Lecture 3
Conditions and branches

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

2018

Controlling the order of execution of instructions

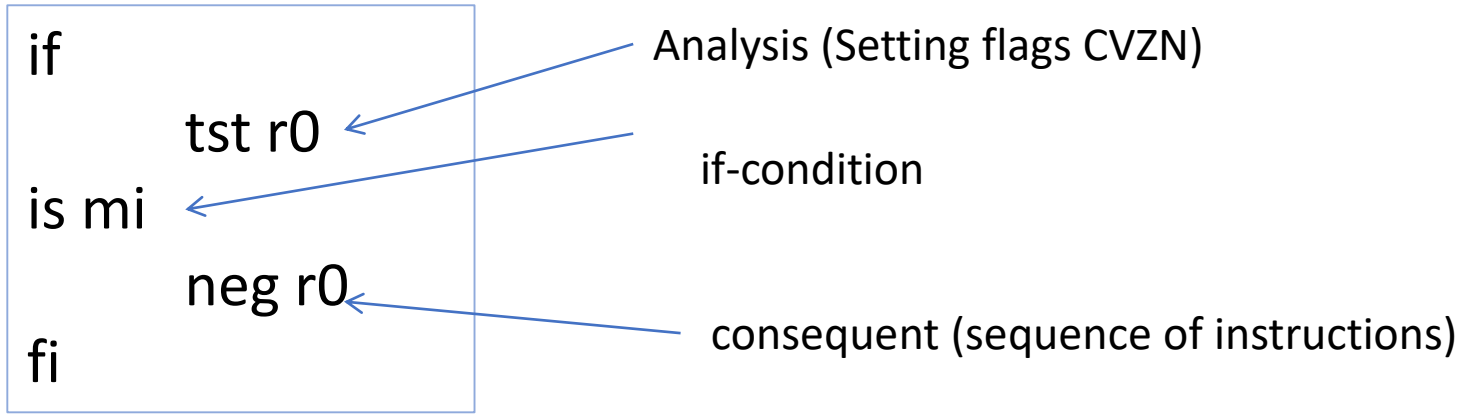
- Управляющие инструкции – изменяют последовательность выполнения программы.

- Управляющая конструкция — это не отдельная инструкция, а группа логически связанных инструкций, которые вместе взятые могут использоваться для управления тем, какая последовательность(и) инструкций выполняется и сколько раз.
- Управляющие конструкции взаимодействуют с остальными инструкциями исключительно путем проверки флагов CVZN, составляющих регистр PS. Флаги CVZN являются собственностью ALU и могут быть изменены только операцией ALU. Остальные операции, включая операции управления (а также те, которые являются второстепенными для управления и также рассматриваются в этом разделе), никогда не меняют никаких флагов.

CdM-8 flag semantics

- N – sign bit of the result. Used for signed comparison
- C – carry bit of the result. Used for unsigned comparison
- Z – result is zero. Used for signed, unsigned and bitwise comparison
- V – signed overflow (sign loss). Can be used to catch errors
- V is also needed for *correct* signed comparison

Selection: do something or do nothing (a simple if)
Выбор: делать что-то или ничего не делать (простое если)



- The instruction if opens the conditional construct
- The instruction is marks the decision point
- And the instruction fi closes the construct.

Full list of CdM-8 branch instructions

condition	test	interpretation
eq/z	Z	equal, equal to zero / Zero is set
ne/nz	¬Z	not equal, not zero, Zero is clear
hs/cs	C	unsigned higher or same / Carry is set
lo/cc	¬C	unsigned lower / Carry is clear
mi	N	negative (minus)
pl	¬N	positive or zero (plus)
vs	V	oVerflow is set
vc	¬V	oVerflow is clear
hi	C∧¬Z	unsigned higher
ls	¬C∨Z	unsigned lower or same
ge	(N∧V) ∨ (¬N∧¬V)	greater than or equal, greater than or equal to zero
lt	(N∧¬V) ∨ (¬N∧V)	less than, less than zero
gt	(¬Z∧N∧V) ∨ (¬Z∧¬N∧¬V)	greater than, greater than zero
le	(Z∨N∧¬V) ∨ (¬N∧V)	less than or equal, less than or equal to zero

Figure 5.4: Control conditions.

