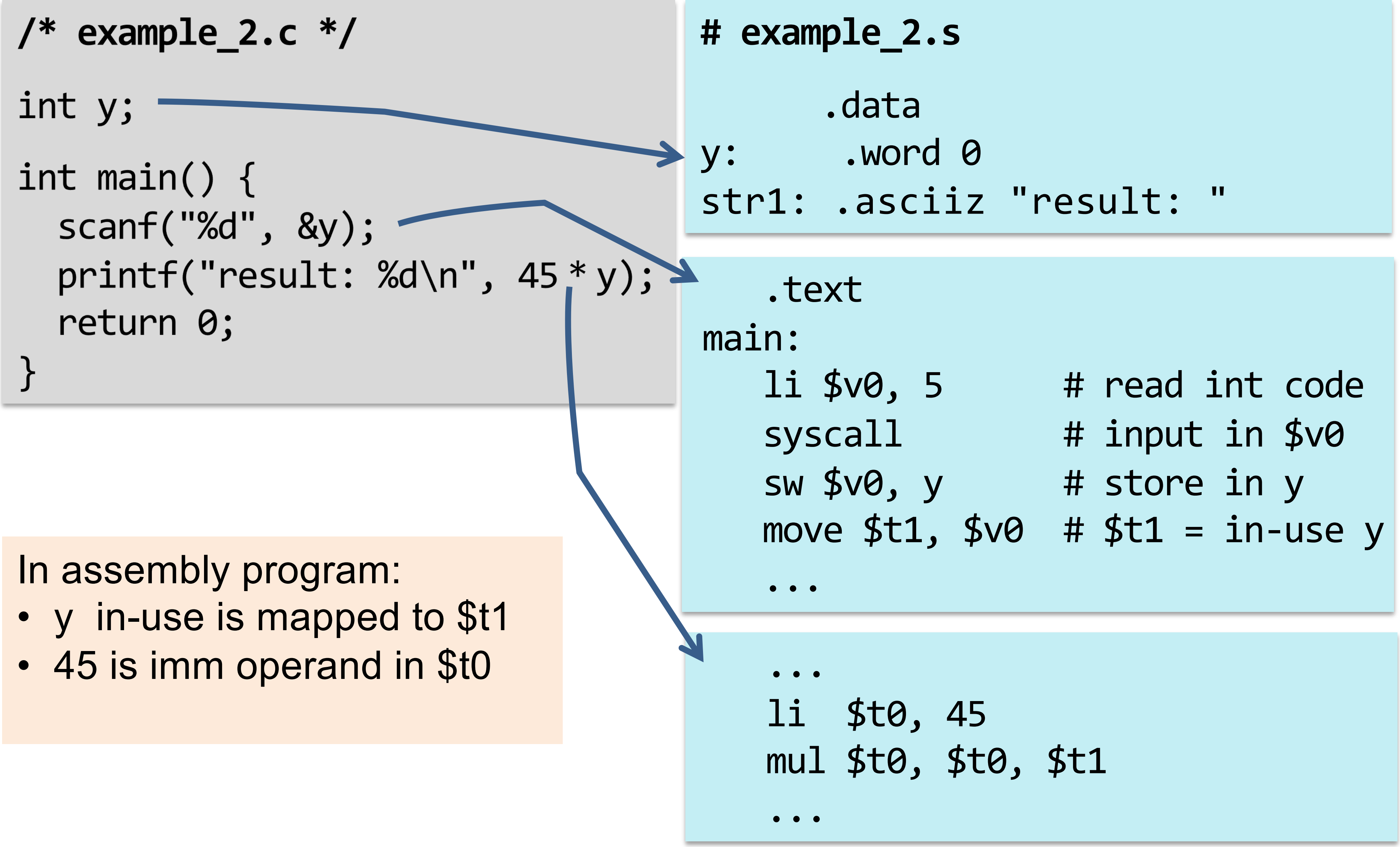
- 45 is immediate operand in reg $t0
- 29 is immediate operand in reg $t1

```
# file: example_1.s

        .data
str1:   .asciiz "result: "

        .text
main:   # pseudocode
        printf("%s", str1)
        $t0 = 45
        $t1 = 29
        $t1 = $t1 * $t0
        printf("%d", $t1)
        printf("%c", ‘\n’)
        return
```

