Structured Programming – Control Structures

Computer Engineering 1

CT Team: A. Gieriet, J. Gruber, B. Koch, M. Loeser, M. Meli,
M. Rosenthal, M. Ostertag, A. Rüst, J. Scheier, T. Welti

Motivation

Spaghetti code

From Wikipedia, the free encyclopedia.

Spaghetti code is a pejorative term for code with a complex and tangled control structure, especially one using many GOTOs, exceptions, or other "unstructured" branching constructs. It is named after spaghetti because a diagram of program flow tends to look like that. Nowadays it is preferable to use so-called structured programming.

Also called *kangaroo code* because such code has so many jumps in it.

-65-

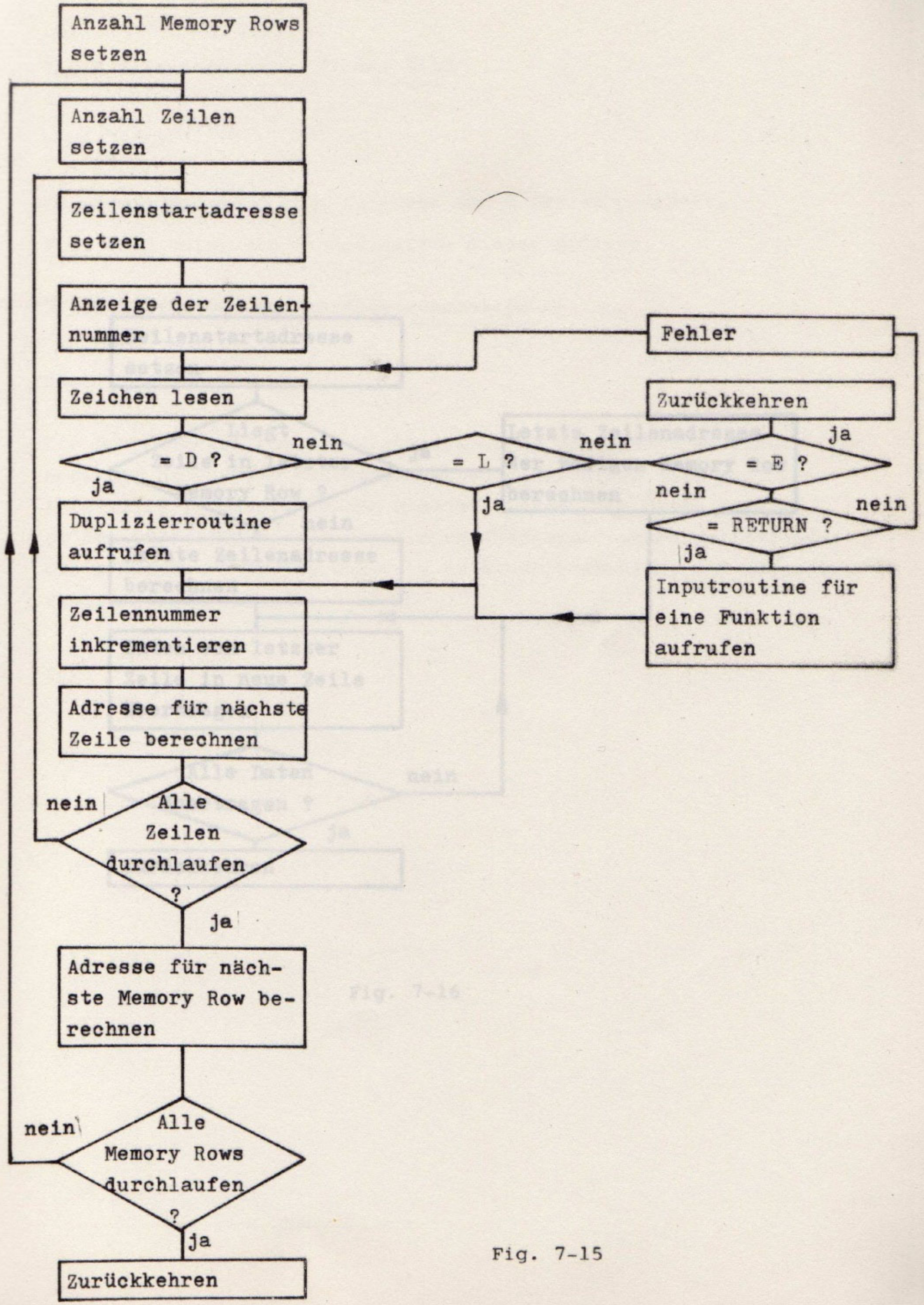

Anzahl Memory Rows setzen
Anzahl Zeilen setzen
Zeilenstartadresse setzen
Anzeige der Zeilennummer
Zeichen lesen
= D ?
nein
ja
= L ?
nein
ja
= E ?
ja
nein
= RETURN ?
nein
ja
Fehler
Zurückkehren
Inputroutine für eine Funktion aufrufen
Duplizierroutine aufrufen
Zeilennummer inkrementieren
Adresse für nächste Zeile berechnen
nein
Alle Zeilen durchlaufen ?
ja
Adresse für nächste Memory Row berechnen
