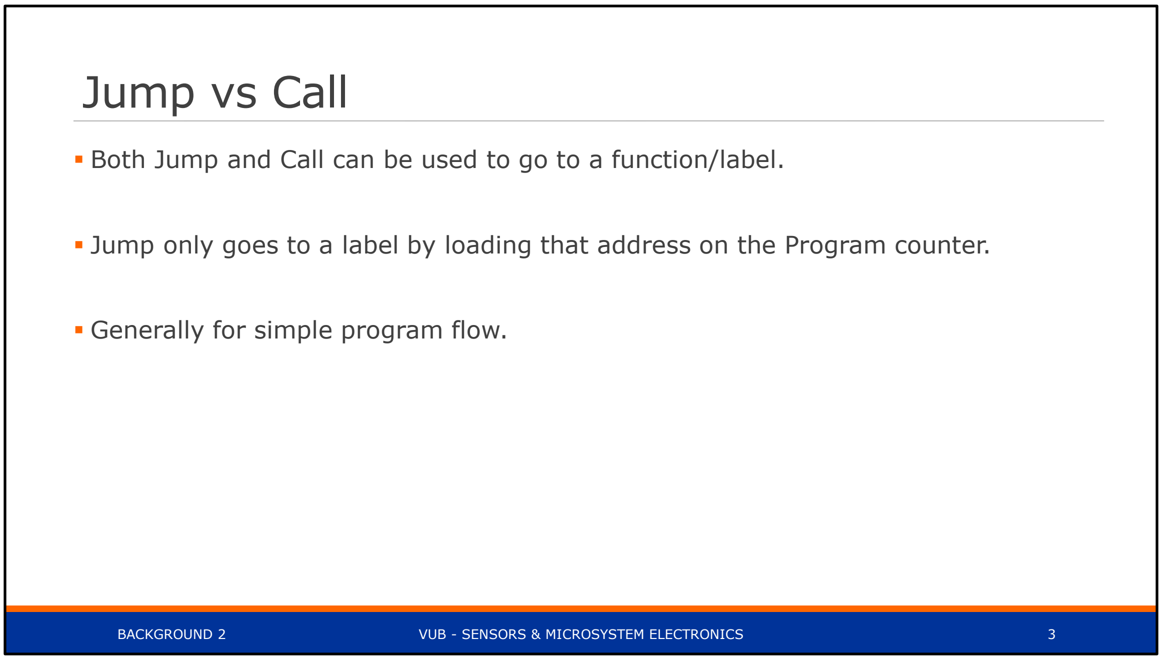
Sensors & Microsystem Electronics: microcontrollers

BACKGROUND 2: PROGRAM FLOW

Program Counter?

- Program Counter (PC) is a 16-bit register that stores the memory address of the instruction that is currently executed from the program memory
- On boot/reset, the Program Counter (PC) is set to 0x0000
- PC increases by 1 per executed instruction (or 2 for certain ones)
- Jumping, subroutine calls, branching and skipping instructions modify the PC directly to change program flow.

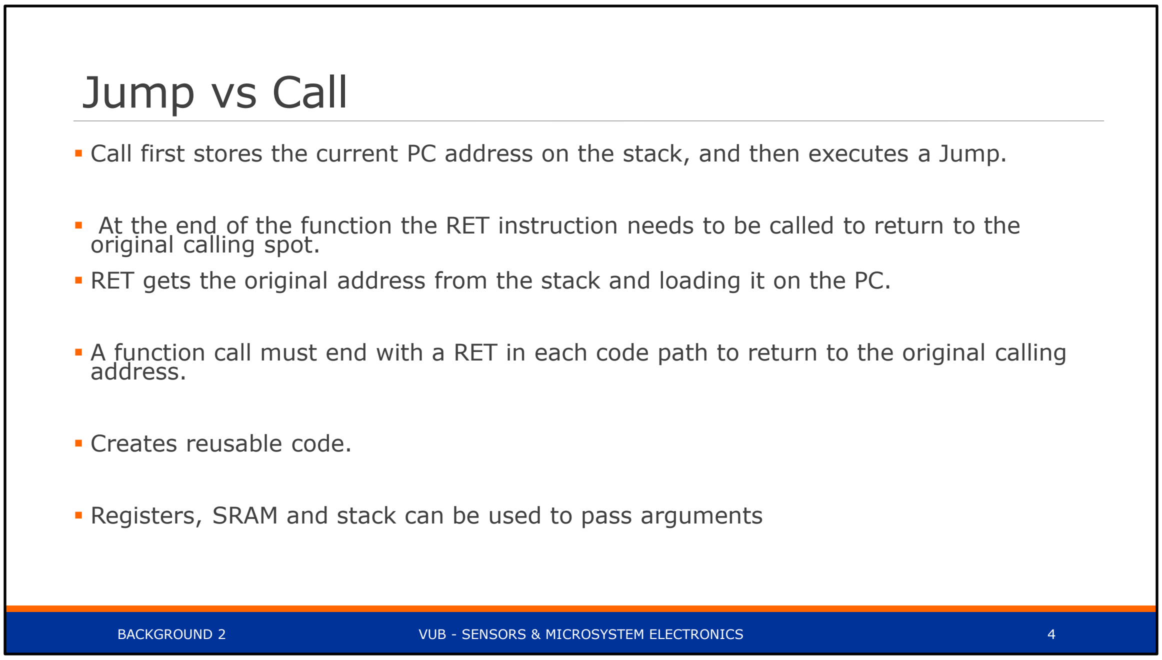
Mnemonic	Operands	Description	Operation	Flag	Cycles
BRNE	k	Branch if Not Equal	if (Z== 0) then PC= PC + k + 1	None	1/2.



