The 68000

Lesson 5 - Branching

Branching

- Labels
- Comparisons, status register...
- Conditional Branch...
- Conditional DEBranch
- Non conditional Branch

Labels

- It is possible to define labels
- Use as in the “goto” functions
- Syntax:
  - “name_of_label:”
- The “:” is essential, allowing the compiler to understand the label

Comparisons

- It is possible to compare 2 operands
- CMP   D0, D1
- This does D1 – D0 and stores the correct flags in the Status Register

Comparison - Notes

- The result of D1 - D0 is not stored in memory or anywhere: it is virtual...
- The result of a standard arithmetic operation usualy affects the flag register in the same way: you may thus remove some cmp from your programs...

Conditional Branch

- In order to do more complicated programs it is possible to use conditional branching
- It “emulates” the *if…then* instructions
- Many different possible branches
- Checks the status register

Branch list... (1 / 4)

- BCC Branch Carry Clear
  - Branch if the C-flag is 0.
- BCS Branch Carry Set
  - Branch if the C-flag is 1.
- BEQ Branch Equal
  - Branch if the Z-flag is 1.
- BNE Branch Not Equal
  - Branch if the Z-flag is 0.

Branch list... (2 / 4)

- BGE Branch Greater or Equal
  - Branch if N and V are equal.
- BGT Branch Greater Than
  - Branch if N and V are equal and Z=0.
- BLT Branch Less Than
  - Branch if N and V are different.
- BLE Branch Less or Equal
  - Branch if Z=1 or if N and V are different.

Branch list... (3 / 4)

- BLS Branch Lower or Same
  - Branch if C=1 or Z=1.
- BHI Branch HIgher than
  - Branch if both C and Z are 0.
- BMI Branch Minus
  - Branch if N=1.
- BPL Branch Plus
  - Branch if N=0.

Branch list... (4 / 4)

- BVC Branch V Clear
  - Branch if V=0
- BVS Branch V Set
  - Branch if V=1.
- BRA BRanch Always

The Fellowship Of The Sign

- As you should have noticed, some instructions are similar
- We will use BGE, BGT, BLE, BLT for signed integers and BHI, BLS for unsigned integers
- Let’s see how to use that…

Branch syntax

- The opcode is: BXX <label>
- The .W and .B may be used, though it is not suitable, since the compiler will automatically select the right value
- Example: BCS isn_t_this_a_nice_label Will go to the selected label if the last instruction set the C flag to 1

Some equivalences

- Imagine you're doing:

```
CMP  B, A
BXX   name_of_label
```

- Let's see when would this go to "name_of_label" depending on the value of XX

Working with unsigned integers

- BHI
  - A > B (branch if higher)
- BCC
  - A >= B (branch if carry clear)
- BLS
  - A <= B (branch if lower or same)
- BCS
  - A < B (branch if carry set)
- BEQ
  - A = B (branch if equal)
- BNE
  - A <> B (branch if non equal)

Working with signed integers

- BGT
  - A > B (branch if greater)
- BGE
  - A >= B (branch if greater or equal)
- BLE
  - A <= B (branch if lower or equal)
- BLT
  - A < B (branch if lower)
- BEQ
  - A = B (branch if equal)
- BNE
  - A <> B (branch if non equal)

Debranching

- Used to quit loops
- The instruction is very similar to BXX
- DBXX   Dn, <label>
- The first operand is a data register that will be decreased by one until it reaches -1, then the loop stops

Debranching (Cont.)

- A loop can also be quit if the flags are set correctly, using for example de ANDI, ORI or EORI instructions
- We usually use DBRA (never quit) that will quit the loop only if Dn equals –1
  - For example to loop 10 times, set Dn to 9

Unconditional Branch

- Normal branch and jump
- Subroutine branch and jump
- Return

Normal Unconditional

- BRA
  - This is the BRanch Always, it doesn't check any flag
  - Syntax: BRA <label>
- JMP
  - This is the JuMP, it is used to move the program control to an effective address. It is equivalent of doing MOVE.L <address>, PC
  - Syntax: JMP <address>

Subroutine Unconditional

- BSR and JSR
- Used to always branch to a subroutine
- A subroutine is something from which you will get back when executed
- After the subroutine is executed the program will continue to execute just after the BSR or JSR opcode

BSR

- Mainly used for your own subroutines
- BSR <label>
- This instruction will:
  - Push the address to the next instruction on the stack
  - Branch to the label specified in the instruction

JSR

- Works as JMP except that before the jump is made, the address to the instruction after JSR is pushed to the stack
- JSR <address>
- Used to call a subroutine from the libraries
  - “JSR lib_name::sub_routine_name“
  - Only works if you included a lib_name.h file