nein
Alle Memory Rows durchlaufen ?
ja
Zurückkehren

Fig. 7-15

Agenda

- **Structured Programming**
- **Selection**
  - if – then - else
- **Loops**
  - Do – While
  - While
  - For
- **Switch Statements**

Learning Objectives

At the end of this lesson you will be able

- to explain the basic concepts of structured programming
- to enumerate and explain the basic elements of a structogram
- to comprehend how a C-compiler implements control structures in assembly language
    - if-then-else
    - do-while loops
    - while loops
    - for loops
    - switch statements
- to program basic structograms in assembly language

Why Structured Programming ?

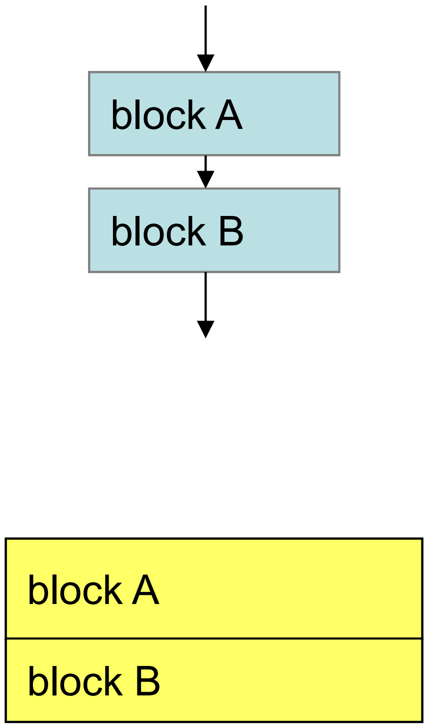
- **Rules for the structure of a program**
  - Patterns for control structures
    - Sequence
    - Selection if - then - else
    - Iteration / Loop for, while, do - while
  - Compilers generate code-blocks based on these patterns

- **Supports program development**
  - Clarity
  - Documentation
  - Maintenance
  - Allows to program on a higher level of abstraction

