THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

COMPSCI 210

C Programming Language

Control Structures

Control Structures

- Conditional
  - making a decision about which code to execute, based on evaluated expression
  - `if`
  - `if-else`
  - `switch`
- Iteration
  - executing code multiple times, ending based on evaluated expression
  - `while`
  - `for`
  - `do-while`
- LC3 Code Translation
  - All control structures are done using Branch operation in LC3
  - `BR{nzp}`

If Statement

- *Condition is a C expression, which evaluates to*
  - *TRUE (non-zero) or*
  - *FALSE (zero).*
- *Action is a C statement,*
  - *which may be simple or compound (a block).*
- *Generate LC3 code from C code*
- C Code

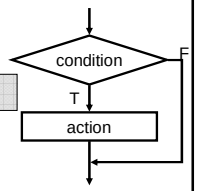
```
if (x == 2) y = 5;
```

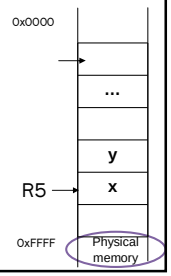
- LC3 Code:

```
        LDR  R0, R5, #0  ; load x into R0
        ADD  R0, R0, #-2 ; subtract 2
        BRnp NOT_TRUE    ; if non-zero, x is not 2
                         ; (R0 != 0)
        AND  R1, R1, #0  ; store 5 to y
        ADD  R1, R1, #5  ; (R0 == 0)
        STR  R1, R5, #-1
NOT_TRUE ...             ; next statement
```

if_else.asm

```
if (0 <= age && age <= 11)
  kids += 1;
```





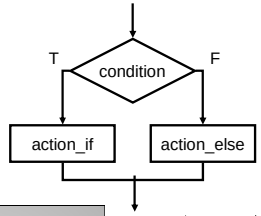
If-else Statement

- *Condition is a C expression, which evaluates to*
  - *TRUE (non-zero) or*
  - *FALSE (zero).*
- *Action is a double C statements,*
  - *True then do action 1*
  - *False then do action 2*
- *Generate LC3 code from C co*
- C Code to LC3 Code:

```
if (x) {
  y++;
  z--;
}
else {
  y--;
  z++;
}
```

if_else.asm

```
if (condition)
  action_if;
else
  action_else;
```

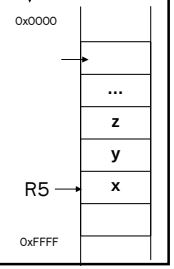


Compare x with 0

```
        LDR  R0, R5, #0
        BRz  ELSE
        LDR  R1, R5, #-1  ; load y
        ADD  R1, R1, #1   ; incr y
        STR  R1, R5, #-1
        LDR  R1, R5, #-2  ; load z
        ADD  R1, R1, #-1  ; decr z
        STR  R1, R5, #-2
        BR   DONE  ; skip else code
ELSE    LDR  R1, R5, #-1  ; decr y
        ADD  R1, R1, #-1
        STR  R1, R5, #-1
        LDR  R1, R5, #-2  ; incr z
        ADD  R1, R1, #1
        STR  R1, R5, #-2
DONE    ...  ; next statement
```

x != 0

x == 0



If-else statement (cont)

- Else is always associated with *closest* unassociated if.

```
if (x != 10)
  if (y > 3)
    z = z / 2;
  else
    z = z * 2;
```

==

```
if (x != 10) {
  if (y > 3)
    z = z / 2;
  else
    z = z * 2;
}
```

!=

```
if (x != 10) {
  if (y > 3)
    z = z / 2;
}
else
    z = z * 2;
```

- Good practice to always use { and } with if-else statement.
  - To reduce bugs
  - Improve readability
- Chaining If's and Else's
  - The conditions are evaluated in order until one that evaluates to true is found. If none of them are true then the else block is executed.
  - The final else is optional

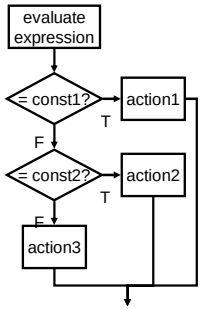
Test 13.4.c

```
if (month == 4 || month == 6 || month == 9 || month == 11)
  printf("Month has 30 days.\n");
else if (month == 1 || month == 3 || month == 5 || month == 7 ||
         month == 8 || month == 10 || month == 12)
  printf("Month has 31 days.\n");
else if (month == 2)
  printf("Month has 28 or 29 days.\n");
else
  printf("Don't know that month.\n");
```

Switch

```
switch (expression) {
case const1:
  action1; break;
case const2:
  action2; break;
default:
  action3;
}
```

- Alternative to long if-else chain.
- If break is not used, then case "falls through" to the next.
- No two case statements can have the same value.
- Value must be an an integer value: type long, int, or char
- The **default** statement is optional