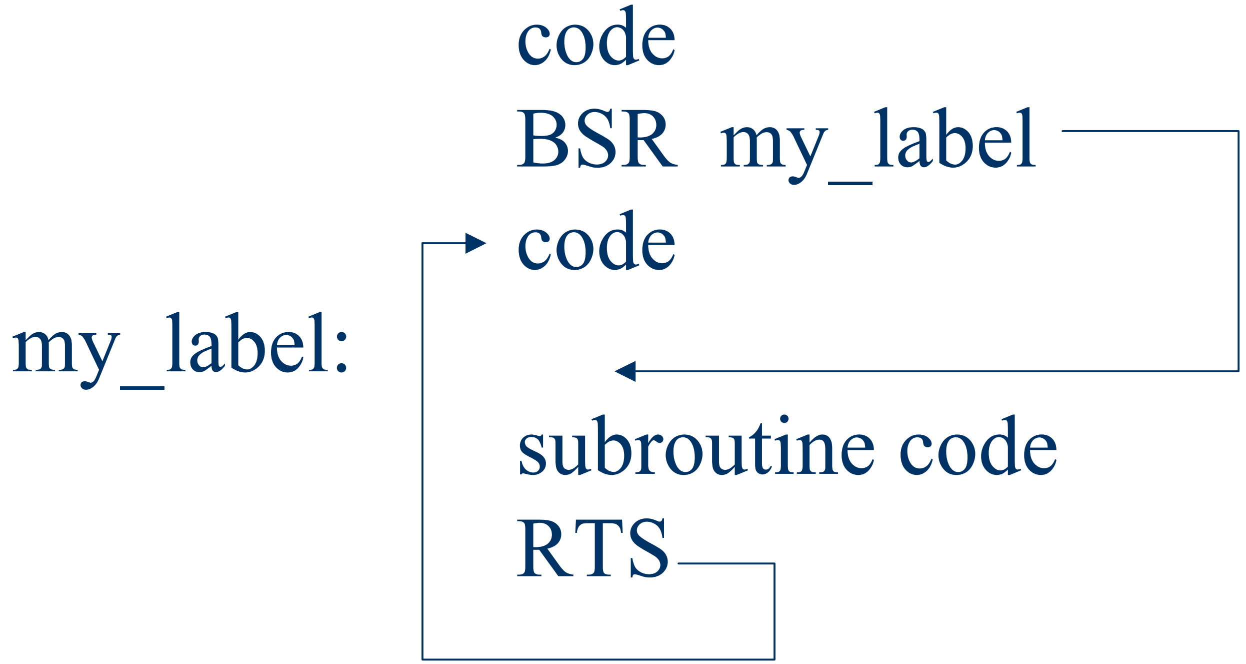
Return - RTR

- ReTurn and Restore
- Pops the flags and the program counter from the stack
- A word is popped from the stack and the lower byte of that word is stored in the flag register (The higher byte is ignored)
- Then a longword is popped into the PC
- The stack pointer will be increased with 6 (= 4 + 2)

Return - RTS

- ReTurn from Subroutine
- Does the opposite to the instructions BSR and JSR
- The longword on top of the stack is stored in the PC
- The instruction is used when ending a subroutine
- The execution will return to the instruction that follows the last JSR- or BSR- instruction

How to use...



```
           code
           BSR  my_label
           code

my_label:
           subroutine code
           RTS
```

Today’s Exercise #1 - Easy

- Write a program that will add 69 successive words stored in memory. The address of the first number is stored in A1.
- You should only use the ADD instruction once in the whole code
- Store the result in the address given by A6

Today’s Exercise #2 - Medium

- Write a program that takes the biggest of 3 given numbers in word format stored in contiguous memory address. The first address given by A1. The number will be stored in the address given by A0

Today’s Exercise #3 – Hard (?)

- Write a program that will store the longword in the address given by A1 (it will be “total”) and then do the following subroutine for each longword in memory starting from the address given by A2:
    - Substract the lower number from the biggest
    - Shift left the total (Logical Shift)
- You shall stop when you get an overflow after the shifting or if you have used 100 numbers
- Store the number of longwords used into the address given by A0

Example for ex #3

- Imagine we have (A1) = “NUM” and A2 = $00000002
- First step: “total” = “NUM” and “count” = 0
- Then, you take the number in $00000002, do the subroutine, and increase count by 1
- Continue with the number in $00000006 until there’s an overflow or count = 100

Some notes...

- You don’t have to use a counter starting from 0 and going to 100, you may want to do it the opposite way (countdown 100->0), or count from 256 to 356...
- Try using everything that we studied (branching, subroutines, autoincrements...)