example_1.s

```
       .data
str1:  .asciiz "result: "

       .text
main:  li     $v0, 4             # print_str code in $v0
       la     $a0, str1          # address of str1 in $a0
       syscall                   # printf("%s", str1)

       li     $t0, 45            # $t0 = 45 (first operand)
       li     $t1, 29            # $t1 = 29 (second operand)
       mul    $t1, $t0, $t1      # $t1 = 45 * 29 (result)

       li     $v0, 1             # print_int code in $v0
       move   $a0, $t1           # $a0 = result
       syscall                   # printf("%d", 45 * 29)

       li     $v0, 11            # print_char code in $v0
       li     $a0, 10            # ascii('\n') == 10
       syscall                   # printf("%c",'\n')

       jr     $ra                # return to kernel
```

example_2



```
/* example_2.c */

int y;

int main() {
  scanf("%d", &y);
  printf("result: %d\n", 45 * y);
  return 0;
}
```

```
# example_2.s

      .data
y:     .word 0
str1: .asciiz "result: "
```

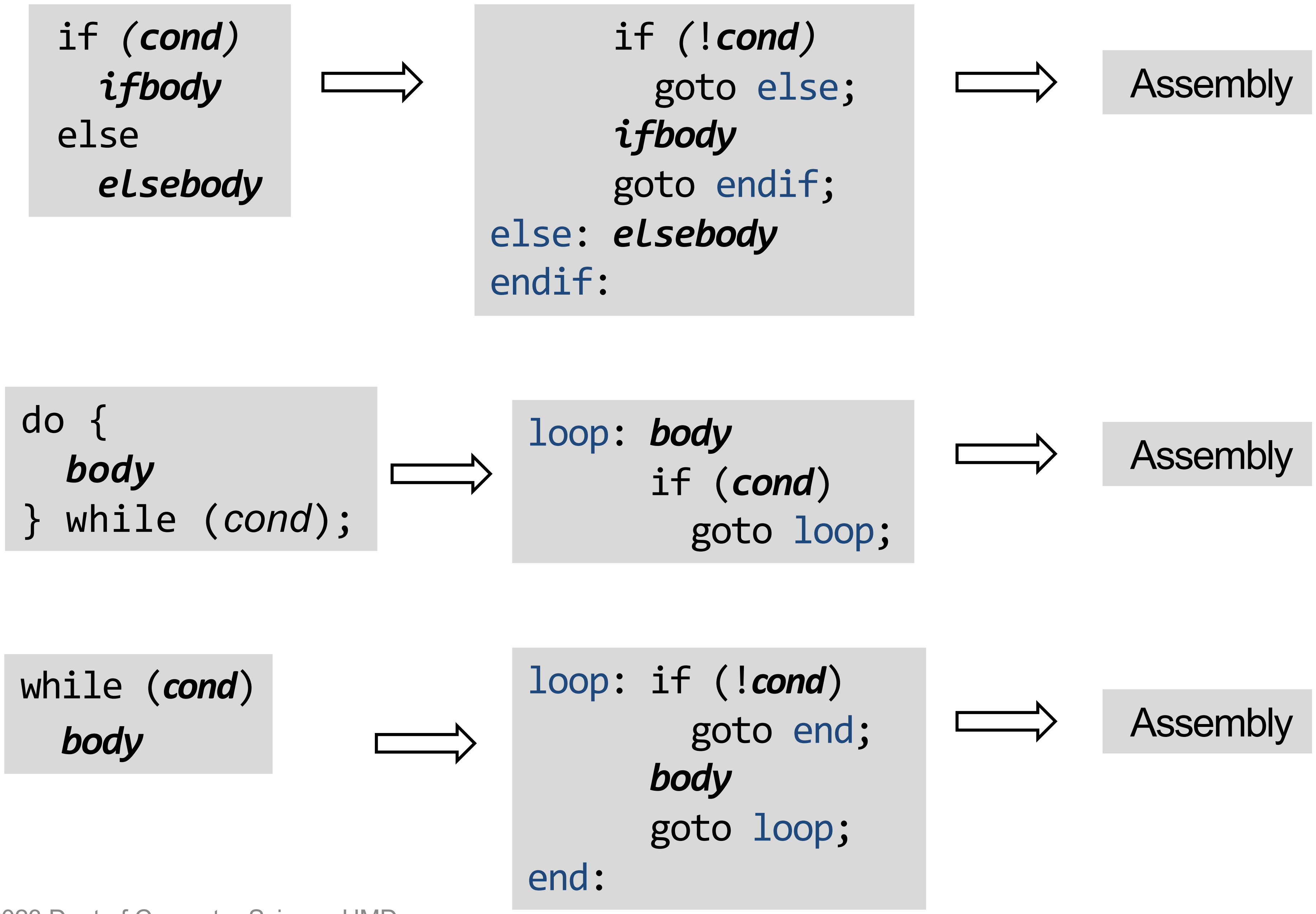
```
   .text
main:
   li $v0, 5       # read int code
   syscall         # input in $v0
   sw $v0, y       # store in y
   move $t1, $v0   # $t1 = in-use y
   ...
```

```
   ...
   li  $t0, 45
   mul $t0, $t0, $t1
   ...
```

