v.23-04-2024

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΑΘΗΜΑ
ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ (206ΕΥΥΚ)
ΠΠΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΕΑΡΙΝΟ 2023-2024

Διάλεξη No7:
M68000: Υπορουτίνες - Subroutines

Δ. Καραμπατζάκης, Επίκουρος Καθηγητής

email. dkara@cs.ihu.gr

Δήλωση προσβασιμότητας

Σε αυτό το μάθημα όλες/οι οι φοιτήτριες/τές απολαμβάνουν – και αντίστοιχα υποχρεούνται να σέβονται – το δικαίωμα της ίσης μεταχείρισης. Δεν είναι ανεκτή και αποδεκτή κανενός τύπου και μορφής διάκριση με κριτήρια την εθνικότητα, τη φυλή, την καταγωγή, τη γλώσσα, το φύλο, τη θρησκεία, την ηλικία, την υγεία, τη σωματική ικανότητα, την ιδιωτική ζωή, τον γενετήσιο προσανατολισμό, τη σωματική ικανότητα και την οικονομική και κοινωνική κατάσταση στην οποία αυτοί βρίσκονται.

Το Πανεπιστήμιο άγρυπνα μεριμνά για τη διασφάλιση της αρχής των ίσων ευκαιριών και της ίσης μεταχείρισης. Οι κοινωνικές προκαταλήψεις και οι ιδεολογικές παρωπίδες είναι έννοιες τελείως ξένες με την επιστημονική πρόοδο την οποία το Πανεπιστήμιο είναι ταγμένο να υπηρετεί.

Ο Διδάσκων

Πληροφορίες για το Μάθημα

Διδάσκων:

Δημήτρης Καραμπατζάκης, Επίκουρος Καθηγητής

Αναλογικά και Ψηφιακά Ηλεκτρονικά Συστήματα

Μέλος Εργαστηρίου Βιομηχανικών και Εκπαιδευτικών Ενσωματωμένων Συστημάτων

Επικοινωνία / πληροφορίες:

Email. dkara@cs.ihu.gr

web. http://www.internetofthings.gr/

Ώρες Γραφείου:

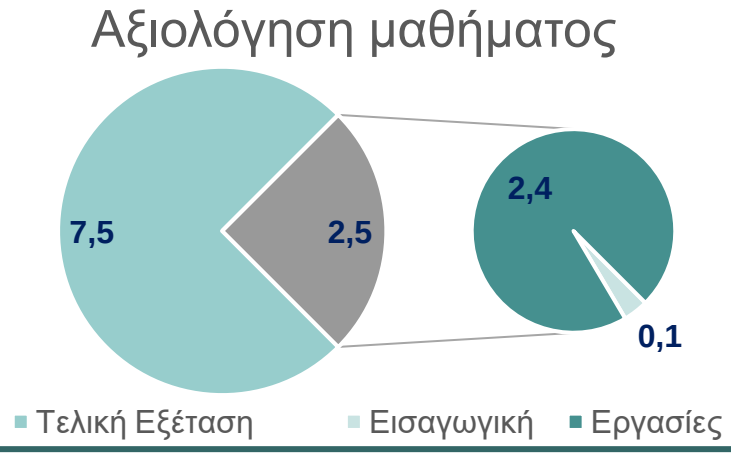
μετά από συνεννόηση με email στο ΦΕ 315 (πάνω από αιθ. Α1)

Πληροφορίες για το Μάθημα (Γενικές)

- **Κάθε Τρίτη, Πέμπτη 12.00 π.μ. - 14.00 μ.μ. μάθημα θεωρίας στο Μεγάλο Αμφιθέατρο (μπορεί να αλλάζει με ανακοινώσεις).**
- **Η διαχείριση του μαθήματος θα γίνει με χρήση της υπηρεσίας https://courses.cs.ihu.gr**
- **Όλοι οι φοιτητές πρέπει να έχουν λογαριασμό στο uregister.**
- **Η ιστοσελίδα με τις πληροφορίες του μαθήματος: http://iees.cs.ihu.gr/?page_id=3209**
- **Υλικό του μαθήματος στο moodle: https://moodle.cs.ihu.gr/**

Πληροφορίες για το Μάθημα (Αξιολόγηση)

- **Η βαθμολογία είναι 75% από την τελική εξέταση και 25% από τις ατομικές εργασίες (1 σετ ασκήσεων) που θα δοθούν για το σπίτι.**
- **Η τελική εξέταση είναι με ανοιχτό το κύριο σύγγραμμα του μαθήματος.**
- **Ο βαθμός του μαθήματος (ΒΜ = ΓΕ*0,75 + ΣΑ*0,25) πρέπει να είναι τουλάχιστον πέντε (5).**



Πληροφορίες για το Μάθημα (Μονάδες)

- **Κωδικός Μαθήματος:** 206ΕΥΥΚ
- **Εξάμηνο:** 2ο
- **Τύπος Μαθήματος:** Υποβάθρου, Ανάπτυξης Δεξιοτήτων
- **Είδος Μαθήματος:** Υποχρεωτικό (ΥΠ)
- **Διδασκαλία Θεωρίας:** 3 ώρες/εβδομάδα
- **Διδασκαλία Φροντιστήριο:** 1 ώρες/εβδομάδα
- **Πιστωτικές μονάδες ECTS: 7**
- **Γλώσσα διδασκαλίας και Εξετάσεων:** Ελληνικά

Πληροφορίες για το Μάθημα (Φόρτος)

Δραστηριότητα	Φόρτος εργασίας εξαμήνου
Διαλέξεις	78 ώρες
Φροντιστηριακές Ασκήσεις	26 ώρες
Γραπτές Εξετάσεις	2 ώρες
Γραπτές Εργασίες	34 ώρες
Αυτοτελής Μελέτη	35 ώρες
Σύνολο	**175 ώρες (7 ECTS)**

Κύριο Σύγγραμμα Μαθήματος (ΕΥΔΟΞΟΣ)

