

# CPE/EE 421 Microcomputers

Instructor: Dr Aleksandar Milenkovic  
Lecture Note  
S05

\*Material used is in part developed by  
Dr. D. Raskovic and Dr. E. Jovanov

## Course Administration

- Instructor: Aleksandar Milenkovic  
[milenka@ece.uah.edu](mailto:milenka@ece.uah.edu)  
[www.ece.uah.edu/~milenka](http://www.ece.uah.edu/~milenka)  
EB 217-L  
Mon. 5:30 PM – 6:30 PM,  
Wen. 12:30 – 13:30 PM
- URL: <http://www.ece.uah.edu/~milenka/cpe421-05F>
- TA: Joel Wilder
- Labs: Lab#1 is on. First session 9/12. Due 9/14.
- Hws: Hw #1 is on. Due 9/21/05, 2:20.
- Text: Microprocessor Systems Design:  
68000 Hardware, Software, and Interfacing
- Review: Chapter 1, Chapter 2;
- Today: Passing Parameters; C and the M68K.

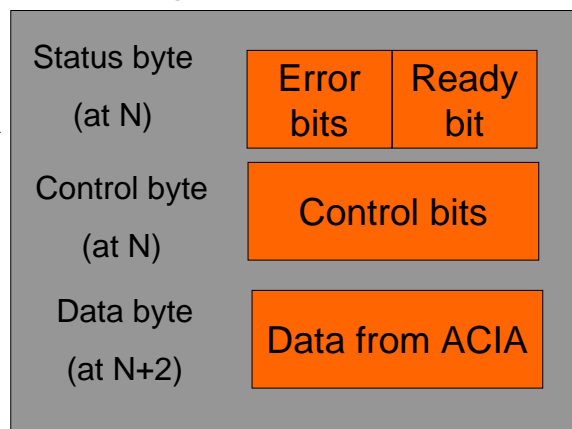
## Assembly Language and C

- We are interested in:
  - ❖ How a high-level language uses low-level language features?
  - ❖ C: System programming, device drivers, ...
  - ❖ Use of addressing modes by compilers
  - ❖ Parameter passing in assembly language
  - ❖ Local storage

## Programmer's view of ACIA

### ACIA registers

- To initialize:
- write #03 to CR
  - write conf. byte to CR
- To read:
- polling on Ready bit
- If no input:
- poll for a specified number of times before exit with an error



## Assembly Language and C, ACIA example

```
Character_Input(Func, Dev_loc, Input_Char, Error_St)
Error_St=0
IF Func = 0
  THEN Initialize Input_Dev
  ELSE Read status of Input_Dev
    IF status OK THEN
      BEGIN
        Set Cycle_Count to max value
        REPEAT
          Read status of Input_Dev
          Decrement Cycle_Count
        UNTIL Input_Dev is ready OR Cycle_Count = 0
        Input_Char = input from Input_Device
        IF Cycle_Count = 0
          THEN Error_St = $FF END_IF
      END
    ELSE Error_St = status from Input_Dev
  END_IF
END_IF
End Character_Input
```

CPE/EE 421/521 Microcomputers

5

## ACIA example, 68000 assembly language version

- \* ACIA\_Initialize and Character\_Input routine
- \* Data register D0 contains Function  
(zero=initialize, non-zero = get a character)
- \* Data register D0 is re-used for the Cycle\_Count  
(a timeout mechanism)
- \* Data register D1 returns Error\_Status
- \* Data register D2 returns the character from the ACIA
- \* Data register D3 is temporary storage for the ACIA's status
- \* Data register D4 is temporary storage for the masked ACIA's status  
(error bits)
- \* Address register A0 contains the address of the ACIA's  
control/status register
- \*

```
Char_In MOVEM.W D3-D4,-(A7)   Push working registers on the stack
CLR.B   D1                   Start with Error_Status clear
CMP.B   #0,D0                IF Function not zero THEN get input
BNE     InPut                 ELSE initialize ACIA
MOVE.B  #3,(A0)              Reset the ACIA
MOVE.B  #$19,(A0)            Configure the ACIA
BRA     Exit_2                Return after initialization
```

CPE/EE 421/521 Microcomputers

6

## ACIA example, 68000 assembly language version

```
*
InPut  MOVE.W  #$FFFF,D0      Set up Cycle_Count for time-out
                                   (reuse D0)
InPut1 MOVE.B  (A0),D3        Read the ACIA's status register
      MOVE.B  D3,D4          Copy status to D4
      AND.B   #%01111100,D4  Mask status bits to error conditions
      BNE    Exit_1         IF status indicates error, set error
                                   flags & return
      BTST   #0,D3          Test data_ready bit of status
      BNE    Data_Ok       IF data_ready THEN get data
      SUBQ.W #1,D0          ELSE decrement Cycle_Count
      BNE    InPut1        IF not timed out THEN repeat
      MOVE.B #$FF,D1          ELSE Set error flag
      BRA    Exit_2         and return

*
Data_Ok MOVE.B (2,A0),D2     Read the data from the ACIA
      BRA    Exit_2         and return

*
Exit_1  MOVE.B  D4,D1        Return Error_Status
Exit_2  MOVEM.W (A7)+,D3-D4  Restore working registers
      RTS                    Return
```

