Lecture 6: Subroutines I

- Introduction to subroutines
- The stack
- BRS/JSR and RTS instructions
- An example: compute the statistical average
- Parameter passing to subroutines
  - Passing by value
  - Passing by reference
Introduction to subroutines

- What is a subroutine?
  - A subroutine is a coherent sequence of instructions that carries out a well-defined function
  - Conceptually, a subroutine is similar to a function call in a high-level language

- Why subroutines?
  - The same sequence of instructions can be used many times without the need to rewrite them over and over
  - Subroutines make programs easier to write (in a top-down fashion) and maintain

- Examples of subroutines
  - Print a message to the display, compute an average value, initialize an I/O port, any more?
Subroutines made easy

- **When a program calls a subroutine**
  - The address $N$ of the next instruction in the program is saved in a special memory location called the **STACK**
  - The PC is loaded with the starting address of the subroutine
  - The CPU performs another “fetch-execute” cycle… *this time at the first instruction of the subroutine!*

- **When a subroutine is done**
  - The old address $N$ is recovered from the **STACK**
  - The program counter is loaded with $N$
  - The CPU performs another “fetch-execute” cycle… *this time at the next instruction in the program after the subroutine call!*
The stack

- The stack is a special area in memory reserved for reading and writing special data items
  - The stack is a LIFO structure (LAST IN, FIRST OUT) since the last item pushed is always the first item popped
  - The address of the last item pushed on the stack is stored in A7, also referred to as the Stack Pointer (SP)
- Data is pushed (saved) on the stack with MOVE.L
  - MOVE.L DO, -(A7)
- Data is popped off (recovered from) the stack with MOVE.L
  - MOVE.L (A7) +, D0
- Always move long-words to the stack!!!
The stack (an example)

MOVE.L D0,-(A7)  MOVE.L D1,-(A7)  MOVE.L D2,-(A7)  MOVE.L (A7)+,D2
**BSR/JSR instruction: calling a subroutine**

- **BSR/JSR used to Branch(Jump) to a SubRoutine**
- **BSR/JSR performs three operations**
  - Decrement the stack pointer
  - Push the program counter on the stack
  - Load the program counter with the target address
- **For example, in RTL “BSR AVG” is equivalent to**

\[
\begin{align*}
[A7] & \leftarrow [A7] - 4 \\
[M([A7])] & \leftarrow [PC] \\
[PC] & \leftarrow \text{AVG}
\end{align*}
\]
**RTS instruction: returning from a subroutine**

- **RTS used to ReTurn from a Subroutine**
- **RTS performs two operations**
  - Pull the return address from the stack
  - Post-increment the stack pointer
- **For example, in RTL “RTS” is equivalent to**

  
  
  \[
  [\text{PC}] \leftarrow [M([A7])] \\
  [A7] \leftarrow [A7] + 4
  \]
Subroutines and the stack

MAIN

BSR SUB1

... END

SUB1

... BSR SUB2

... RTS

SUB2

... RTS

LEGEND

A: stack immediately before BSR SUB1
B: stack immediately after BSR SUB1
C: stack immediately after BSR SUB2
D: stack immediately after first RTS
E: stack immediately after second RTS
### A simple example

- **Compute the average of an array of numbers**
  - In our case

\[
\text{avg}(1,\ldots,10) = \frac{1+\ldots+10}{10} = 5.5
\]

