Control Structures

Essential control structures for all imperative programming languages are:

- Conditionals: if, case, switch, ...
- Open loops: while, repeat, ...
- Bound loops: for, foreach, forall, ...
- Procedures and functions (already covered)

References for this chapter

[Patterson17]
David A. Patterson & John L. Hennessy
Computer Organization and Design - The Hardware/Software Interface
Chapter 2: "Instructions: Language of the Computer"
Morgan Kaufmann, 2017

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```
if (register1 == register2) {
    register3 = 1;
} else {
    register3 = 0;
}
```

How do those control structures in programming languages translate into Assembly?

Control Structures

It seems there are three distinguishable code sections and one status flag condition.

- Instructions to generate status flags
- Code for True (then)
- Code for False (else)

Can we form a general pattern for this?

Control Structures

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```
cmp r1, r2 ; 1. Instructions to generate status flags
beq then ; 2. Branch depending on the status flags
mov r3, #0 ; 4. Instructions for the then branch
b end_if
test r3, #1 ; 3. Instructions for the then branch

register3 = (if register1 == register2 then 1 else 0);
```

end_if:

We might need a lot of those, hence the labels need to be unique to each if-else block.

Control Structures

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```
if (register1 = Register_2)
    Register_3 := 1;
else
    Register_3 := 0;
end if;
```

Control Structures

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```
if (register1 == register2) {
    register3 = 1;
} else {
    register3 = 0;
}
```

It seems there are three distinguishable code sections and one status flag condition.

Can we form a general pattern for this?

Control Structures

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```
if (condition_code)
    then_code
else
    else_code
end if;
```

Control Structures

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```
if (condition_code)
    then_code
else
    else_code
end if;
```

Control Structures

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```
if (condition_code)
    then_code
else
    else_code
end if;
```

We might need a lot of those, hence the labels need to be unique to each if-else block.
Control Structures

Conditionals – IF-ELSE

Assuming the values have already been transferred from memory into registers:

```assembly
if Register_1 = Register_2 then
    Register_3 = 1
else
    Register_3 = 0
end if;
```

We can now write:

```assembly
register_3
```

Condition

```assembly
mov
```

Computational complexity #13

Condition

```assembly
mov
```

Loops – FOR

Assuming the values have already been transferred from memory into registers:

```assembly
for Register_1 in 1..100 do
    Register_3 := Register_3 + Register_1;
end do;
```

We can find the index, the start and end values and the body code.

```assembly
We can form a general pattern for this?
```
Control Structures

Loops – FOR

for Register_1 in 1..100 loop
  Register_1 := Register_1 ** 2;
end loop;

Loops – WHILE

while Register_1 < 100 loop
  Register_1 := Register_1 ** 2;
end loop;

Computational complexity \( \Theta(n) \)

We can now write:

\[
\text{macro while }
 rst, r1, \text{ while_condition, body}
\]

\[
\text{while (r1 < 100) }
\{ \text{body} \}
\]

... try to rewrite our power functions from the previous chapter with the macros you have now.

We can now write:

\[
\text{while (cmp r1, #100, lt) }
\{ \text{body} \}
\]
Control Structures

Loops – WHILE

while Register_1 < 100
    Register_1 := Register_1 ** 2
end loop;

while Register_1 < 100
    Register_1 := Register_1 ** 2
end loop;

Control Structures

Conditionals – CASE (indexed)

type Colour is (Red, Green, Blue);

case Register_1 is
    when Red   => Register_3 := Register_2;
    when Green => Register_4 := Register_2;
    when Blue  => Register_5 := Register_2;
end case;

A table based branching implementation of:

while (register1 < 100)
{
    register1 = register1 ** 2
    while (register1 < 100)
    loop
        case register1 is
            end loop;
       ...
    }

Computational complexity: Undefined

The parts which are actually producing code are highlighted.
... recursive parts are following on the next page ... hold on to something!
Control Structures

Conditionals – CASE (indexed)

```haskell
r0 :: Int -> Int -> Int -> Int
```

This is again recursive to handle the variable number of cases:

```haskell
cmp | r1 < r2 = r2
    | r1 > r2 = r1
```  

```haskell
bgt case_a
```

```haskell
r1 == r2 =
```

```haskell
bgt case_b
```

```haskell
otherwise = error "How did I get here?"
```

```haskell
```

Keep in mind: Macro programming is prone to bad design. The result is a mix which is then translated by the assembler into machine code:

```haskell
while "cmp r1, el00", li, "add r1, r1"
```

```haskell
for r1, 1, 100 "add r1, r1"
```

The parts which are actually producing code are highlighted. ... and we still need to generate the list of guards, followed by the list of code sections.
You can form all common sequential control structures (or generate them via macros if you wish) (including function calls)

Summary

Control Structures

- Assembler Macros
  - Local labels
  - Recursive macros
- Control Structures in machine code
  - IF
  - WHILE
  - FOR
  - CASEs