CPE/EE 421/521 Microcomputers

7

## Passing Parameters via Registers

- Two registers are used in subroutine and have to be saved on the stack:  
    MOVE.W D3-D4, -(A7)  
    (otherwise, data would be lost)
- D0 is simply reused without saving, because the old data will not be needed
- PROS:
  - ❖ Position independent code
  - ❖ Re-entrancy (subroutine has to save registers before they are reused)
- CONS:
  - ❖ Reduces number of registers available to programmer
  - ❖ Number of parameters limited by the number of registers

CPE/EE 421/521 Microcomputers

8

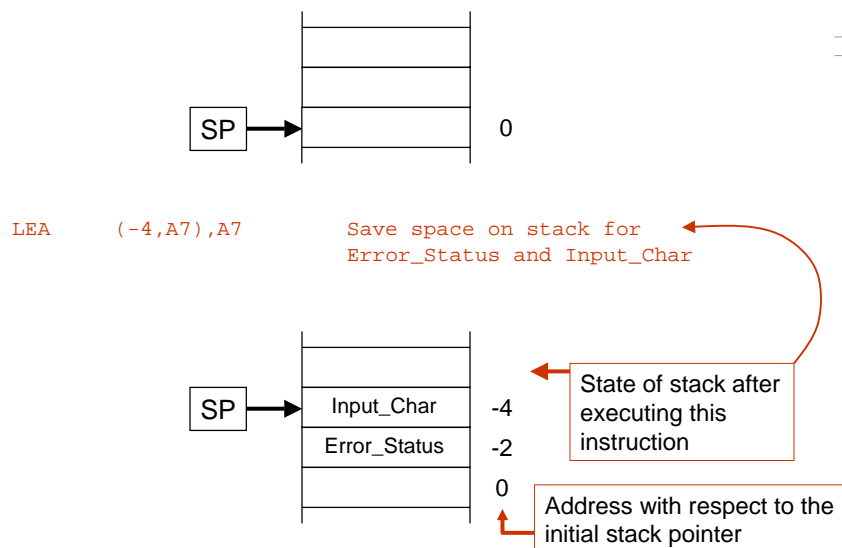
## Mechanisms for Parameter Passing

- Passing parameters by value
  - ❖ Actual parameter is transferred
  - ❖ If the parameter is modified by the subroutine, the "new value" does not affect the "old value"
- Passing parameters by reference
  - ❖ The address of the parameter is passed
  - ❖ There is only one copy of parameter
  - ❖ If parameter is modified, it is modified globally

CPE/EE 421/521 Microcomputers

9

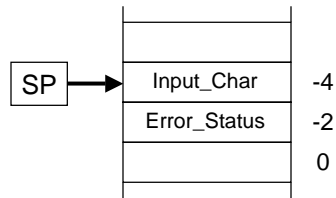
## Passing Parameters by Value



CPE/EE 421/521 Microcomputers

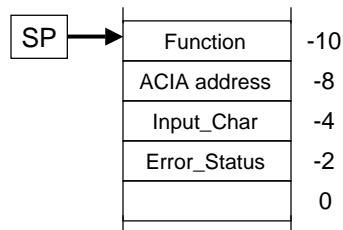
10

### Passing Parameters by Value



```

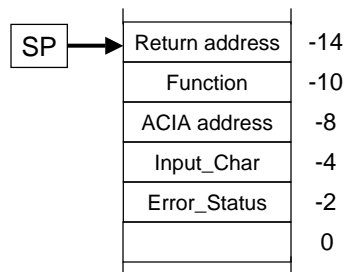
MOVE.L #ACIA,-(A7)      Push ACIA address on the stack
MOVE.W Func,-(A7)      Push function code on the stack
    
```



### Passing Parameters by Value

```

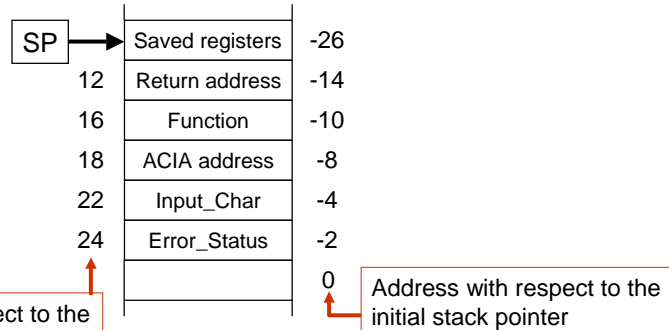
BSR      Char_In      Call subroutine
LEA      (6,A7),A7    Clean up stack - remove parameters
                        Function/ACIA
MOVE.W   (A7)+,Char   Pull the input character off the stack
MOVE.W   (A7)+,Err    Pull the Error_Status off the stack
    
```



