# Microcomputers I – CE 320

Mohammad Ghamari, Ph.D.
Electrical and Computer Engineering
Kettering University

# Announcement

- You are going to have quiz no.4 next Friday, Feb 12.
  - Covers homework exercise 5
    - Stack and subroutines

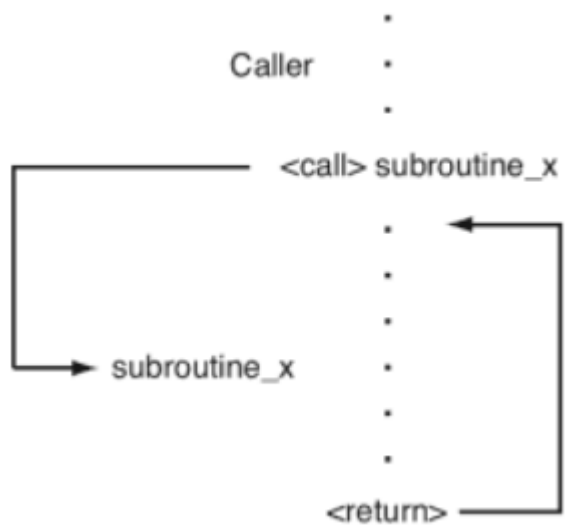
# Lecture 16: Parameter Passing

# Today’s Topics

- Use different methods to pass parameters to subroutines
- Use stack frames to track data in use by a subroutine