Οργάνωση και Σχεδίαση Υπολογιστών

Συγγραφέας: Πογαρίδης Δημήτριος

Έτος Έκδοσης: 2019

Κωδικός στον Εύδοξο: **86192986**

Λογισμικό - Αναπτυξιακό

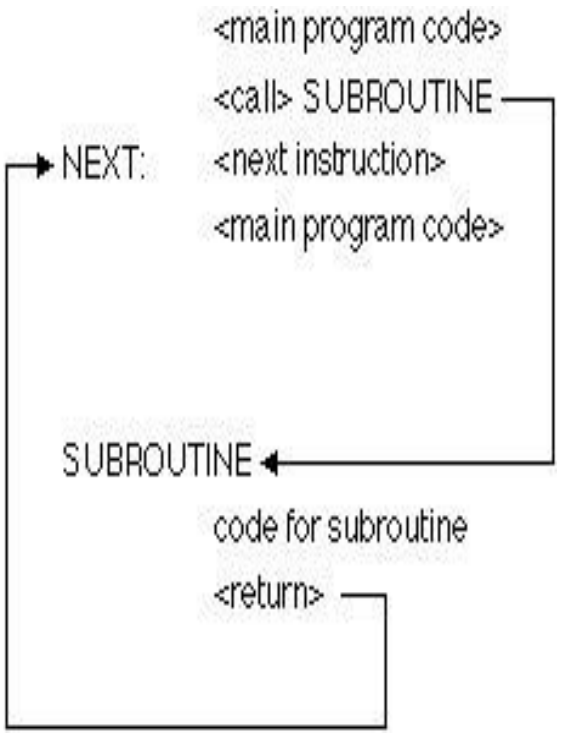
- **Α’ μέρος μαθήματος (CISC):**
  - Assembly για τον Motorola68000
  - Λογισμικό easy68k http://www.easy68k.com/
- **Β’ μέρος μαθήματος (RISC):**
  - Υλοποίηση σχεδιάσεων σε αναπτυξιακό Arduino (προαιρετική αγορά του υλικού σύμφωνα με τις οδηγίες)
  - Λογισμικό Arduino IDE https://www.arduino.cc/en/Main/Software
  - Η γλώσσα προγραμματισμού (C++) και οι εντολές που υποστηρίζει είναι διαθέσιμες στο: https://www.arduino.cc/reference/en/

M68000
Υπορουτίνες - Subroutines

A subroutine (aka function) is a piece of code that can be called from some point in a program to carry out a specific task.

Subroutines Basics

- A subroutine is a sequence of, usually, consecutive instructions that carries out a single specific function or a number of related functions needed by calling programs.
- A subroutine can be called from one or more locations in a program.
- Subroutines may be used where the same set of instructions sequence would otherwise be repeated in several places in the program.



Programming Subroutines

- Why use subroutines?
  - Code re-use
  - Easier to understand code (readability)
  - Divide and conquer
    - Complex tasks are easier when broken down into smaller tasks
  - Simplify the code debugging process.
- How do we call a subroutine in assembly?
  - Place the parameters somewhere known
  - JSR or BSR to jump to the subroutine
  - RTS to return

C

```
main() {
   int a, b;
   a = 5;
   b = sqr(a);
   printf("%d\n" b);
}

/* subrtn sqr */
int sqr(int val) {
   int sqval;
   sqval = val * val;
   return sqval;
}
```

Assembly

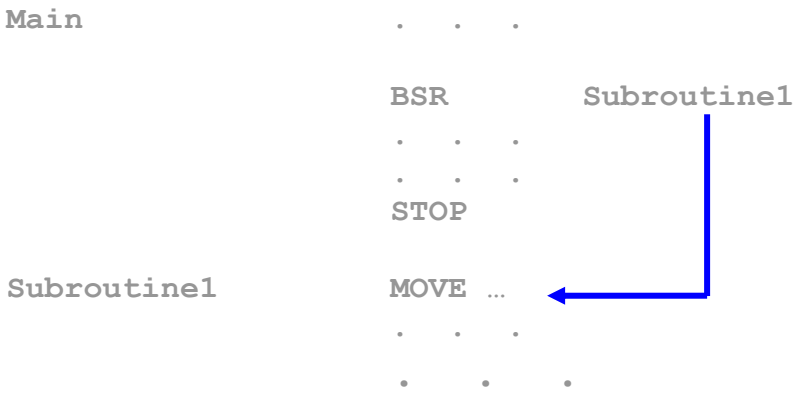
```
MAIN        MOVE.W      A,D1
            JSR         SQR
            MOVE.W      D0,B
            STOP        #$2700
;*** subroutine SQR ***
SQR         MULU.W      D1,D1
            MOVE.W      D1,D0
            RTS
;*** data area ***
            ORG         $2000
A           DC.W        5
B           DS.W        1
            END MAIN
```

68000 Subroutine Calling Instructions

```
BSR  <subroutine_label>
```

- BSR = Branch to SubRoutine
- **`subroutine_label`** is the address label of the first instruction of the subroutine.
- **`subroutine_label`** must be within no more than a 16-bit signed 2s complement offset, i.e. within plus or minus 32K of the BSR instruction.
- Supports Short branch (8 bit – 2 bytes, one opcode one offset) and Long branch (16 bit – 4 bytes, two opcode two offset).
- Does not affect CCR

Example:



68000 Subroutine Calling Instructions

JSR <ea>

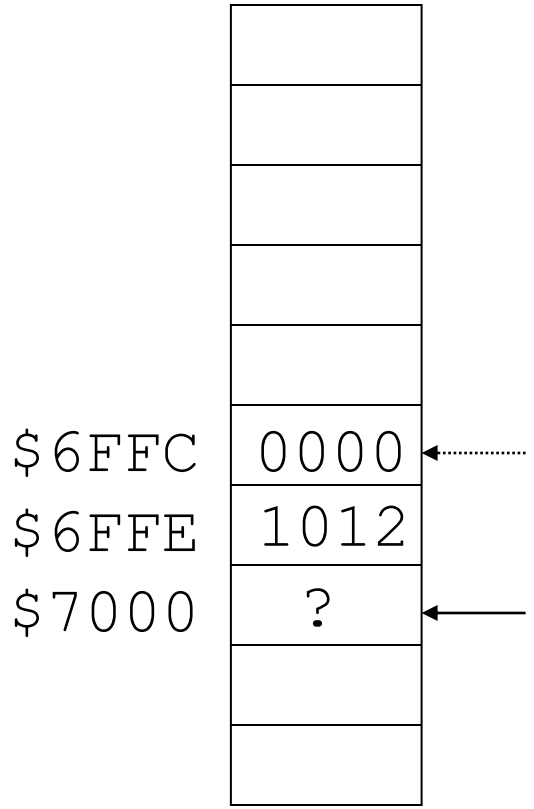
Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	(xxx).W	(xxx).L	#<data>	d(PC)	d(PC,Rn)
		✔			✔	✔	✔	✔		✔	✔

- JSR = Jump to SubRoutine
- Similar in functionality to BSR, addressing mode <ea> must be a memory addressing mode.
  - i.e. <EA> cannot be a data or address register.
- The advantages of this instruction:
  - A number of different addressing modes are supported.
  - The address of the subroutine can be computed dynamically at execution time using ARI addressing modes
    - Allows the selection of the subroutine to call at runtime
  - JSR does not affect CCR
- Supports longword branch (6 bytes, two for opcode & four Address).
- JSR is the most common form used for calling a subroutine.