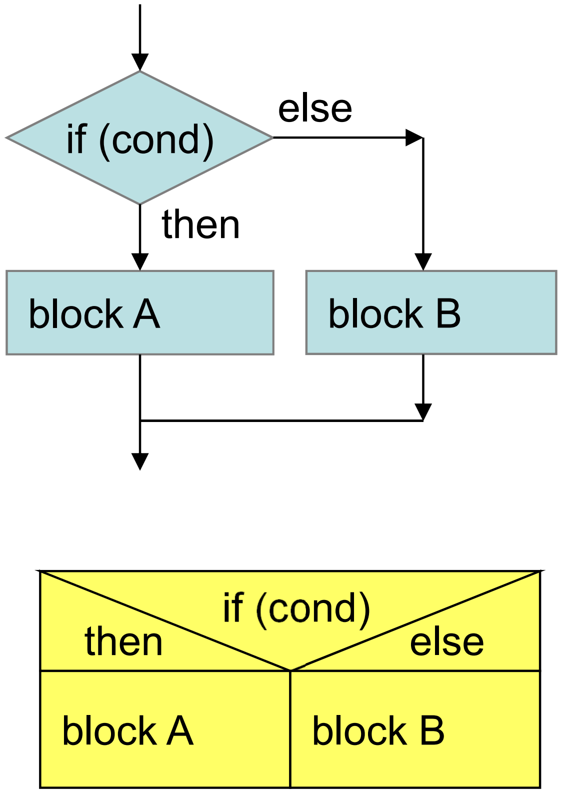
Structured Programming

- **Program flow can be represented with three elements**

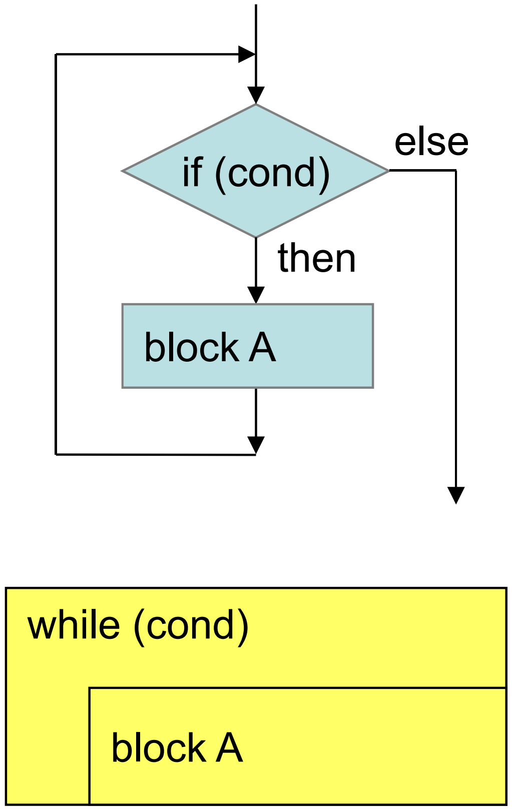
Sequence



Selection



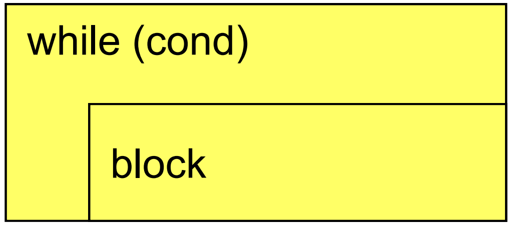
Iteration / loop



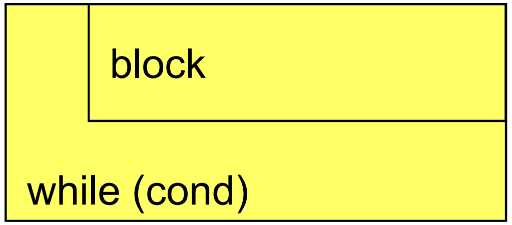
Further Structograms

- **Iteration**

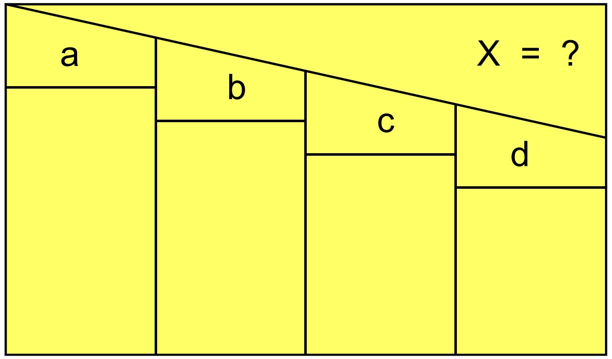
pre-test loop



post-test loop



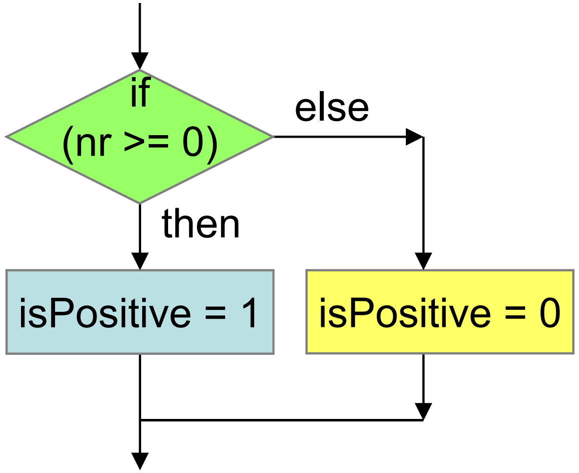
- **Switch statement (case)**

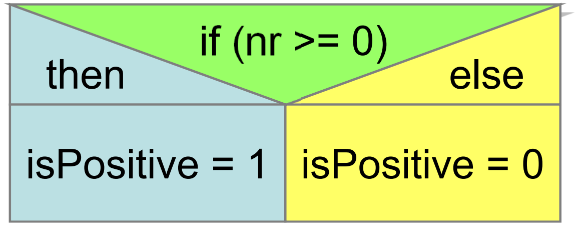


Selection

- **if(...) – then - else**

```
int32_t nr, isPositive;
...