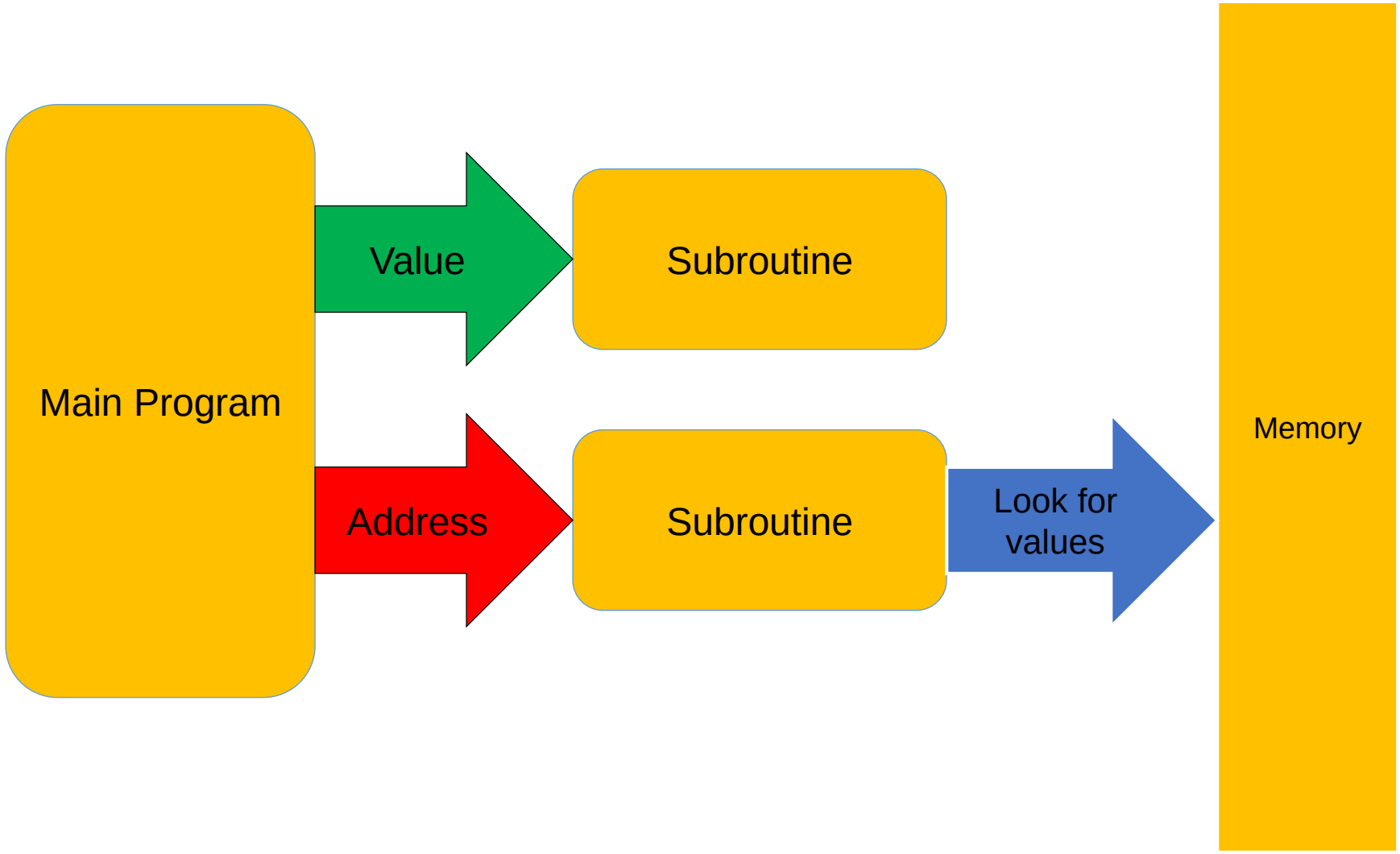
# Subroutine

- A subroutine is a sequence of instructions that can be called from many different places in a program.
- The program unit that makes the subroutine call is referred to as a **caller**.
- The subroutine called by other program units is referred to as a **callee**.
- The caller usually wants the subroutine to perform a computation using the parameters passed to it.




**Program flow during a subroutine call**

# Call by Value vs. Call by Reference



# Parameter Passing Methods

- The caller may use the following methods to pass parameters to the subroutine:
  - **Use Registers**
    - In this method, parameters are **placed in CPU registers** before the subroutine is called.
    - This method is very convenient when there are only a few parameters to be passed.
  - **Use the stack**
    - In this method, parameters are **pushed into the stack** before the subroutine is called.
    - The stack must be cleaned up after the computation is completed. This can be done by either the caller or the callee.
  - **Use the global memory**
    - **Global memory is accessible to both the caller and the callee**.
    - As long as the caller places parameters in global memory before it calls the subroutine, the callee will be able to access them.

# Parameter Passing Methods

- Passing parameters using memory:
  - The caller sets memory locations and the callee reads them.

Example:

```
movb #2,$1000
movb #4,$1001
bsr addition
```

The caller stores the data in memory locations, e.g., $1000 and $1001

```
addition:
    ldaa $1000
    adaa $1001
```

The subroutine reads the data from memory locations, e.g., $1000 and $1001

# Parameter Passing Methods

- Passing parameters using registers:
  - The caller sets registers and the callee reads them.

Example:

```
ldaa #2
ldab #4
bsr
addition
```

The caller stores the data in registers, e.g., a and b

```
addition:
        aba
```

The subroutine reads the data from the registers

# Parameter Passing Methods

- Passing parameters using stack:
  - The caller pushes the parameters in the stack and the callee reads them.

Example:

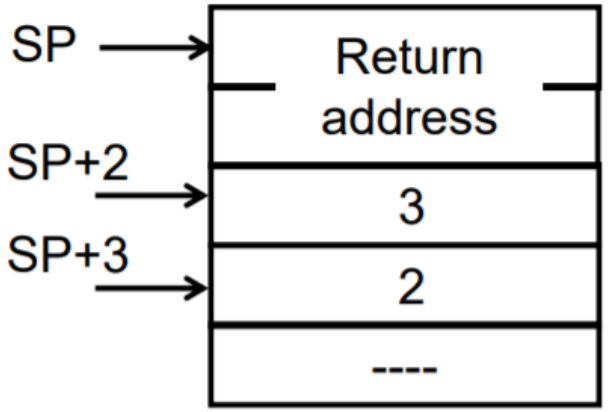
```
ldaa #2
psha
ldaa #3
psha
bsr addition
leas 2,+sp
```

The caller pushes the data in stack before calling the subroutine

The stack is cleaned after the subroutine call

```
addition:
    tfr sp,X
    ldaa 2,+x
    ldaa 1,+x
```

The subroutine reads the locations sp+2 and sp+3



# Passing Parameters in Registers

- This is often the **preferred method** of passing data to and from the subroutine.

- This method has the following advantages and disadvantages:

  - Data is **immediately available** to the subroutine.

  - Registers must usually be used to move data using the other methods, and this saves the extra preparation.

  - **Often fastest execution**, smallest code size

  - There are **only a limited number of registers available**, however, pass-by-reference may be used to point to a list of input/output values.

# Example

- Write a subroutine that adds an array of two-byte numbers and a sample main program that calls it.
- The array is ***passed by reference*** in X, the length is ***passed by value*** in D.
- The sum should be ***returned by value*** in D.
- Do not worry about indicating signed or unsigned overflow.

```
            ORG     $3000
Array       dc.w    $1234,$5678,$ABCD
Length      dc.w    3

            ORG     $C000
            LDS     #$3600
            LDX     #Array      ; load X with address of list
            LDD     Length      ; Load D with actual length
            JSR     sumword
            SWI

sumword     TFR     D,Y
            LDDD    #0
            CPY     #0
loop        BEQ     endsum
            ADDD    0,X
            INX
            INX
            DEY
            BRA     loop
endsum      RTS
```