In assembly program:

- y in-use is mapped to $t1
- 45 is imm operand in $t0

C if-else, loops → C labels, gotos → assembly



```
if (cond)
   ifbody
else
   elsebody
```

```
          if (!cond)
             goto else;
          ifbody
          goto endif;
else: elsebody
endif:
```

Assembly

```
do {
   body
} while (cond);
```

```
loop: body
      if (cond)
         goto loop;
```

Assembly

```
while (cond)
   body
```

```
loop: if (!cond)
         goto end;
      body
      goto loop;
end:
```

Assembly

example_3

```c
/* example_3.c */

int n;

int main() {
  printf("Enter int n: ");
  scanf("%d", &n);
  if (n % 2)
    printf("n odd\n");
  else
    printf("n even\n");
  return 0;
}
```

Assembly:

- n in-use is mapped to $t0

```
# example_3.s
      .data
str1: .asciiz "Enter int n: "
str2: .asciiz "n even\n"
str3: .asciiz "n odd\n"
n:    .word 0
```

```
   .text
   ...
   # $t0 == n
   rem  $t0, $t0, 2  # $t0 = n % 2
   beqz $t0, else    # if $t0 == 0
                     #   goto else
   la   $a0, str3    # $a0 = "n odd\n"
   j    endif
else:
   la   $a0, str2    # $a0 = "n even\n"
endif:
   li   $v0, 4       # print_str code
   syscall
```

example_4

```
/* example_4.c */

int n = 20;
int i = 1;
int sum = 0;

int main() {
  while (i <= n) {
    sum += i * i;
    i++;
  }
  printf("sum: %d\n", sum);
  return 0;
}
```

Assembly:

- n in-use in $t0
- i in-use in $t1
- sum in-use in $t2

```
# example_4.s
      .data
str1: .asciiz "sum: "
n:    .word 20
i:    .word 1
sum:  .word 0
```

```
  .text
  ...
  # $t0 = n, $t1 = i, $t2 = sum
loop:
  bgt $t1, $t0, endloop # if i > n
                        #  goto endloop
  mul  $t3, $t1, $t1    # $t3 = i*i
  add  $t2, $t2, $t3    # sum += i*i
  add  $t1, $t1, 1      # i++
  j    loop
endloop:
```