if (nr >= 0) {
    isPositive = 1;
}
else {
    isPositive = 0;
}
```





Selection: if – then – else

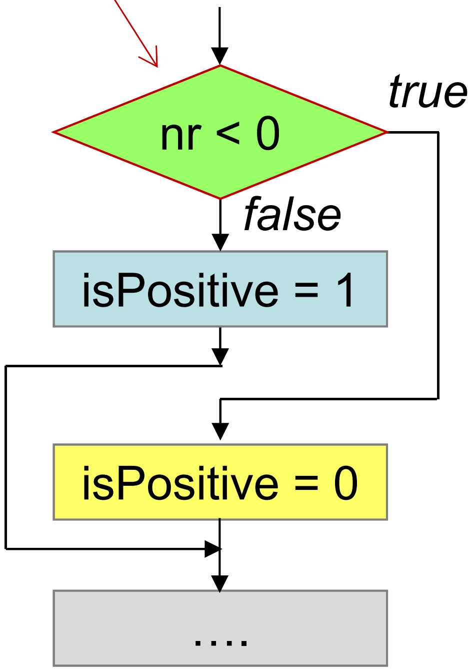
- Compiler translates *selection* into assembly code
  - uses conditional and unconditional jumps

```
int32_t nr;
int32_t isPositive;
...
if (nr >= 0) {
    isPositive = 1;
}
else {
    isPositive = 0;
}
```

Assume: nr in R1
isPositive in R2

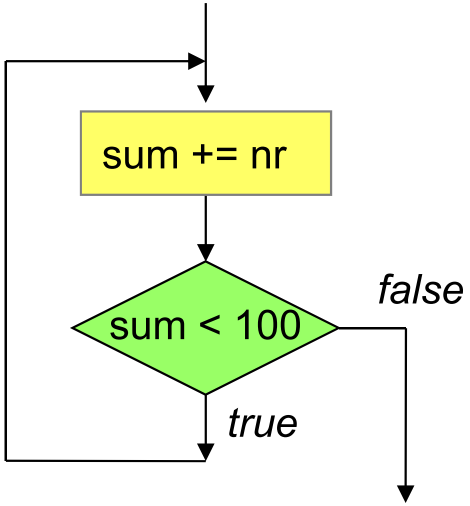
inverted condition

```
         CMP     R1,#0x00
         BLT     else
         MOVS    R2,#1
         B       end_if
else
         MOVS     R2,#0
end_if
         ....
```