Month_switch.c

Enter a month and program print out different sentence for different month

Use break to get out of the switch, if no break next action will take place after

```
switch (month) {
  case 4:
  case 6:
  case 9:
  case 11:
    printf("Month has 30 days.\n");
    break;
  case 1:
  case 3:
  /* some cases omitted for brevity...*/
    printf("Month has 31 days.\n");
    break;
  case 2:
    printf("Month has 28 or 29 days.\n");
    break;
  default:
    printf("Don't know that month.\n");
}
```

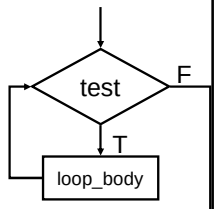
Writing Loops

- Loops
  - Repeated execution of one or more statements until a terminating condition occurs
  - Pre-test and post-test loops
- Types of loops:
  - Pre-test loops
    - while
    - for
  - Post-test loop
    - do while

While loop

- *Executes loop body as long as test evaluates to TRUE (non-zero).*
  - *Note: Test is evaluated before executing loop body*
  - *When false, the loop body is not executed at all*
  - *Do..while loop is an alternative*
- *Generate LC3 code from C code*

```
while (test)
  loop_body;
```




```
x = 0;
while (x < 10) {
  printf("%d ", x);
  x = x + 1;
}
```

X is stored at address pointed by R5

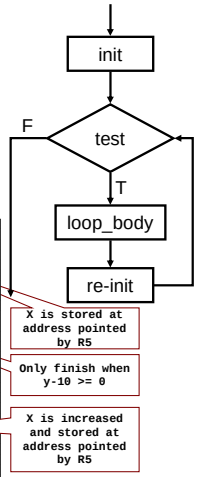
Only finish when x = 0

X is added by 1 and stored at address pointed by R5

```
        AND  R0, R0, #0
        STR  R0, R5, #0 ; x = 0
        ; test
LOOP    LDR  R0, R5, #0 ; load x
        ADD  R0, R0, #-10
        BRzp DONE
        ; loop body
        LDR  R0, R5, #0 ; load x
        ...
        <printf>
        ...
        ADD  R0, R0, #1 ; incr x
        STR  R0, R5, #0
        BR   LOOP   ; test again
DONE    ; next statement
```

For loop

```
for (init; end-test; re-init)
  statement
```



- Executes loop body as long as test evaluates to TRUE (non-zero).
  - Probably the most used loop statement
  - Initialization and re-initialization code included in loop statement.
  - Note: Test is evaluated before executing loop body.

```
for (i = 0; i <= 10; i ++)
  printf("%d ", i);
```

```
Output
0 1 2 3 4 5 6 7 8 9 10
```

```
letter = 'a';
for (c = 0; c < 26; c++)
  printf("%c ", letter+c);
```

```
Output
a b c d e f g h i j k l m n o p q r s
```

- C code to LC3 translation:

```
for (i = 0; i < 10; i++)
  printf("%d ", i);
```

```
; init
        AND  R0, R0, #0
        STR  R0, R5, #0  ; i = 0
        ; test
LOOP    LDR  R0, R5, #0 ; load i
        ADD  R0, R0, #-10
        BRzp DONE

        ; loop body
        LDR  R0, R5, #0 ; load i
        ...
        <printf>
        ...
        ; re-init
        ADD  R0, R0, #1 ; incr i
        STR  R0, R5, #0
        BR   LOOP   ; test again
DONE    ; next statement
```

X is stored at address pointed by R5

Only finish when y-10 >= 0

X is increased and stored at address pointed by R5

Nested Loops

Star.c

- Loop body can (of course) be another loop.

```
/* print a multiplication table */
for (mp1 = 0; mp1 < 10; mp1++) {
  for (mp2 = 0; mp2 < 10; mp2++) {
    printf("%d\t", mp1*mp2);
  }
  printf("\n");
}
```

Braces aren't necessary, but they make the code easier to read.

- The test for the inner loop depends on the counter variable of the outer loop.

```
for(i = 0; i < row; i ++) {
    for (j = 0; j <= i; j++) {
      printf("*");
    }
    printf("\n");
}
```

Output for row=8

```
*
**
***
****
*****
******
```

```
for (outer = 1; outer <= input; outer++) {
  for (inner = 0; inner < outer; inner++) {
    sum += inner;
  }
}
```

Output for Input = 4

```
Input an integer: 4
The result is 10
```

For vs. While

- In general:
  - They are equivalent
  - Either kind of loop can be expressed as the other, so it's really a matter of style and readability.
  - These 2 loops are equivalent and are endless loop

```
while (1)
  printf("*");
```

```
for (;;)
  printf("*");
```

- For loop is preferred for counter-based loops.
  - Explicit counter variable
  - Easy to see how counter is modified each loop

- While loop is preferred for sentinel-based loops.
  - Test checks for sentinel value.

```
for(;;){
  if(conditionMet)
    break; //code
}
```

```
while(!conditionMet){
  //code
}
```