JSR Example

- What JSR *label* does (using absolute mode):
  1. Decrement SP by 4.
  2. Save current PC on top of stack.
  3. Jump to subroutine.
- In other words:
  1. SP ← [SP] - 4
  2. [SP] ← [PC]
  3. PC ← *address-of-label*
- BSR *label*
  - Same, but offset from the current PC is stored instead of the absolute address

```
1000                    MOVE.L          #5,D1
1006                    LEA             ARRAY,A0
100C                    JSR             SUMARR
1012                    MOVE.L          D0,SUM
1018                    STOP            #$2700
101E                    NOP
1020        SUMARR      CLR.L           D0
                        …
                        RTS
            ARRAY       DC.L            12,15,31
            SUM         DS.L            1
                        END
```



	Before	After
SP	00007000	00006FFC
PC	0000100C	00001020

JSR vs BSR

- JSR *label* does:
  1. Decrement SP by 4
  2. Save current PC on top of stack
  3. Jump to subroutine. New PC can be derived using absolute mode and several address register indirect mode.
- In other words:
  1. SP ← [SP] – 4
  2. [SP] ← [PC]
  3. PC ← *<ea>*

- BSR *label* does:
  1. Decrement SP by 4
  2. Save current PC on top of stack
  3. Branch to subroutine. New PC is computing using current PC and offset provided by instruction.
- In other words:
  1. SP ← [SP] – 4
  2. [SP] ← [PC]
  3. PC ← *PC + offset*

68000 Subroutine Return Instruction

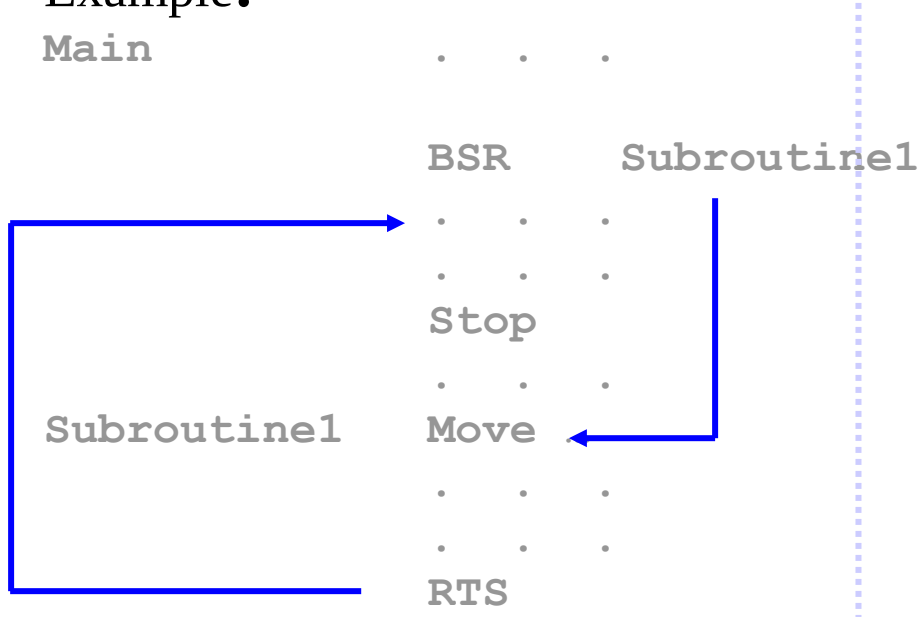
RTS

- RTS = ReTurn from Subroutine
- Pops off the long word (return address) of the top of the stack and puts it in the program counter in order to start executing after the point of the subroutine call.
- Post increments the stack pointer SP by 4
- Equivalent to the set of instructions:

  MOVEA.L (SP)+,A0

  JMP (A0)
- Does not affect CCR

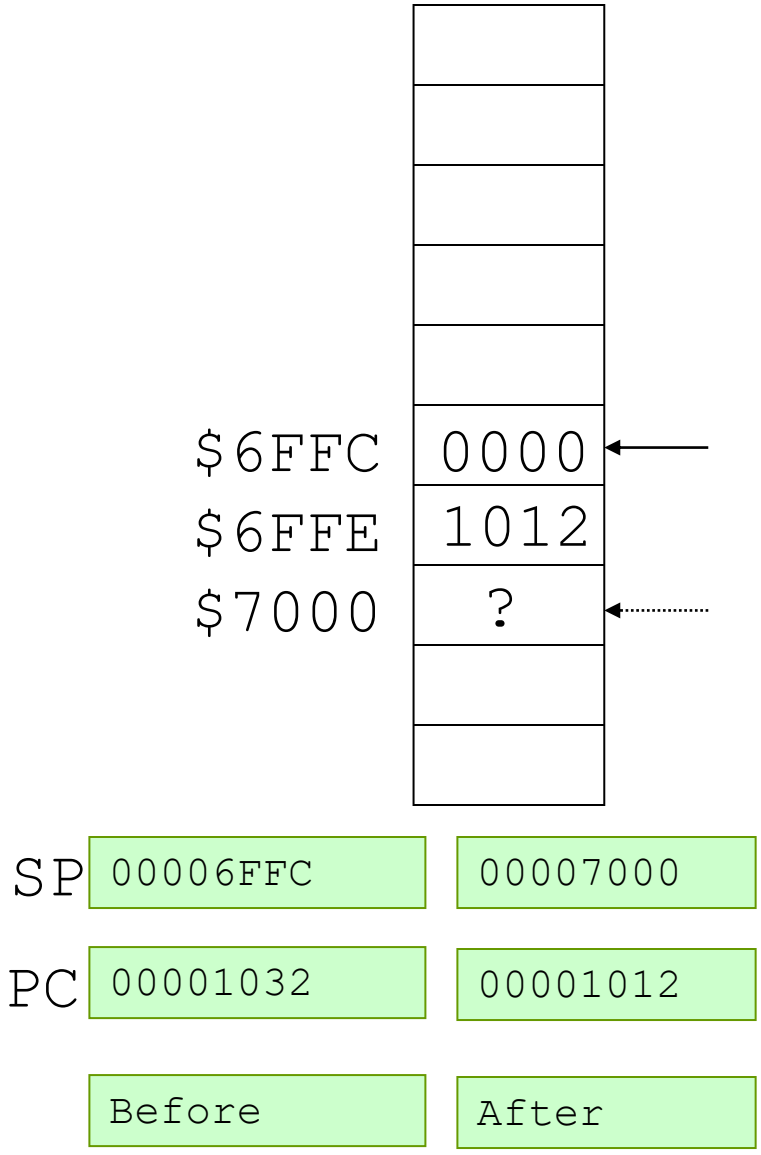
Example:



RTS Example

- What RTS does:
  - *Pop the address from the stack back into the PC*
  - MOVE.L (SP)+, PC
- In other words:
  - PC ← [SP]
  - [SP] ← [SP] + 4

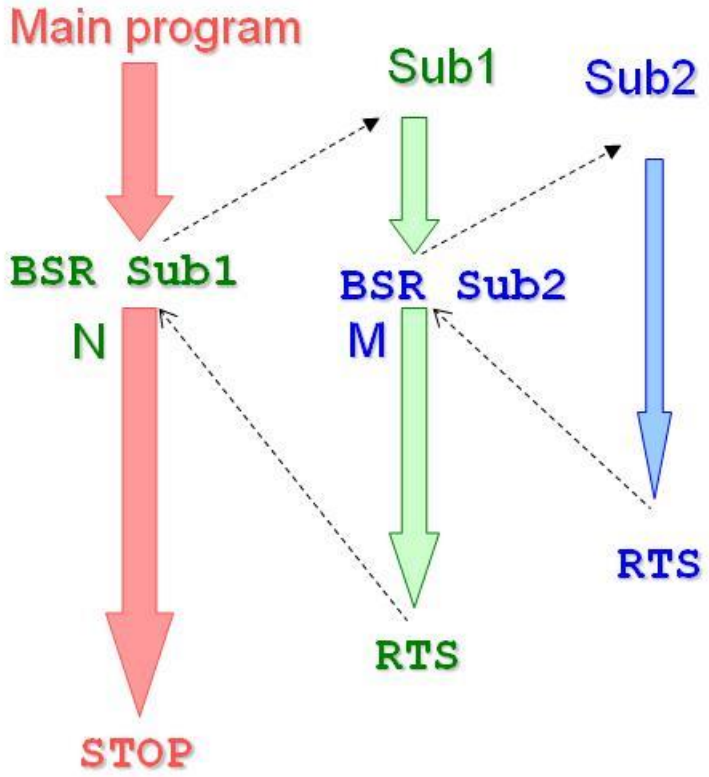
```
1000                    MOVE.L      #5,D1
1006                    LEA         ARRAY,A0
100C                    JSR         SUMARR
1012                    MOVE.L      D0,SUM
1018                    STOP        #$2700
101E                    NOP
1020        SUMARR      CLR.L       D0
                        …
1032                    RTS
            ARRAY       DC.L        12,15,31
            SUM         DS.L        1
                        END
```



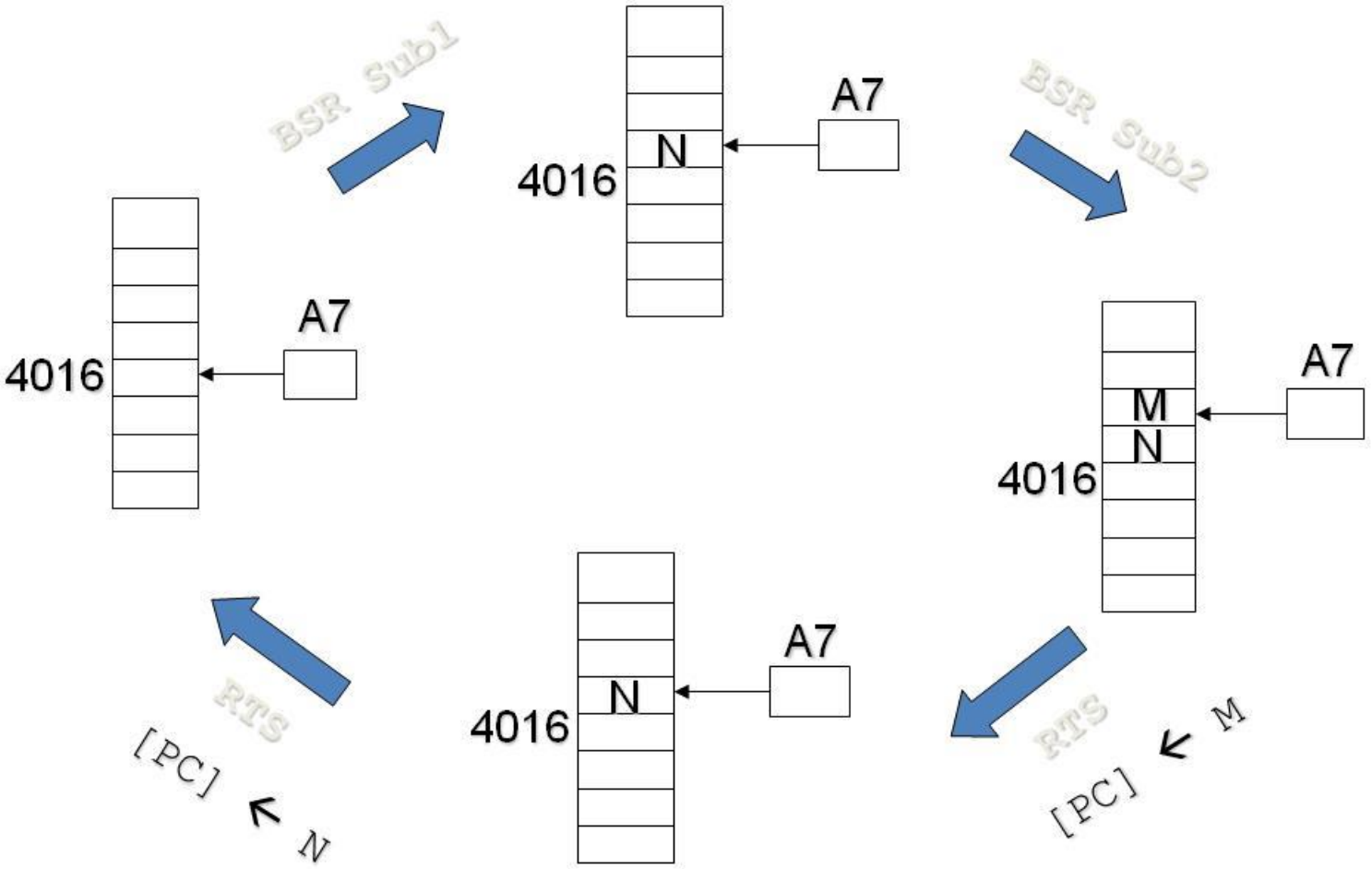
Nested Subroutines

```
* Main Program
        ...
        BSR Sub1
N:      ...
        ...
        STOP #$2700
Sub1:   ...
        BSR Sub2
M:      ...
        ...
        RTS

Sub2    ...
        RTS
```