Loops: Do-While Loops

```
int32_t nr;
int32_t sum;
...
sum = 0;
do {
    sum += nr;
} while (sum < 100);
```



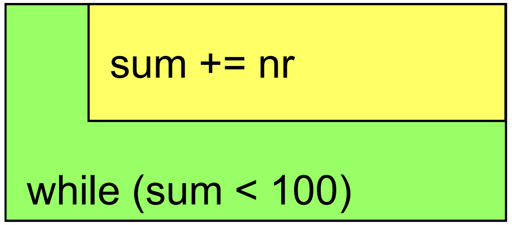
sum += nr

sum < 100

false

true

post-test loop



sum += nr

while (sum < 100)

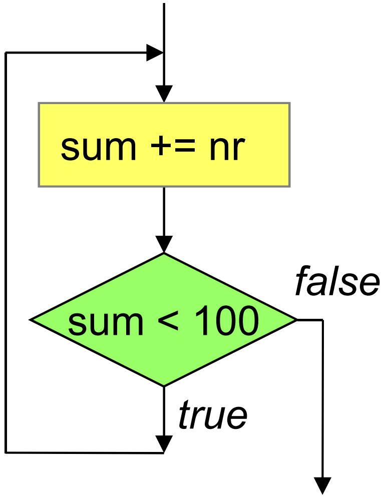
Loops: Do-While Loops

- Compiler translates *post-test loop* to assembly code

```
int32_t nr;
int32_t sum;
...
sum = 0;
do {
    sum += nr;
} while (sum < 100);
```

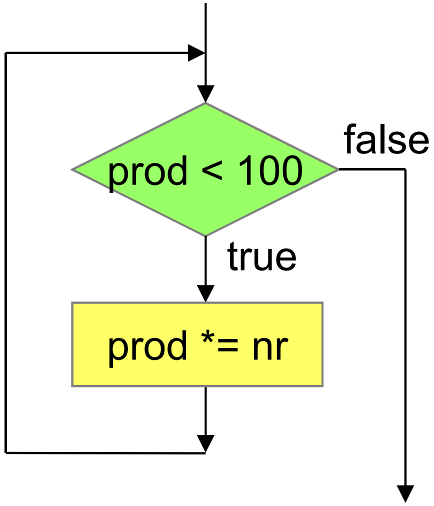
Assume: nr in R1
sum in R2

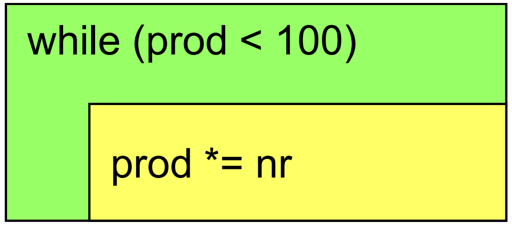
```
        MOVS   R2,#0
loop    ADDS   R2,R2,R1
        CMP    R2,#100
        BLT    loop
        ....
```



Loops: While Loops

```
int32_t nr;
int32_t prod;
...
prod = 1;
while (prod < 100) {
    prod *= nr;
}
```





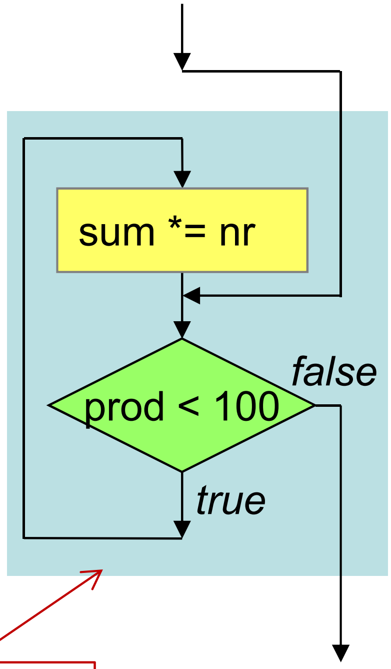
Loops: While Loops

- Compiler translates *pre-test loop* to assembly code
  - Re-using structure of do-while (pre-test loop)

```
int32_t nr;
int32_t prod;
...
prod = 1;
while (prod < 100) {
    prod *= nr;
}
```