## Passing Parameters by Value

\* Character\_Input and ACIA\_Initialize routine  
 \* Data register D3 is temporary storage for the ACIA's status  
 \* Data register D4 is temporary storage for the Cycle\_Count  
 \* Address register A0 contains the address of the ACIA's control/status register  
 \*

```
Char_In MOVEM.L A0/D3-D4,-(A7) Push working registers on the stack
        MOVE.L (18,A7),A0      Read address of ACIA from the stack
        CLR.B (24,A7)         Start with Error_Status clear
```



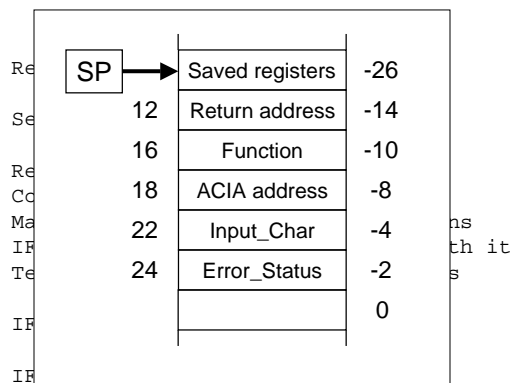
CPE/EE 421/521 Microcomputers

13

## Passing Parameters by Value

```
CMPI.B #0,(16,A7) IF Function not zero THEN get input
BNE InPut ELSE initialize ACIA
MOVE.B #3,(A0)
MOVE.B #19,(A0)
BRA Exit_2
```

```
*
InPut MOVE.W #$FFFF,D0
InPut1 MOVE.B (A0),D3
        MOVE.B D3,D4
        AND.B #%01111100,D4
        BNE Exit_1
        BTST #0,D3
        BNE Data_OK
        SUBQ.W #1,D0
        BNE InPut1
        MOVE.B #$FF,(24,A7)
        BRA Exit_2
```



ELSE Set error flag and return

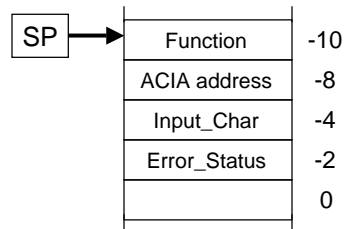
CPE/EE 421/521 Microcomputers

14

## Passing Parameters by Value

```

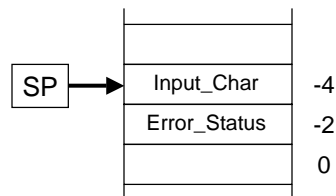
Data_OK MOVE.W (2,A0),(22,A7) Read the data from the ACIA
                                and put on the stack
        BRA      Exit_2          and return
*
Exit_1  MOVE.B  D4,(24,A7)      Return Error_Status
Exit_2  MOVEM.L (A7)+,A0/D3-D4 Restore working registers
        RTS                    Return
    
```



## Passing Parameters by Value

```

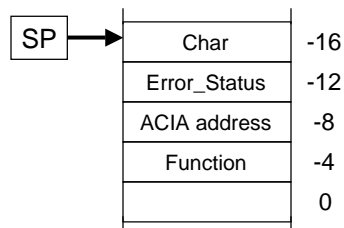
*   BACK TO MAIN PROGRAM :
*
BSR      Char_In          Call subroutine
LEA      (6,A7),A7       Clean up stack - remove parameters
                                Function/ACIA
MOVE.W   (A7)+,Char      Pull the input character off the stack
MOVE.W   (A7)+,Err       Pull the Error_Status off the stack
    
```



## Passing Parameters by Reference

```

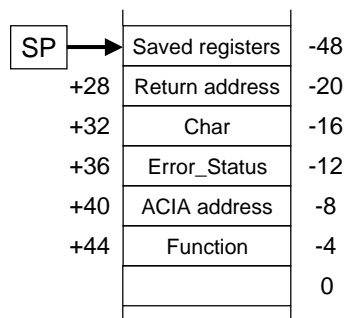
PEA    Func           Push Function address on the stack
PEA    ACIA           Push ACIA address on the stack
PEA    Error_Status   Push address of Error_Status
PEA    Char           Push address of input data
BSR    Char_In        Call subroutine
LEA    (16,A7),A7     Clean up the stack - remove the four
                       addresses
    
```



## Passing Parameters by Reference

```

BSR    Char_In        Call subroutine
LEA    (16,A7),A7     Clean up the stack-remove the 4 addr
*
* D0 is temporary storage for the timeout counter
* D3 is temporary storage for the ACIA's status
* D4 is temporary storage for the Cycle_Count
* A0 points at the location of the character input from the ACIA
* A1 points at the location of the Error_Status
* A2 points at the location of the ACIA
* A3 points at the location of the Function code
*
Char_In MOVEM.L A0-A3/D0/D3-D4,-(A7) Push working regs on the stack
    
```