Control conditions (Z)

eq/z	Z	equal, equal to zero / Zero is set
ne/nz	¬Z	not equal, not zero, Zero is clear

• при сравнении двух одинаковых чисел, в этом случае флаг имеет значение «равно», или
• при тестировании одного 0-значного операнда или его перемещении (т.е. копировании) в другой регистр, в этом случае флаг имеет значение «ноль».

Control conditions (C)

hs/cs	C	unsigned higher or same / Carry is set
lo/cc	¬C	unsigned lower / Carry is clear

• При сравнении чисел без знака флаг C равен 1, если первое число больше или равно второму, и 0, если первое число меньше второго. Для этих обстоятельств мы используем имена hs и lo.

• Другие операции ALU (важно сдвиги) изменяют C, и может потребоваться выполнение различных действий в зависимости непосредственно от его значения. В этих обстоятельствах мы используем имена cs (C set) и cc (C cleared).

C and unsigned subtraction/comparison again

- Subtraction ⇔ adding 2’complement
- When the result < 0, C is 0
- 1-255 = 1+0000 0001 = 2
- When the result > 0, C is 1
- 3-2 = 11+1111 1110 = 1+C

More about branches

- In typical assembler, branch is like goto statement.
- You must invent label names and jump to labels
- Typical equivalent of

```
if (condition) { then-block } else {else-block)
```

  requires one comparison, two labels, one branch and one jump
- (unconditional branch)

Condition calc
b[!cond] $1
    Then-block
Br $2
$1: Else-block
$2: …

CdM-8 assembler has richer syntax

```
If
    Calc condition
is cond
    Then-block
Else
    Else-block
Fi
```

- Инструкция if открывает условную конструкцию
- Затем следует анализ: сегмент кода, который устанавливает флаги в регистре PS
- Инструкция отмечает точку принятия решения и содержит условие
- За этим следует консеквент: сегмент кода, который выполняется, когда условие истинно.
- Инструкция else отмечает конец последовательного
- Далее следует альтернатива: сегмент кода, который выполняется, когда условие ложно.
- Инструкция fi закрывает конструкцию.

Real example

```
if
        tst r0
is z
        ldi r1, 10
        add r1, r0
else
        shla r0
fi
```

- Consult tome.pdf for syntax for complex conditions
- (it is not so elegant)

Repetition (do the same thing over and over again)
Повторение (делать одно и то же снова и снова)

while

• A while loop executes a segment of code repeatedly while a certain condition stays true each time the program gets to the start of the segment
(Цикл while повторно выполняет сегмент кода, при этом определенное условие остается истинным каждый раз, когда программа достигает начала сегмента.)

until

• An until loop executes a segment of code repeatedly until a certain condition becomes true when the program reaches the end of the segment.
(Цикл until повторно выполняет сегмент кода до тех пор, пока определенное условие не станет истинным, когда программа достигнет конца сегмента.)

общие черты циклов

- Сегмент кода обычно называют телом цикла.
- Каждый раз, когда выполняется тело, мы называем это итерацией цикла.
- Количество выполнений тела контролируется проверкой условия

– Непосредственно перед каждым выполнением тела ((while-пока). Известен как цикл проверки при входе.

Или же

— Сразу после каждого выполнения тела (until-до). Известен как цикл «тест-выход».

- Во многих алгоритмах важно знать, сколько итераций уже было выполнено. Это делается путем настройки счетчика итераций.
- Должна быть возможность изменения результата проверки условия от одной итерации к другой, иначе цикл никогда не завершится.

The while loop construct

```
# r2=r0*r1 (assuming r1 is non-negative)
        clr r2
while
        tst r1
stays gt
        add r0, r2
        dec r1
wend
```

while — конструкция цикла проверки при входе. Тело цикла выполняется тогда и только тогда, когда условие продолжения истинно в точке, где программа достигает начала цикла. Выполнив тело, программа возвращается в начало цикла для повторной проверки условия. Это продолжается до тех пор, пока проверка условия продолжения не вернет ложный результат, после чего выполнение программы продолжится с первой инструкции после тела цикла.