Jump vs Call

- Both Jump and Call can be used to go to a function/label.
- Jump only goes to a label by loading that address on the Program counter.
- Generally for simple program flow.

It is important to pay attention when mixing jumps and calls. One should make sure that all code paths after the function call end with a return back to the original call location. If this is not done a stack overflow will take place and the execution will be disturbed.

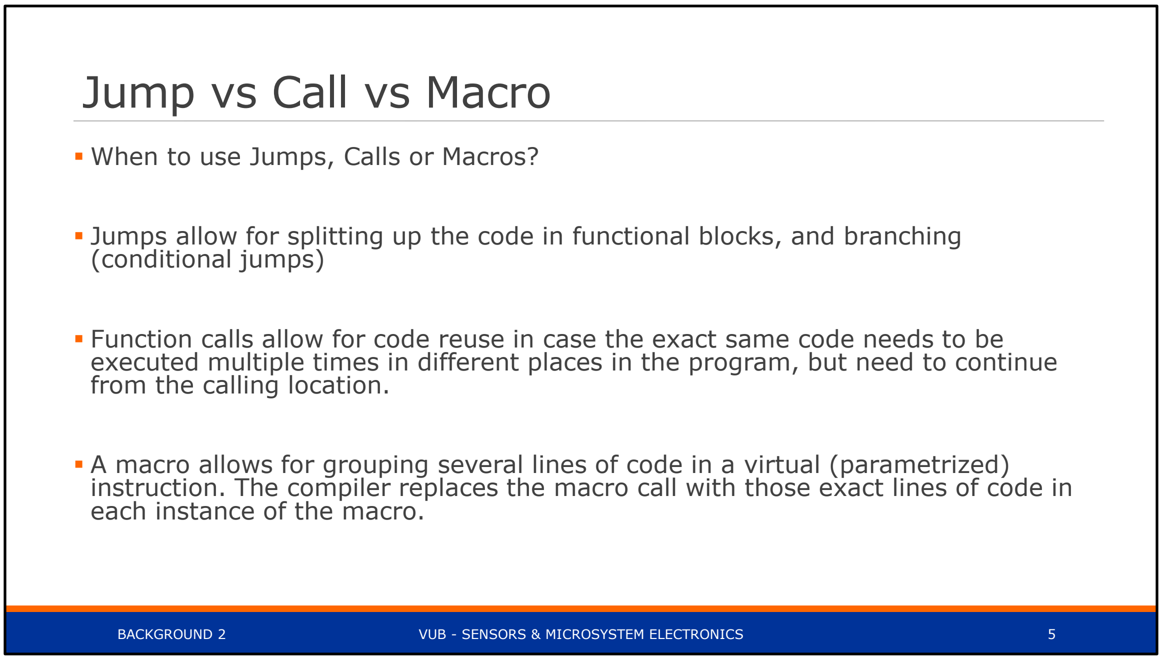


Jump vs Call

- Call first stores the current PC address on the stack, and then executes a Jump.
- At the end of the function the RET instruction needs to be called to return to the original calling spot.
- RET gets the original address from the stack and loading it on the PC.
- A function call must end with a RET in each code path to return to the original calling address.
- Creates reusable code.
- Registers, SRAM and stack can be used to pass arguments

BACKGROUND 2 VUB - SENSORS & MICROSYSTEM ELECTRONICS 4

It is important to pay attention when mixing jumps and calls. One should make sure that all code paths after the function call end with a return back to the original call location. If this is not done a stack overflow will take place and the execution will be disturbed.