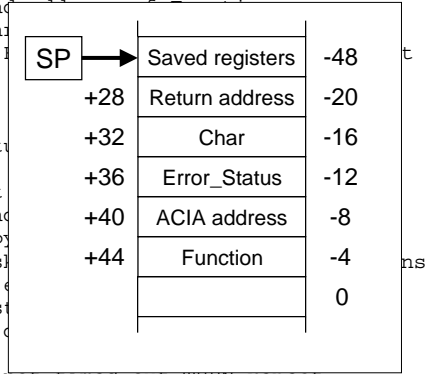


## Passing Parameters by Reference

```

MOVEA.L (32,A7),A0      Read address of Char from the stack
MOVEA.L (36,A7),A1      Read address of Error_Status
MOVEA.L (40,A7),A2      Read address of ACIA from the stack
MOVEA.L (44,A7),A3      Read address of Function from the stack
CLR.B (A1)              Stack pointer
CMPI.B #0,(A3)          IF not timed out THEN repeat
BNE InPut               ELSE Set error flag and return
MOVE.B #3,(A2)          Read Error_Status
MOVE.B #19,(A2)         Read Error_Status
BRA Exit_2              Return Error_Status

*
InPut MOVE.W #FFFF,D0    Set error flag
InPut1 MOVE.B (A2),D3    Read Error_Status
MOVE.B D3,D4            Copy Error_Status to D4
AND.B #01111100,D4     Mask Error_Status
BNE Exit_1              IF Error_Status is not zero
BTST #0,D3              Test Error_Status
BNE Data_OK             IF Error_Status is not zero
SUBQ.W #1,D0            Decrement error flag
BNE InPut1             IF error flag is not zero
MOVE.B #FF,(A1)        ELSE Set error flag and return
BRA Exit_2              Return Error_Status
    
```



CPE/EE 421/521 Microcomputers

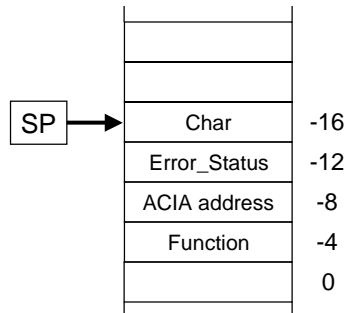
19

## Passing Parameters by Reference

```

Data_OK MOVE.W (2,A2),(A0)  Read the data from the ACIA
        BRA Exit_2

*
Exit_1 MOVE.B D4,(A1)       Return Error_Status
Exit_2 MOVEM.L (A7)+,A0-A3/D0/D3-D4 Restore working registers
        RTS
    
```



CPE/EE 421/521 Microcomputers

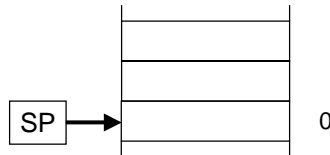
20

## Passing Parameters by Reference

```

* Back to main program
* ...
  BSR    Char_In    Call subroutine
  LEA    (16,A7),A7 Clean up the stack-remove the 4 addr

```



CPE/EE 421/521 Microcomputers

21

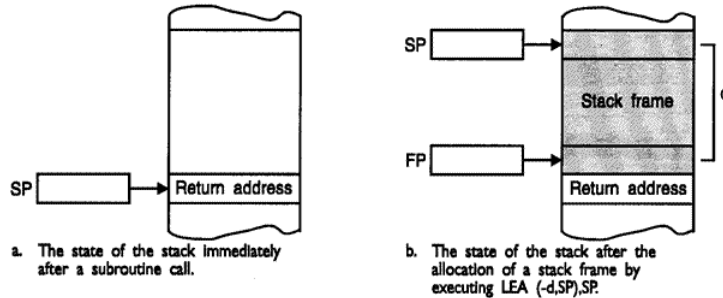
## The Stack and Local Variables

- Subroutines often need local workspace
- We can use a fixed block of memory space – *static allocation* – but:
  - ❖ The code will not be relocatable
  - ❖ The code will not be reentrant
  - ❖ The code will not be able to be called recursively
- Better solution: *dynamic allocation*
  - ❖ Allocate all local variables on the stack
  - ❖ **STACK FRAME** = a block of memory allocated by a subroutine to be used for local variables
  - ❖ **FRAME POINTER** = an address register used to point to the stack frame

CPE/EE 421/521 Microcomputers

22

## The Stack and Local Variables



It can be done simply by modifying the stack pointer:

```
AnySub  LEA  (-4,A7),A6    Set up A6 as the frame pointer
        LEA  (-200,A7),A7  Create the stack frame
        .
        .                  The subroutine proper
        .
        LEA  (200,A7),A7   Collapse the stack frame
        RTS                and return from subroutine
```

## The Stack and Local Variables

➤ **LINK** and **UNLK** automate the creation and removal of the stack frame

```
Sub1  LINK  A1,#-64    Allocate 64 bytes (16 long words) of storage
        .              in this stack frame - use A1 as frame pointer
        .
        .              Body of the subroutine
        .
        UNLK  A1        De-allocate Subroutine 1's stack frame
        RTS            Return to calling point.
```