The until loop construct

- until — это конструкция цикла test-to-exit. Тело цикла выполняется от начала до конца, а в конце проверяется условие завершения. Если оно ложно, программа возвращается к началу тела цикла и выполняет его снова. Это продолжается до тех пор, пока проверка условия завершения не вернет истинный результат, после чего выполнение программы продолжится с первой инструкции после окончания цикла.

Post-condition loop

```
# find a zero
        ldi r0, array-1
# Initialise r0 to point to the cell before the first element of the array.
do
        inc r0          # point r0 to the next element
        ld r0, r1       # read the element into r1
        tst r1          # examine it
until z                 # if r1 is 0 then exit, otherwise continue
```

Early termination of a single iteration or of a whole loop
Досрочное завершение одной итерации или всего цикла

break Прерывание инструкции приводит к тому, что выполнение программы переходит к первой инструкции после окончания цикла, как если бы было выполнено правильное условие для выхода из цикла.
Интересно, что разрыв может произойти внутри конструкции **if** произвольной сложности, поэтому таким образом можно установить сложное условие завершения для любого типа цикла.

```
# r2=r0*r1 with overflow detection
            clr r2
while
           dec r1
stays ne
           if
               add r0, r2
          is vs
            break
           fi
wend
# At this point V=1 if multiplication resulted in overflow
# Otherwise V=0, in which case the result in r2 is valid.
```

Early termination of a single iteration or of a whole loop
Досрочное завершение одной итерации или всего цикла

continue Когда необходимо прекратить выполнение текущей итерации цикла, но продолжить выполнение цикла с начала следующей итерации.

Например, если мы хотим работать с массивом в поисках первого нуля, которому не предшествует непосредственно 3, мы можем изменить цикл **until.**

```
# find a zero that does NOT come after 3 in an array
        run find
array:
        dc 3,5,2,7,3,0,57,8,0,6
find:
        clr r1                  # предположим, что предыдущий элемент
равен 0 в начале
        ldi r0, array-1         # предыдущий адрес перед началом
do
        move r1, r2             # r2 будет содержать предыдущий элемент
        inc r0                  # указание r0 на следующий элемент
        ld r0, r1               # получите это число в r1
        if
            ldi r3, 3
            cmp r2, r3          # предыдущий элемент = 3?
        is eq                   # if yes
        continue                # не интересует текущий элемент, переходите к
следующей итерации
        fi
        tst r1                  # examine current element
until z                         # if current element = 0 then exit, otherwise go on
halt
End
```

break n

По умолчанию разрыв завершает самый внутренний цикл, в котором он появляется, поэтому, если он находится внутри цикла, который сам вложен в другой цикл, он завершит только самый внутренний цикл.
Тем не менее, одиночный разрыв может быть использован для одновременного завершения всего гнезда циклов (до 4 в глубину) путем предоставления числового аргумента, поэтому **break 2** завершит как самый внутренний цикл, в котором он находится, так и цикл, в котором он вложен (но ничего дальше).

continue n

Инструкция **continue** также может иметь необязательный аргумент и завершать итерацию внешнего цикла. Например **continue 2** передаст управление в начало следующей итерации цикла, в который вложена текущая; текущая петля будет полностью заброшена. Альтернативой может быть выход из внутреннего цикла, а затем использование **continue** для продолжения выполнения с начала следующей итерации внешнего цикла, но это будет означать наличие двух тестов: по одному на каждом уровне, и это потребует некоторой программной акробатики, чтобы гарантировать, что внешний тест вызовет прерывание только тогда, когда внутренний тест вызовет продолжение.

Nesting of if’s and loops is possible

- You can use them like blocks in high-level languages
- You do not need to invent label names
- You do not need to worry about correct nesting
- Much harder to write spaghetti code (than with raw branches)
- This is why CdM-8 assembly is called Platform 3 ½
- Actually, it is much simplier to implement than you probably think
- It is all described in tome.pdf
- Beware: in some exercises using structural statements is explicitly prohibited