Nested Subroutines



Passing Parameters High Level Programming Langs

Passing by value:

```
...
n=5;
sub(n);
printf("%d",n);
...

void sub (int n) {
   n++;
}
```

The output here will be 5

Passing by reference:

```
...
n=5;
sub(&n);
printf("%d",n);
...

void sub (int* n) {
   (*n)++;
}
```

The output here will be 6

Passing Parameters to Subroutines

- **Parameters may be passed to a subroutine by using:**
  - **Data and Address Registers:**
    - **Efficient, position-independent.**
    - **It reduces the number of registers available for use by the programmer.**
  - **Memory locations:**
    - **This is similar to using static or global data in high level languages.**
    - **Does not produce position independent code and may produce unexpected side effects.**
  - **Stacks:**
    - **This is the standard, general-purpose approach for parameter passing. The LINK and UNLK instructions may be used to create and destroy temporary storage on the stack.**
    - **Similar to the approach used by several high-level languages including C.**

Writing *Transparent* Subroutines

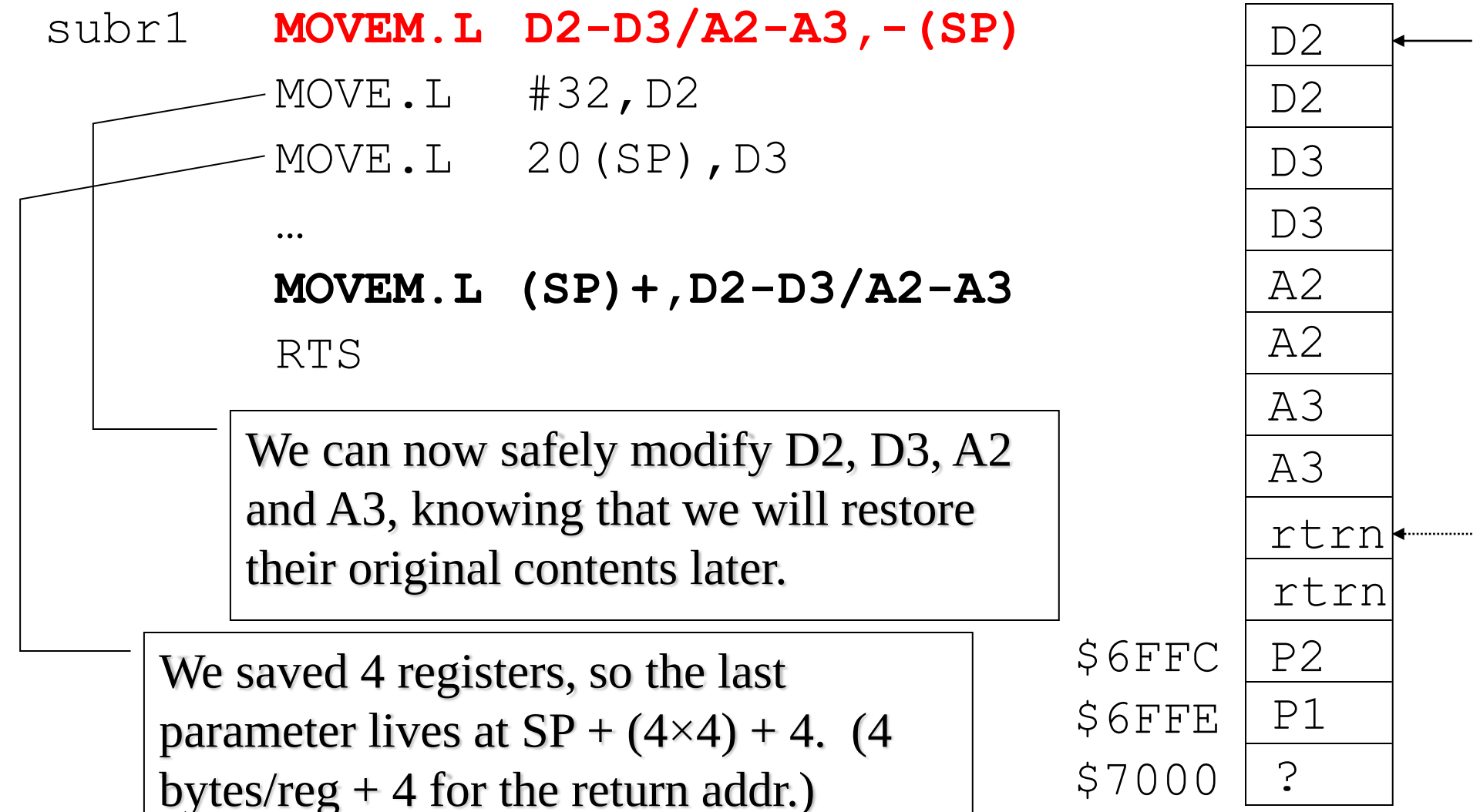
- A ***transparent*** subroutine doesn't change any registers except ***D0***, ***D1***, ***A0*** and ***A1***.
- If we need more registers than this, we must save the register values when we enter the subroutine and restore them later.
- Where do we store them? You guessed it: ***the stack.***
- The 68000 provides a convenient instruction, MOVEM, to push the contents of several registers to the stack at one time.

A Transparent Subroutine

```
subr1   MOVEM.L  D2-D3/A2-A3,-(SP)
        MOVE.L   #32,D2
        MOVE.L   20(SP),D3
        …
        MOVEM.L  (SP)+,D2-D3/A2-A3
        RTS
```

We can now safely modify D2, D3, A2 and A3, knowing that we will restore their original contents later.

We saved 4 registers, so the last parameter lives at SP + (4×4) + 4. (4 bytes/reg + 4 for the return addr.)



Two Mechanisms For Passing Parameters

- **By Value**:
  - Actual value of the parameter is transferred to the subroutine.
  - This is the safest approach unless the parameter needs to be updated.
  - Not suitable for large amounts of data.
  - In order to pass a parameter by value through the stack, one may use the instruction:

```
MOVE   <EA>,-(SP)
```

- **By Reference:**
  - The address of the parameter is transferred.
  - This is necessary if the parameter is to be changed.
  - Recommended in the case of large data volume.
  - In order to pass a parameter by reference through the stack, one may use the instruction:

```
PEA    <EA>
```

Passing By Value & Reference

- We pushed the value of NUM1, NUM2, and NUM3 on the stack.
- *What if we want to change the input parameter values?*
- For example, what if we want to write a subroutine that will multiply all of its arguments' values by 2, *actually changing the values in memory?*
- We must pass the parameters by reference…

The PEA Instruction

- **PEA** – *Push effective address*
- PEA <ea>

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	(xxx).W	(xxx).L	#<data>	d(PC)	d(PC,Rn)
		✔			✔	✔	✔	✔		✔	✔

PEA label

is the same as…

```
LEA       label,A0
MOVEA.L   A0,-(SP)
```

but without using A0.