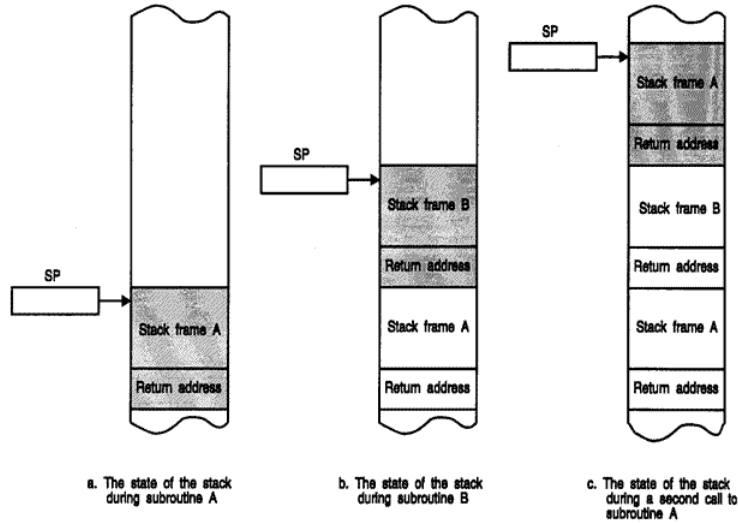
➤ Implementation

```
LINK:  [SP]    ← [SP] - 4    Decrement the stack pointer by 4
        [M([SP])] ← [A1]    Push the contents of address register A1
        [A1]    ← [SP]      Save stack pointer in A1
        [SP]    ← [SP] - 64  Move stack pointer up by 64 locations
```

```
UNLK:  [SP] ← [A1]
        [A1] ← [M([SP])]
        [SP] ← [SP] + 4
```

## The Stack and Local Variables

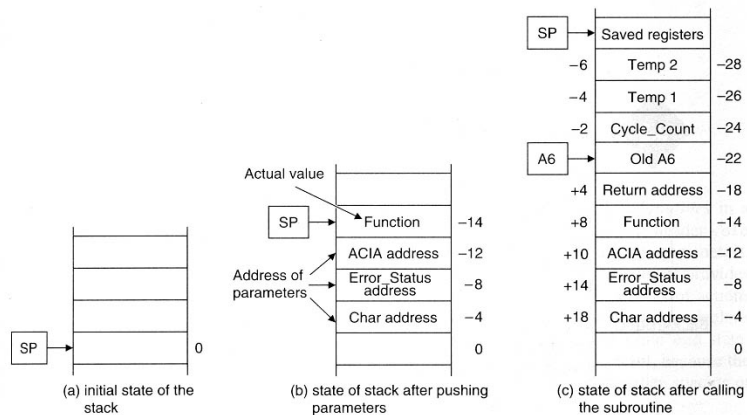
Nested subroutines: A calls B, then B calls A



CPE/EE 421/521 Microcomputers

25

PEA Char Push address of dest. for the input  
 PEA Error\_Status Push address of Error\_Status message  
 PEA ACIA Push ACIA's address on the stack  
 MOVE.W Function,-(A7) Push value of function code on the stack  
 BSR Char\_In Call subroutine  
 LEA (14,A7),A7 Clean up the stack - remove the four parameters



CPE/EE 421/521 Microcomputers

26

```

* Character_Input and ACIA_Initialize routine
* SF location A6 - 6 holds the ACIA's status
* SF location A6 - 4 holds the ACIA's masked status (error bits only)
* SF location A6 - 2 holds the Cycle_Count
* A1 contains the address of the Error_Status
* A2 contains the address of the ACIA's control register
*
Char_In LINK    A6,#-6        Create a link
             MOVEM.L A1-A2,-(A7)  Push work
             MOVEA.L (14,A6),A1    Read address
             MOVEM.L (10,A6),A2    Read address
             CLR.B (A1)           Clear Error
             MOVE.W #$FFFF,(-2,A6) Set up Cycle
             CMPI.B #0,(8,A6)     IF Function
             BNE InPut           ELSE
             MOVE.B #3,(A2)       Reset ACIA
             MOVE.B #$19,(A2)     Configure
             BRA Exit_2           Return after

```

(c) state of stack after calling the subroutine

words  
stack  
om  
tack  
input

```

InPut  MOVE.B (A2),(-4,A6)  Read the ACIA's status register -
             save in Temp1
             MOVE.B (-4,A6),(-6,A6) Copy status
             ANDI.B #%01111100,(-6,A6) Mask
             BNE Exit_1     IF status
             BTST #0,(-4,A6) ELSE Test
             BNE Data_OK   IF data
             SUBQ.W #1,(-2,A6)
             BNE InPut     IF not
             MOVE.B #$FF,(A1)
             BRA Exit_2

```

(c) state of stack after calling the subroutine

tus  
nt  
flag

Data_OK	MOVE.L (18,A6),(A1)	Get address for data dest.
*		(reuse A1)
	MOVE.B (2,A2), (A1)	Read data from ACIA
	BRA Exit_2	
*		
Exit_1	MOVE.B (-6,A6),(A1)	Return Error_Status
Exit_2	MOVEM.L (A7)+,A1-A2	Restore working registers
	UNLK A6	
	RTS	

## C and The 68000

- Compiler and 68000 instruction set
- C data types and implementation
- Storage classes
- Functions and parameters
- Pointers