<table>
<thead>
<tr>
<th>Address</th>
<th>Mnemonic</th>
<th>Op1</th>
<th>Op2</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001000</td>
<td>ORG</td>
<td>$1000</td>
<td></td>
<td>data section</td>
</tr>
<tr>
<td>00001000</td>
<td>SIZE</td>
<td>DC.L</td>
<td>10</td>
<td>size of the array</td>
</tr>
<tr>
<td>00001004</td>
<td>ARRAY</td>
<td>DC.L</td>
<td>1,2,3,4,5,6,7,8,9,10</td>
<td>array data</td>
</tr>
<tr>
<td>00002000</td>
<td>ORG</td>
<td>$2000</td>
<td></td>
<td>main program section</td>
</tr>
<tr>
<td>00002000</td>
<td>MOVE.L</td>
<td>SIZE,D0</td>
<td></td>
<td>pass size in D0</td>
</tr>
<tr>
<td>00002004</td>
<td>LEA</td>
<td>ARRAY,A0</td>
<td></td>
<td>pass array address in A0</td>
</tr>
<tr>
<td>00002008</td>
<td>BSR</td>
<td>AVG</td>
<td></td>
<td>call subroutine</td>
</tr>
<tr>
<td>0000200C</td>
<td>RTS</td>
<td></td>
<td></td>
<td>graceful exit</td>
</tr>
<tr>
<td>00003000</td>
<td>ORG</td>
<td>$3000</td>
<td></td>
<td>subroutine section</td>
</tr>
<tr>
<td>00003000</td>
<td>AVG</td>
<td>MOVE.L</td>
<td>D0,D1</td>
<td>D1 is the countdown index</td>
</tr>
<tr>
<td>00003002</td>
<td>CLR.L</td>
<td>D2</td>
<td></td>
<td>D2 returns the avg. value</td>
</tr>
<tr>
<td>00003004</td>
<td>REPT</td>
<td>ADD.L</td>
<td>(A0)+,D2</td>
<td>traverse the array</td>
</tr>
<tr>
<td>00003006</td>
<td>SUBI</td>
<td>#1,D1</td>
<td></td>
<td>decrement counter</td>
</tr>
<tr>
<td>0000300A</td>
<td>BNE</td>
<td>REPT</td>
<td></td>
<td>repeat while D1&gt;0</td>
</tr>
<tr>
<td>0000300C</td>
<td>DIVS</td>
<td>D0,D2</td>
<td></td>
<td>divide the sum by #elements</td>
</tr>
<tr>
<td>0000300E</td>
<td>RTS</td>
<td>D0,D2</td>
<td></td>
<td>return to the main program</td>
</tr>
<tr>
<td>00003010</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Run on sim68k

MC68000/ECB Simulator.
Copyright (C) Livadas and Ward, 1992. Author Wayne Wolf
Version 2.3
SIM68000 2.3 > LO L6.S
SIM68000 2.3 > PC 2000
SIM68000 2.3 > DF
PC=00002000 SR=0000=.0... US=00007000 SS=00007E00
D0=F9315AE6 D1=390E2E5F D2=8E0FB027 D3=B78A9DAA
A0=718D7B96 A1=8C93757E A2=713179ED A3=6439C3E7
---------------00002000 2038 1000
MOVE.L $001000,D0

SIM68000 2.3 >
PC=00002004 SR=8000=.T... US=00007000 SS=00007E00
D0=0000000A D1=0000000A D2=0000000A D3=B78A9DAA
A0=00010108 A1=8C93757E A2=713179ED A3=6439C3E7
---------------00002004 41F8 1004
LEA $001004,A0

SIM68000 2.3 >
PC=00002008 SR=8000=.T... US=00007000 SS=00007E00
D0=00000004 D1=00000004 D2=00000004 D3=B78A9DAA
A0=00001010 A1=8C93757E A2=713179ED A3=6439C3E7
---------------00002008 6100 0FF6
MOVE.L  $001000,D1

SIM68000 2.3 >
PC=00003000 SR=8000=.T... US=00007000 SS=00007E00
D0=0000000A D1=0000000A D2=0000000A D3=B78A9DAA
A0=00000100 A1=8C93757E A2=713179ED A3=6439C3E7
---------------00003000 2200
MOVE.L  D0,D2

SIM68000 2.3 >
PC=00003004 SR=8000=.T... US=00007000 SS=00007E00
D0=0000000A D1=0000000A D2=0000000A D3=B78A9DAA
A0=00000100 A1=8C93757E A2=713179ED A3=6439C3E7
---------------00003004 0441 0001
SUB.W  #$0001,D1

SIM68000 2.3 >
PC=0000300C SR=8000=.T... US=00007000 SS=00007E00
D0=0000000A D1=0000000A D2=0000002D D3=B78A9DAA
A0=0001010C A1=8C93757E A2=713179ED A3=6439C3E7
---------------0000300C 4E75
RTS

SIM68000 2.3 >
AVG(1,...,10) = 5.5
A look at the STACK on sim68k
Parameter passing

- **Two ways to pass parameters to a subroutine**
  - **By value**: the actual data is passed in a register
  - **By reference**: the address in memory of the parameter is passed in a register

- **In the previous example (BSR AVG)**
  - The number of items in the array (**SIZE**) was passed by value since it was loaded on D0 (**MOVE.L SIZE, D0**)
  - The items (**ARRAY**) were passed by reference since their starting address was loaded on A0 (**LEA ARRAY, A0**)

- **These parameter passing methods are limiting**
  - They do not allow re-entrant code
  - Using the stack is the preferred method (next lecture)