- *You can "abuse" this instruction to push a constant or any value on the stack.*

Passing Parameters in Registers

```
main        MOVE.W      A,D1
            MOVE.W      B,D2
            MOVE.W      C,D3
            JSR         MUL3
            MOVE.W      D0,D
            STOP        #$2700
MUL3        MOVE.W      D1,D0
            MULS.W      D2,D0
            MULS.W      D3,D0
            RTS
; data area
            ORG         $2000
A           DC.W        5
B           DC.W        6
C           DC.W        7
D           DS.W        1
            END         $1000
```

- **The number to be MULTIPLIED is in D1,D2 & D3.**
- **The result is returned in D0, and eventually passed to D.**

Passing Parameters in Registers

```
; caller
main        MOVE.W      A,D1
            JSR         sqr
            MOVE.W      D0,B
            STOP        #$2700
; callee
sqr         MOVE.W      D1,D0
            MULS.W      D0,D0
            RTS
; data area
            ORG         $2000
A           DC.W        5
B           DS.W        1
            end
```

- **The number to be squared is in D1.**
- **The result is returned in D0.**
- **D1 is unchanged.**

Passing Parameters in Memory

```
main        MOVE.W      A,TEMP1
            MOVE.W      B,TEMP2
            MOVE.W      C,TEMP3
            JSR         MUL33
            MOVE.W      TEMP,D
MUL33       MOVE.W      TEMP1,D0
            MULS.W      TEMP2,D0
            MULS.W      TEMP3,D0
            MOVE.W      D0,TEMP
            RTS
; data area
            ORG         $2000
A           DC.W        5
B           DC.W        6
C           DC.W        7
D           DS.W        1
TEMP        DS.W        1
TEMP1       DS.W        1
TEMP2       DS.W        1
TEMP3       DS.W        1
            end
```

- **The numbers to be squared is stored in TEMP(X)s first.**
- **The result is returned in TEMP.**

Passing Parameter on the Stack

- **If we use registers to pass our parameters:**
  - **Limit of 7 parameters to/from any subroutine (D0 is reserved to hold the single value returned by the subroutine).**
  - **We use up registers so they are not available to our program.**
- **So, instead we push the parameters onto the stack.**
- **Our conventions:**
  - **Parameters are passed on the stack.**
  - **One return value can be provided in D0.**
  - **D0, D1, A0, A1 can be used by a subroutine. Other registers must first be saved.**

First Things First…

- **Both the subroutine and the main program must know how many parameters are being passed!**
  - **In C we would use a prototype:**

    int power (int number, int exponent);

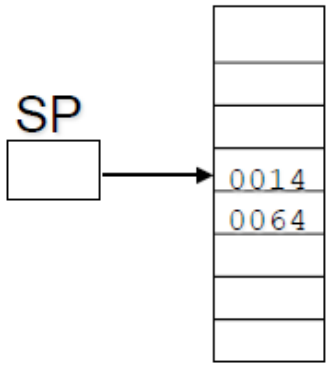
  - **In assembly, you must take care of this yourself.**
- **Things to do:**
  - Push parameters onto the stack
  - Access parameters on the stack using indexed addressing mode
  - Draw the stack to keep track of subroutine execution
    - Parameters
    - Return address
  - Clean the stack after a subroutine call

Steps in using Stacks

CALLER

1. Push parameters on stack

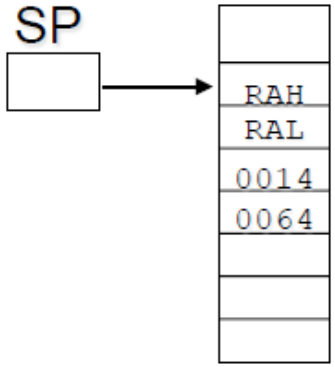
```
MOVE.W   #100,-(SP)
MOVE.W   #20,-(SP)
```



CALLER

2. Call the subroutine

```
JSR      SQR
```



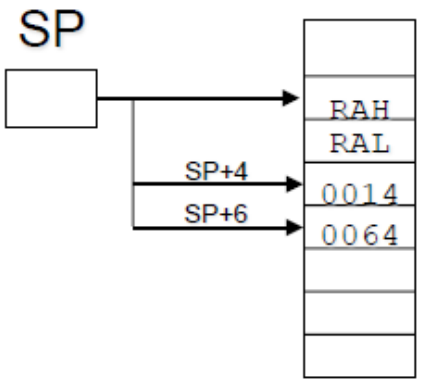
RAH = Return Address High
RAL = Return Address Low

Steps in using Stacks

CALLEE

3. Extract parameters from stack

```
MOVE.W   4(SP),D0
MOVE.W   6(SP),D1
```



CALLEE

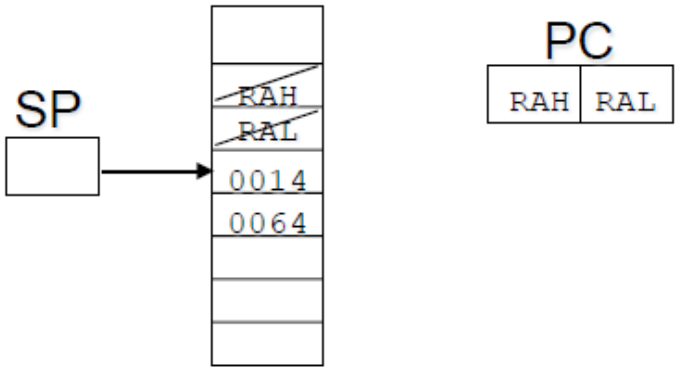
4. Use the parameters & store calculation result in D0.

```
MULU    D1,D0
```

CALLEE

5. Return to caller

```
RTS
```



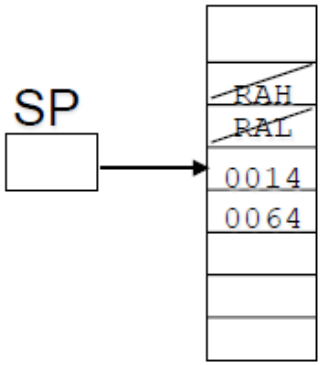
RAH = Return Address High
RAL = Return Address Low