Assume: nr in R1
prod in R2

```
        MOVS    R2,#1
        B       test
loop
        MULS    R2,R1,R2
test
        CMP     R2,#100
        BLT     loop
        ...
```



re-use do-while

Loops: For Loops

- For Loops are converted into While Loops
  - break/continue statements require special treatment

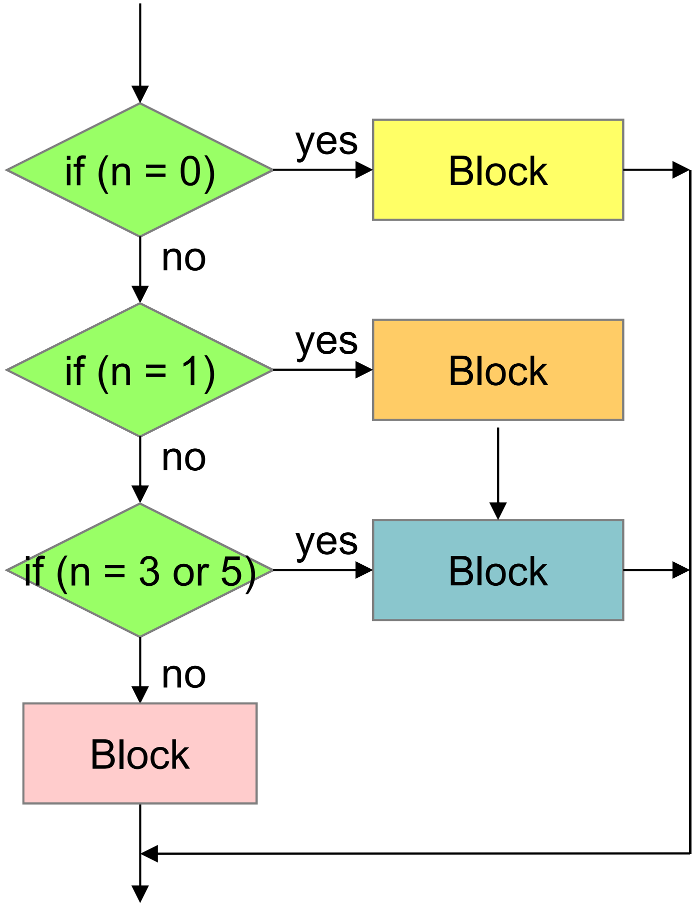
```
for (init-expr; test-expr; update-expr)
     body-block
```

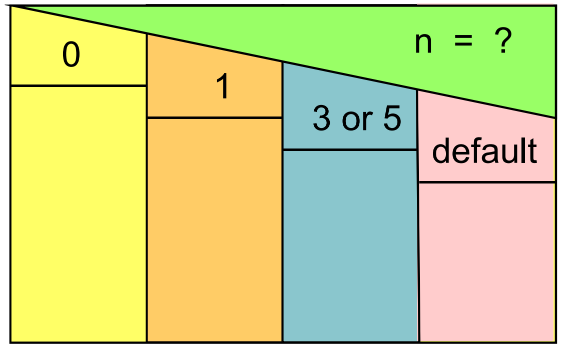
```
init-expr;
while (test-expr) {
     body-block
     update-expr;
}
```

Switch Statements

```
uint32_t result, n;

switch (n) {
case 0:
    result += 17;
    break;
case 1:
    result += 13;
    //fall through
case 3: case 5:
    result += 37;
    break;
default:
    result = 0;
}
```