# Preserving Registers

- The previous example used Y within the subroutine, which destroyed the original value in that register, and it modified X.
- If Y was in use by the main program, the main program would continue after the subroutine with an incorrect value in Y.
- To avoid this, registers used by a subroutine may be **saved to the stack** before a subroutine uses them, and they are restored before the main program resumes using them.
- For this, there are two options:
  - The caller (main program) does this.
  - The callee (subroutine) does this.

```
            ORG     $3000
Array       dc.w    $1234,$5678,$ABCD
Length      dc.w    3

            ORG     $C000
            LDS     #$3600
            LDX     #Array      ; load X with address of list
            LDD     Length      ; Load D with actual length
            JSR     sumword
            SWI

sumword     TFR     D,Y
            LDDD    #0
            CPY     #0
loop        BEQ     endsum
            ADDD    0,X
            INX
            INX
            DEY
            BRA     loop
endsum      RTS
```

# Preserving Registers

- Responsibility of the Caller:
  - The calling main program assumes all registers are destroyed by the subroutine.
  - Calling program saves only those registers that are in use.
  - If the registers used by the subroutine are unknown (i.e. using a sub. provided by someone else), may save registers that the subroutine wouldn't affect.
  - Code to save/restore registers duplicated with every subroutine call.
- Responsibility of the Callee:
  - Saves only the registers that will be used by the subroutine.
  - Save/restore code only occurs once.
  - May waste time saving registers not in use by the main program.
- Notes:
  - Caller responsible is the **safest bet** if we do not know how the subroutine is implemented.

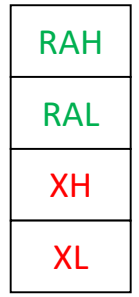
# Example – Caller Responsible

```
        ORG     $3000
Array   dc.w    $1234,$5678,$ABCD
Length  dc.w    3

        ORG     $C000
        LDS     #$3600
        LDX     #Array      ; load list address
        LDD     Length      ; Load actual length
        PSHX
        JSR     sumword
        PULX
        SWI
```

```
sumword TFR     D,Y
        LDDD    #0
        CPY     #0
loop    BEQ     endsum
        ADDD    0,X
        INX
        INX
        DEY
        BRA     loop
endsum  RTS
```

Stack Frame




We push index register X onto stack before we call a subroutine.

# Example – Callee Responsible

**Q:** Why isn't register D saved and restored?

**A:** It is used to return the answer!

```
        ORG     $3000
Array   dc.w    $1234,$5678,$ABCD
Length  dc.w    3

        ORG     $C000
        LDS     #$3600
        LDX     #Array      ; load list address
        LDD     Length      ; Load actual length
        JSR     sumword
        SWI
```

```
sumword PSHX
        PSHY
        TFR     D,Y
        LDDD    #0
        CPY     #0
loop    BEQ     endsum
        ADDD    0,X
        INX
        INX
        DEY
        BRA     loop
endsum  PULY
        PULX
        RTS
```

Stack Frame

| |
|---|
| YH |
| YL |
| XH |
| XL |
| RAH |
| RAL |

We push index registers X and Y onto stack at the beginning of a subroutine.

# Passing Parameters in the Stack

- The stack is the **next preferred** location to pass data. This method has the following advantages and disadvantages.
  - The stack is already in use for saving the return address
  - Indexed addressing can easily access data stored on the stack
  - The amount of data passed is **not limited by the register set**
  - The data passed on the stack must be removed, and this is usually the responsibility of the caller (although this can be done by the callee)
  - However, it takes up more CPU clocks to pass parameters on stack and for the subroutine to access them compared to data passed in registers

- Note:
  - If the caller will save registers on the stack, it should be done before passing parameters.

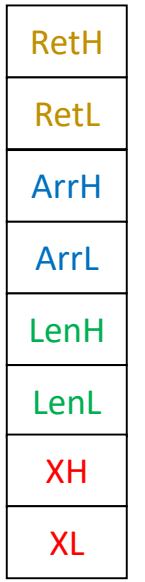
# Example – Parameter Passing in the Stack

```
        ORG     $3000
Array   dc.w    $1234,$5678,$ABCD
Length  dc.w    3

        ORG     $2000
        LDS     #$3600
        LDX     #$1234    ; something to save
        PSHX
        LDD     Length
        PSHD
        LDD     #Array
        PSHD
        JSR     sumword
        LEAS    4,SP
        PULX
        SWI
```

```
sumword LDDD    #0
        LDX     2,SP
        LDY     4,SP
Loop    BEQ     endsum
        ADDD    0,X
        INX
        INX
        DEY
        BRA     loop
endsum  RTS
```

Stack Frame




The **address of the array** and the **two-byte length** are passed on the stack. The sum should be returned by value in D.

# Questions?

# Wrap-up

## What we've learned

- Parameter Passing

- Pros and Cons of each method

# What to Come

- More about subroutines