Steps in using Stacks

CALLER

6. Use returned data

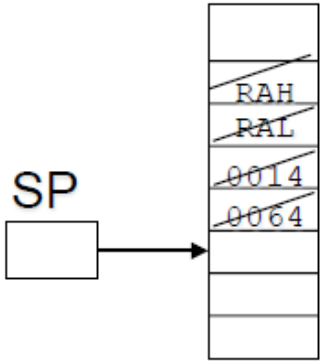
```
MOVE.W   D0,SAVE
```



CALLER

7. Clean up the stack

```
ADDA   #4,SP
```



Passing Parameters On The Stack

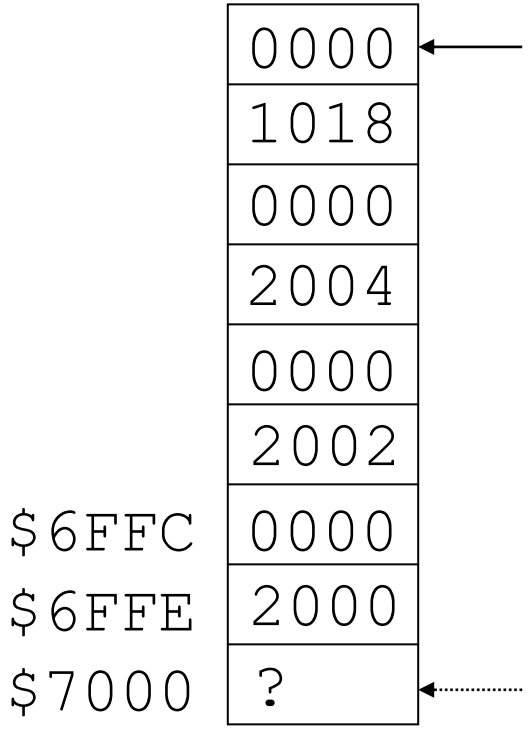
Mul3 – multiply three numbers and place the result in D0.

```
; ******** Main Program ********************************
1000    START    MOVE.W    NUM1,-(SP)    ;Push first param
1006             MOVE.W    NUM2,-(SP)    ;Push 2nd param
100C             MOVE.W    NUM3,-(SP)    ;Push 3rd param
1012             JSR       MUL3
1018             ADDA.L    #6,SP         ;Clean the stack!
101E             STOP      #$2700
1020             NOP
; ******* Subroutine Mul3 ******************************
1022    MUL3     MOVE.W    4(SP),D0      ;D0 = NUM3
1026             MULS.W    6(SP),D0      ;D0 *= NUM2
102A             MULS.W    8(SP),D0      ;D0 *= NUM1
102E             RTS                     ;SP --> rtrn addr!
                 ORG       $2000
2000    NUM1     DC.W      5
2002    NUM2     DC.W      6
2004    NUM3     DC.W      7
2005    NUM4     DS.L      1
                 END
```

Passing Parameters By Reference

dbl3 – double the values of three parameters

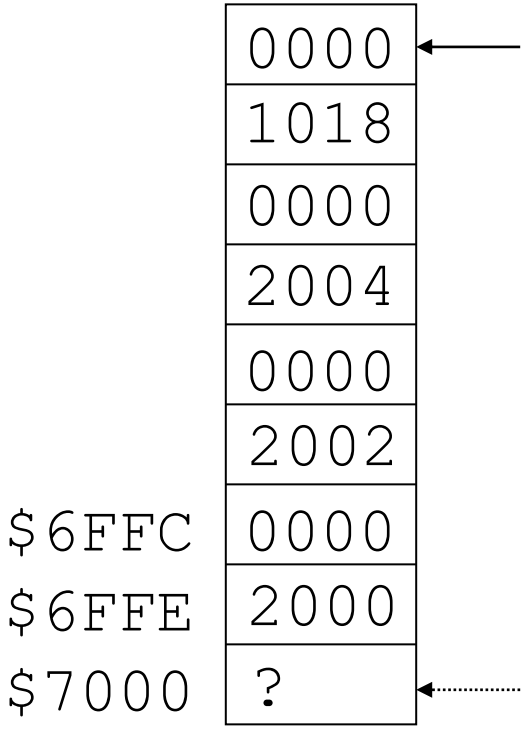
```
; ******** Main Program *********
1000   START   PEA       NUM1
1006           PEA       NUM2
100C           PEA       NUM3
1012           JSR       DBL3
1018           ADDA.L    #12,SP
101E           STOP      #$2700
1020           NOP
               ORG       $2000
2000   NUM1    DC.W      5
2002   NUM2    DC.W      8
2004   NUM3    DC.W      2
               END
```



Using Parameters Passed By Reference

dbl3 – double the values of three parameters

```
; ******* Subroutine dbl3 *********
      DBL3    MOVEA.L  4(SP),A0
              MOVE.W   (A0),D0
              MULS.W   #2,D0
              MOVE.W   D0,(A0)
              MOVEA.L  8(SP),A0
              … ; repeat for each
              RTS
              ORG      $2000
2000  NUM1    DC.W     5
2002  NUM2    DC.W     8
2004  NUM3    DC.W     2
              END
```



Characteristics Of Good Subroutines

- **Generality** – can be called with *any* arguments
  - Passing arguments on the stack does this.
- **Transparency** – you have to leave the registers like you found them, except for D0, D1, A0, and A1.
  - We use the MOVEM instruction for this purpose.
- **Readability** – well documented.
  - See the example handout.
- **Re-entrant** – subroutine can call itself if necessary
  - This is done using *stack frames*, something we will look at in a few days…

Example: Power Calculation Subroutine

- **A subroutine is needed which accepts two integers as input parameters:**
  - **a base, B (a signed integer), Size = one byte (range: $-128 \leq B \leq 127$)**
  - **an exponent E (a positive integer) Size = one byte,**
  - **and, compute the function B^E size of answer = long word**