Structogram without fall-through

Switch Statements

- **Jump Table**

```
uint32_t result, n;
switch (n) {
case 0:
      result += 17;
      break;
case 1:
      result += 13;
      //fall through
case 3: case 5:
      result += 37;
      break;
default:
      result = 0;
}
```

```
NR_CASES      EQU   6
case_switch   CMP   R1, #NR_CASES
              BHS   case_default
              LSLS  R1, #2    ; * 4
              LDR   R7, =jump_table
              LDR   R7, [R7, R1]
              BX    R7
case_0        ADDS  R2, R2, #17
              B     end_sw_case
case_1        ADDS  R2, R2, #13
case_3_5      ADDS  R2, R2, #37
              B     end_sw_case
case_default  MOVS  R2,#0
end_sw_case   ...
```

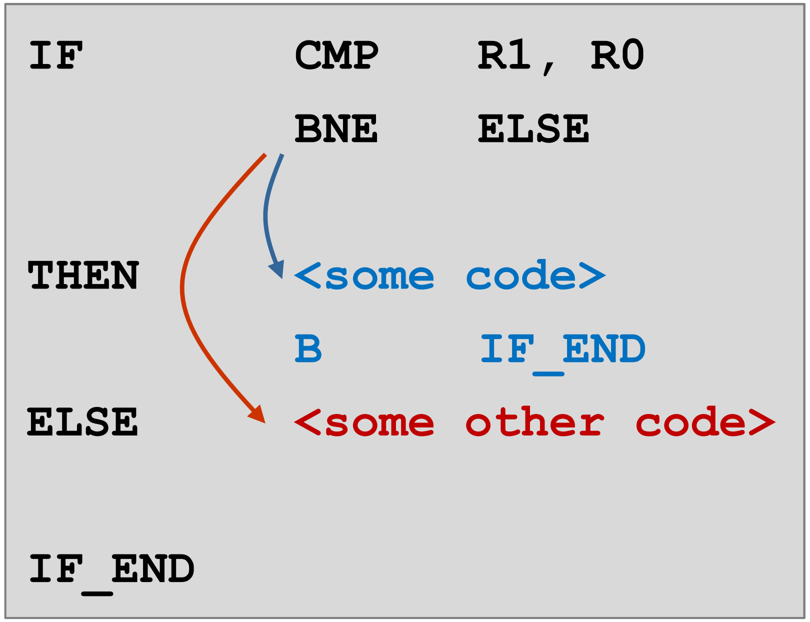
```
jump_table    DCD   case_0
              DCD   case_1
              DCD   case_default
              DCD   case_3_5
              DCD   case_default
              DCD   case_3_5
```

Assume: n in R1
result in R2

Limitations of Conditional Branches

- **Limited range of -256..254 Bytes**
  - Example

Unconditional branch has longer range than conditional branch

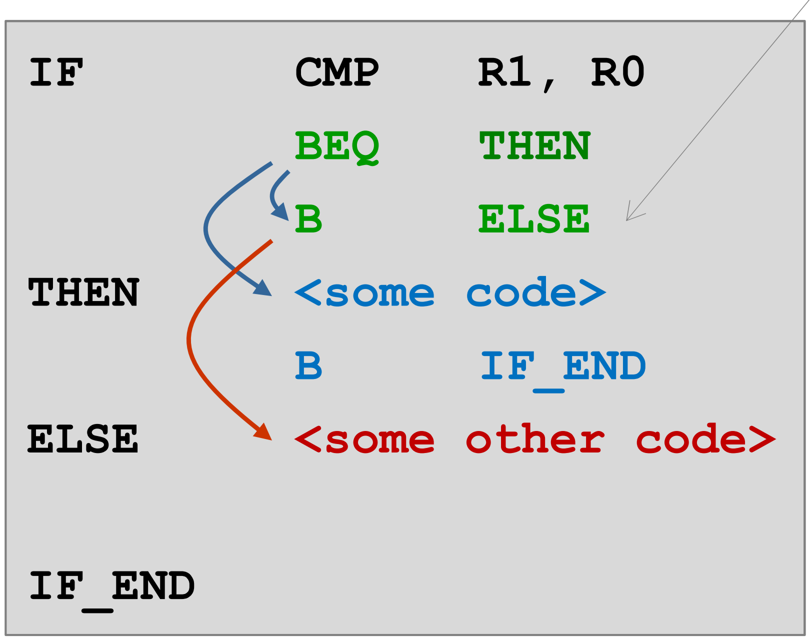


```
IF          CMP     R1, R0
            BNE     ELSE

THEN        <some code>
            B       IF_END
ELSE        <some other code>


IF_END
```