Jump vs Call vs Macro

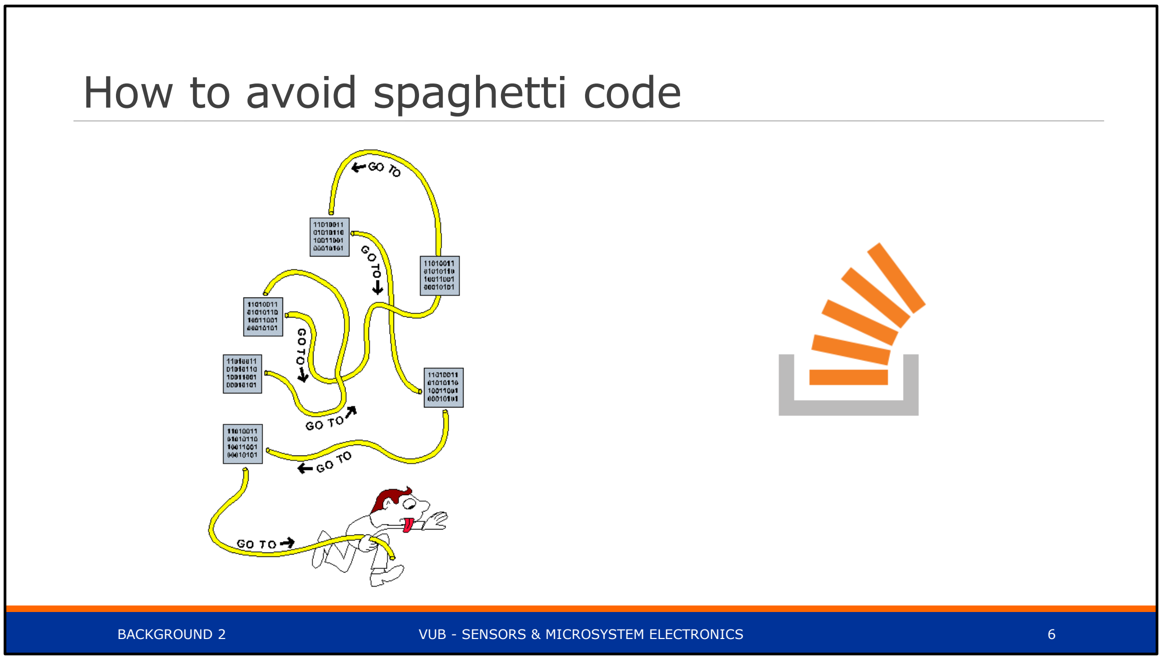
- When to use Jumps, Calls or Macros?
- Jumps allow for splitting up the code in functional blocks, and branching (conditional jumps)
- Function calls allow for code reuse in case the exact same code needs to be executed multiple times in different places in the program, but need to continue from the calling location.
- A macro allows for grouping several lines of code in a virtual (parametrized) instruction. The compiler replaces the macro call with those exact lines of code in each instance of the macro.

BACKGROUND 2 VUB - SENSORS & MICROSYSTEM ELECTRONICS 5

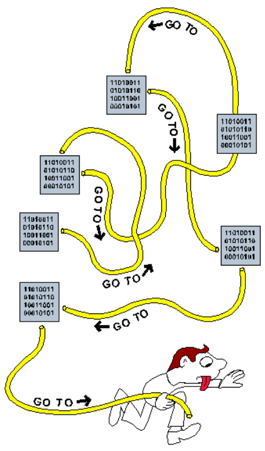
Do not mistake macros for function calls. Macros will be replaced by their equivalent lines of codes at compile time. An example: say you have a macro of 10 lines, that you use 10 times in your program, at compile time this results in 100 lines of code due to the macro. If this would be a function call, this would be 10*1 line for each function call + 10 lines of the function.
Generally macros are only used to package small amounts of code with parameters to make "custom" instructions. E.g this macro to load a constant in the lower 16 registers:

```
.macro ldi_low   ;macro to safely load constant in Rx < 16
                              ; (LDI only works on R16-31)
push R16                      ;store R16 on stack
ldi R16, @1                   ; load immidiate from argument 1 in R16
mov @0, R16                   ; copy R16 to register specified by arg 0
Pop R16                       ; restore R16 with original value
.endmacro

; USE OF MACRO:
ldi_low R3, 0x20  ; loads 0x20 in R3
```



How to avoid spaghetti code

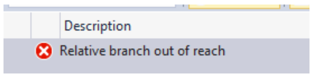


When code structure and modularity is lost, spaghetti code comes into existence. In assembly the developer is fully responsible to manage every little thing of the program: code flow, memory, and peripherals. When using only jumps subroutines can only be used once without code duplication. However, when using function calls one must make sure that each different code path within the function terminates with an RET statement. Should you accidentally decide to jump back to somewhere in the main function, the calling address is never cleared from the stack. Which might keep stacking those addresses on the stack until a “stack overflow” happens.

Relative branch out of reach

- Branching has a maximum distance of 64 addresses
- RJMP can jump up to 2K addresses
- JMP can jump 4M addresses

```
CPI temp, 0b01110111
BREQ B0b ;0 →Relative branch out of reach
```



Description
Relative branch out of reach

- Relative branch out of reach → Destination label is too far away.
- Solution: branch to label with (R)JMP to original label

```
CPI temp, 0b01110111
BREQ Tmp_B0b ;0 →Relative branch out of reach

…
Tmp_B0b: RJMP B0b ;Intermediate label with rjmp
```