## Compiling a C Program

<pre>void main (void) {     int i;     int j;     i = 1;     j = 2;     i = i + j; }</pre>	<pre>* Comments SECTION S_main,,"code" XREF __main     * Variable i is at -2(A6)     * Variable j is at -4(A6) XDEF __main __main LINK A6,#-4     *2 {     *3 int i;     *4 int j;     *5 i = 1; MOVE #1,-2(A6)     *6 j = 2; MOVE #2,-4(A6)     *7 i = i + j; MOVEQ.L #1,D1 ADDQ #2,D1 MOVE D1,-2(A6)     *8 } UNLK A6 RTS</pre>
--	---

## C Data Types

- The 68000 family supports three basic data types:
  - ❖ Byte, Word, Longword
  - ❖ Each can be interpreted as signed or unsigned
- C built-in types:
  - ❖ Integer, character, floating point, double-precision
  - ❖ Void – refers to the null data type
  - ❖ Implementation dependant!

Data type	C name	Width (b)	Range
integer	int	16	-32768 to 32767
short integer short	int	8	-128 to 127
long integer	long int	32	-2147483648 to 2147483647
unsigned integer	unsigned int	16	0 to 65535
character	char	8	0 to 255
single-precision floating point	float	32	10 <sup>-38</sup> to 10 <sup>+38</sup>
double-precision floating point	double	64	10 <sup>-300</sup> to 10 <sup>+300</sup>

## C Data Types, cont'd

- Local variables
  - ❖ Defined inside a function
  - ❖ Cannot be accessed from outside the function
  - ❖ Normally lost when a return from the function is made
- Global variables
  - ❖ Defined outside a function
  - ❖ Can be accessed both from inside and outside the function
- Variables defined in a block exist only within that block

```
int i; /*global variable, visible to everything from this point*/
void function_1(void) /*A function with no parameters*/
{
    int k; /*Integer k is local to function_1*/
    {
        int q; /*Integer q exists only in this block*/
        int j; /*Integer j is local and not the same as j in main*/
    }
}
void main(void)
{
    int j; /*Integer j is local to this block within function main*/
} /*This is the point at which integer j ceases to exist*/
```

CPE/EE 421/521 Microcomputers

33

## Storage Class

- Storage class specifiers
  - ❖ **auto**
    - Variable is no longer required once a block has been left; Default
  - ❖ **register**
    - Ask compiler to allocate the variable to a register
    - Also is automatic
    - Cannot be accessed by means of pointers
  - ❖ **static**
    - Allows local variable to retain its value when a block is reentered
    - Initialized only once, by the compiler!
  - ❖ **extern**
    - Indicates that the variable is defined outside the block
    - The same global variable can be defined in more than one modul

CPE/EE 421/521 Microcomputers

34

## Storage Class, cont'd

### ➤ Access Modifiers

#### ❖ **volatile**

- To define variables that can be changed externally
- Compiler will not put them in registers
- Think about Status Registers !

#### ❖ **const**

- Variable may not be changed during the execution of a program
- Cannot be changed unintentionally, but CAN be changed externally (as a result of an I/O, or OS operations external to the C program)

### ➤ Type conversion

#### ❖ In C, done either automatically or explicitly (casting)

```
X DS.L 1 Reserve a longword for X
Y DS.W 1 Reserve a word for Y
```

#### USUALY WRONG

```
MOVE.L X,D0
ADD.W Y,D0
```

#### CORRECT

```
MOVE.W Y,D0
EXT D0
ADD.L X,D0
```

CPE/EE 421/521 Microcomputers

35

## Returning a Value from a Function

### ➤ Example: **main** calls function **adder**

- ❖ **adder** function has 2 formal parameters (x and y)
- ❖ Formal parameters behave like local variables within the function
- ❖ When the function is called, formal parameters are replaced by the values of the actual parameters (a and b)

```
int adder(int x, int y) /* returns an integer */
{
    return x + y; /* return sum of x and y to the calling program */
}

void main (void)
{
    register int a, b, c; /* assign variables a, b, and c to regs */
    a = 1; b = 2; /* provide some dummy values for a and b */
    c = adder(a, b); /* c is assigned the integer returned by adder */
}
```

CPE/EE 421/521 Microcomputers

36

## Returning a Value from a Function, cont'd

```

*1 int adder(int x, int y)
* Parameter x is at 8(A6)
* Parameter y is at 10(A6)
_ader
  LINK A6,#0
*2 {
*3 return x + y;
  MOVE 8(A6),D1
  ADD 10(A6),D1
  MOVE D1,D0
*4 }
  UNLK A6
  RTS
    
```

```

*5 void main (void)
* Variable a is at -2(A6)
* Variable b is at -4(A6)
* Variable c is at -6(A6)
_main
  LINK A6,#-6
*6 {
*7 int a, b, c;
*8 a = 1, b = 2;
  MOVE #1,-2(A6)
  MOVE #2,-4(A6)
*9 c = adder(a, b);
  MOVE #2,-(A7) } Not taken from
  MOVE #1,-(A7) the stack frame
  JSR _ader
  MOVE D0,-6(A6)
*10 }
  UNLK A6
  RTS
    
```