Simple code for the case when <some code> is short



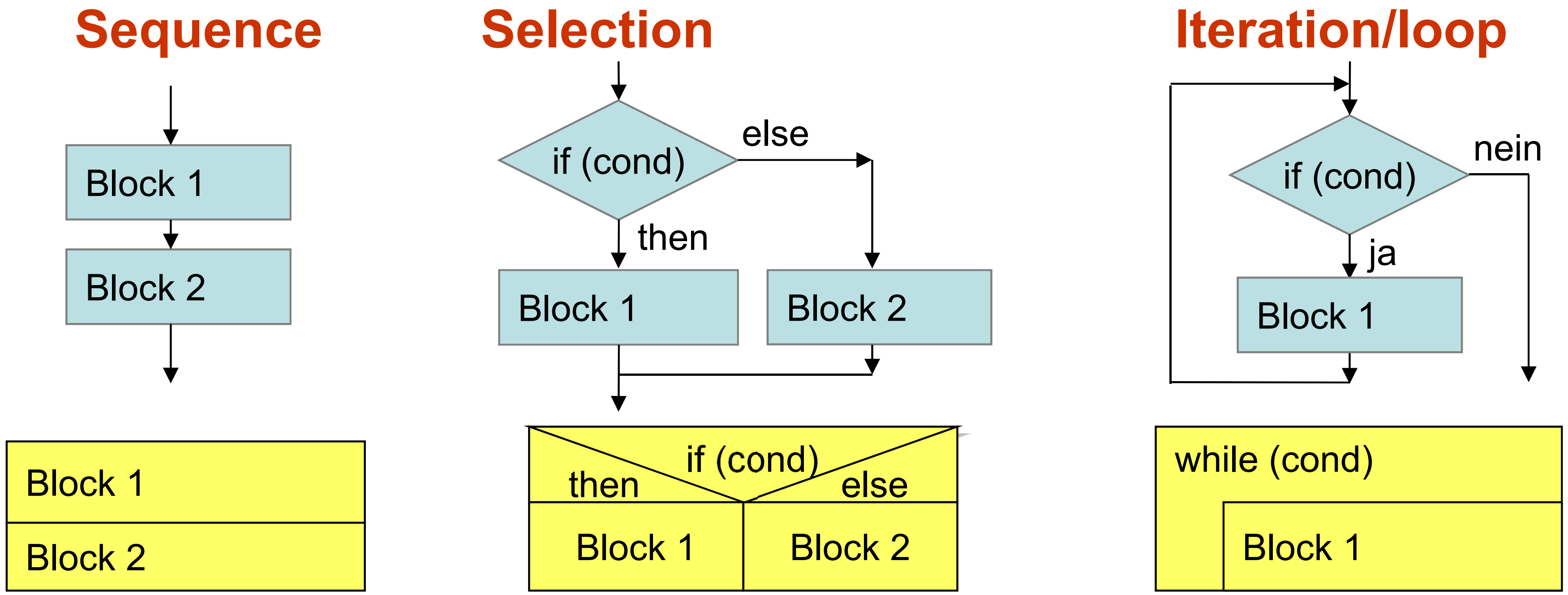
```
IF          CMP     R1, R0
            BEQ     THEN
            B       ELSE
THEN        <some code>
            B       IF_END
ELSE        <some other code>


IF_END
```

Code requires additional branch in case when <some code> is too long

Conclusion

- Program flow can be represented with three elements



- High level programming language provides these control structures
- Compiler translates control structures to assembly using conditional and unconditional jumps