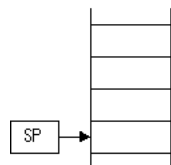
Parameters accessed from the **main's** stack frame

a and b are pushed on stack prior to the function call

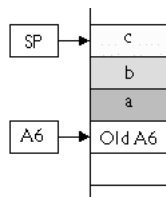
CPE/EE 421/521 Microcomputers

37

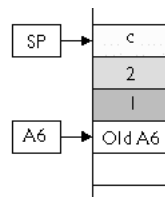
## Returning a Value from a Function USE OF STACK



a. Initial state of the stack



b. The stack after LINK A6,#-6



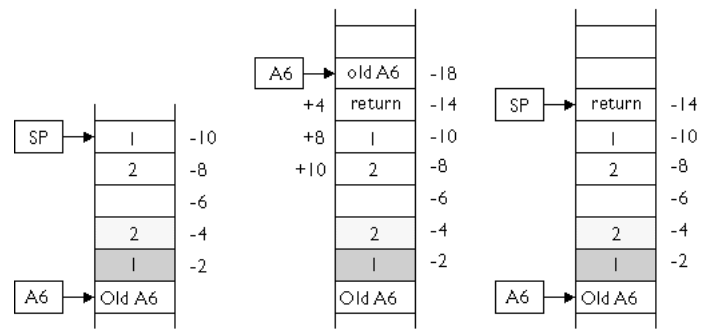
c. Stack after setting up data  
MOVE #1,-2(A6)  
MOVE #2,-4(A6)

Figure 3.9

CPE/EE 421/521 Microcomputers

38

### Returning a Value from a Function USE OF STACK, cont'd



d. Stack after copying data to stack with  
MOVE #2,-(A7)  
MOVE #1,-(A7)

e. Stack after calling subroutine and creating a stack frame with  
JSR \_adder  
LINK A6,#0

f. Stack after collapsing stack frame with  
UNLK A6

Figure 3.9

### Pointers and C

- C is *pointer-oriented*
- Pointers in 68000 assembly language: (Ai)
- Example:

C Code	68000 code	comment
x=*y;	MOVE (A0),D0	x is in D0, y is in A0
a=&b;	LEA B,A0	a is in A0

- Pointer Arithmetic

```

char x='A';
int y=0;
register char *P_x=&x;
register int *P_y=&y;
P_xx++;
P_yy++;

LINK A6,#-4 /*x:-1(A6),y:-4(A6)*/
MOVE.B #65,-1(A6)
CLR -4(A6)
LEA.L -1(A6),A3
LEA.L -4(A6),A2
ADDQ #1,A3
ADDQ #2,A2
UNLK A6
RTS
    
```

## Pointers and C, cont'd

```
void main(void)
{
  int x;
  int *P_port; /*pointer*/

  P_port = (int*) 0x4000;

  /* wait for port ready */
  do { }
  while
    ((*P_port&0x0001)==0);

  x = *(P_port + 1);
  /* read from 2 bytes
  beyond port */
}
```

```
*1 main()
* Variable x is at -2(A6)
* Variable P_port is at -6(A6)
_main
LINK A6,#-6
*2 {
*3 int x;
*4 int *P_port;
*5 P_port = (int*) 0x4000;
MOVE.L #16384,-6(A6)
*6 do { }
*7 while ((*P_port&0x0001)==0);
L1 MOVEA.L -6(A6),A4
MOVE (A4),D1
ANDI #1,D1
BEQ.S L1
*7 x = *(P_port + 1);
MOVE 2(A4),-2(A6)
*8 }
UNLK A6
RTS
```

## Functions and Parameters

- Passing parameters to a function
- Passing by value/reference
- Is this going to work?

```
/* this function swaps the values of a and b */
void swap (int a, int b) {
  int temp;
  /* copy a to temp, b to a, and temp to b */
  temp = a;
  a = b;
  b = temp;
}
void main (void) {
  int x = 2, y = 3;
  swap (x, y); /* let's swap a and b */
}
```

No, because this program is using a *call-by-value* mechanism

## Functions and Parameters, cont'd

```

*1 void swap (int a, int b)
* Parameter a is at 8(A6)
* Parameter b is at 10(A6)
* Variable temp is at -2(A6)
_swap
LINK A6,#-2
*2 {
*3 int temp;
*4 temp = a;
MOVE 8(A6),-2(A6)
*5 a = b;
MOVE 10(A6),8(A6)
*6 b = temp;
MOVE -2(A6),10(A6)
*7 }
UNLK A6
RTS

*8 void main (void)
* Variable x is at -2(A6)
* Variable y is at -4(A6)
_main
LINK A6,#-4
*9 {
*10 int x = 2, y = 3;
MOVE #2,-2(A6)
MOVE #3,-4(A6)
*11 swap (x, y);
MOVE #3,-(A7)
MOVE #2,-(A7)
JSR _swap
*12 }
UNLK A6
RTS

```

CPE/EE 421/521 Microcomputers

43

## Functions and Parameters USE OF STACK – *call-by-value*

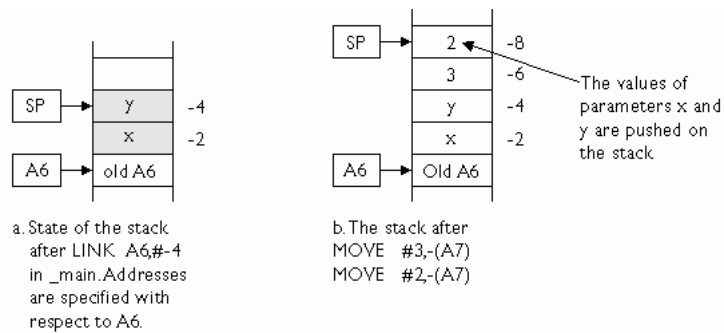


Figure 3.11

CPE/EE 421/521 Microcomputers

44

## Functions and Parameters

### USE OF STACK – *call-by-value*, cont'd

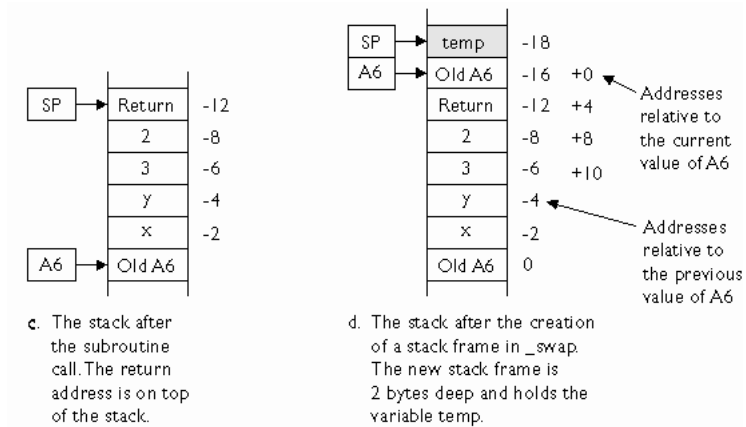


Figure 3.11

## Functions and Parameters

### *Call-by-reference*

- To permit the function to modify the parameters, pass the address of the parameters
- This will work...

```

/* swap two parameters in the calling program */
void swap (int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
void main (void) {
    int x = 2, y = 3;
    /* call swap and pass the addresses of the parameters */
    swap(&x, &y);
}

```

## Functions and Parameters *Call-by-reference, cont'd*

```

*1 void swap (int *a, int *b)
* Parameter a is at 8(A6)
* Parameter b is at 12(A6)
* Variable temp is at -2(A6)
_swap
LINK A6,#-2
*2 {
*3 int temp;
*4 temp = *a;
MOVEA.L 8(A6),A4
MOVE (A4),-2(A6)
*5 *a = *b;
MOVEA.L 12(A6),A0
MOVE (A0),(A4)
*6 *b = temp;
MOVEA.L 12(A6),A4
MOVE -2(A6),(A4)
*7 }
UNLK A6
RTS

*8 main ()
* Variable x is at -2(A6)
* Variable y is at -4(A6)
_main
LINK A6,#-4
*9 {
*10 int x = 2, y = 3;
MOVE #2,-2(A6)
MOVE #3,-4(A6)
*11 swap (&x, &y);
PEA.L -4(A6)
PEA.L -2(A6)
JSR _swap
*12 }
UNLK A6
RTS

```

CPE/EE 421/521 Microcomputers

47

## Functions and Parameters *USE OF STACK, Call-by-reference*

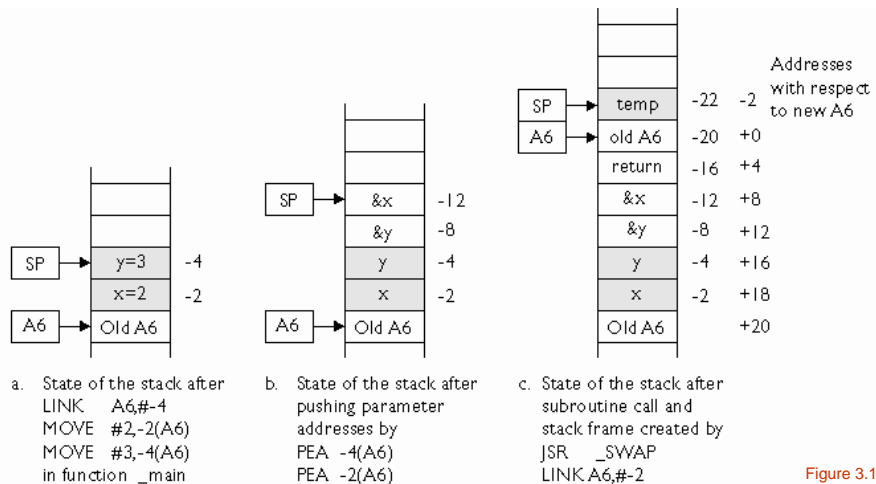


Figure 3.12

CPE/EE 421/521 Microcomputers

48