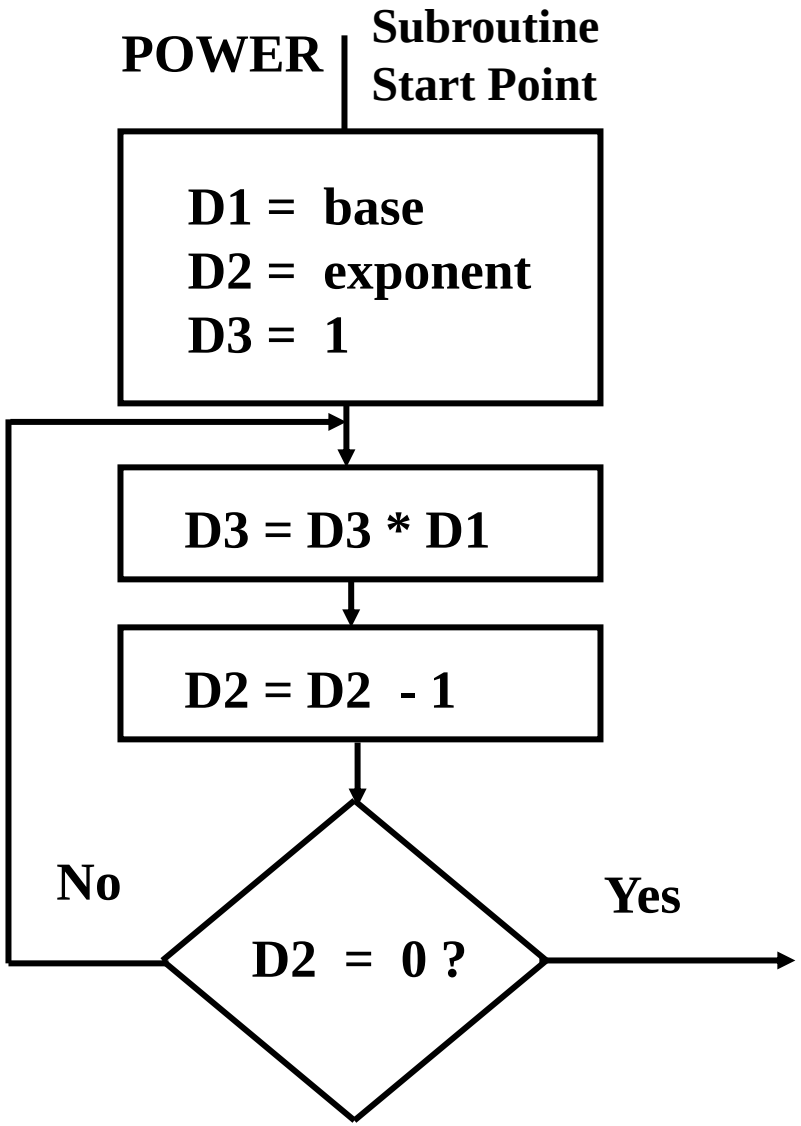
Functional specification (pseudo code) of subroutine POWER:

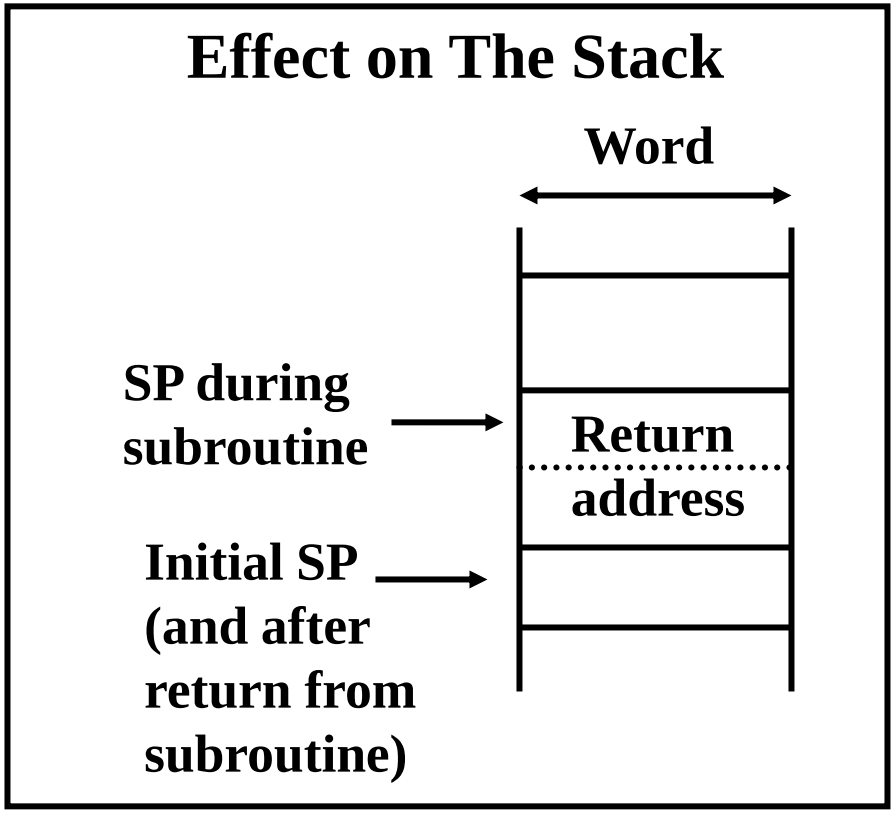
```
POWER (B, E)

D1  = B                          ;input arguments, base
D2  = E                          ;exponent, a positive integer

initialize D3 to 1               ;answer initialized to 1
while  D2  > 0
     D3 =  D1*D3                 ;compute function  using
     D2 = D2  - 1;               ;continued product of base
end POWER
Return to calling program
```

Basic Flow Chart of Power





POWER: Four Parameter Passing Cases

- **We'll examine four assembly versions of the subroutine POWER and sample Main programs that calls it.**
- **Each version uses a different parameter passing method:**

  - **Case 1: Parameter passing *by value*, using data registers.**
  - **Case 2: Parameter passing *by reference*, using address registers.**
  - **Case 3: Parameter passing *by value*, using the stack.**
  - **Case 4: Parameter Passing *by reference*, using the stack**

POWER Subroutine Example (Case 1)

Parameter Passing *by Value*: Using Data Registers

- Main Program -

```
MAIN    ORG       $400          Main Program origin
        MOVEA.L   #$07FFE,SP    Initialize  Stack Pointer
        MOVE.B    B,D1          Put base number into D1
        EXT.W     D1            Sign extend base to word length
        CLR.W     D2            Clear D2 before loading exponent
        MOVE.B    E,D2          Put exponent number into D2
        BSR       POWER         Call subroutine  POWER
        LEA       A,A5           put address of  answer into A5
        MOVE.L    D3,(A5)       save  answer
        STOP      #$2700        Done
        ORG       $600
B       DC.B      4             Base number stored here
E       DC.B      2             Exponent number stored here
A       DS.L      1             answer to be stored  here
```

POWER Subroutine Example (Case 1)

Parameter Passing *by Value*: Using Data Registers Continued - Subroutine

```
        ORG     $800        Subroutine  POWER origin
POWER   MOVE.L  #1,D3       initialize result to 1
LOOP    MULS    D1,D3       multiply result with base
        SUB     #1,D2       decrement power by one
        BNE     LOOP        and repeat as long as power > 0
        RTS                 Done, return to calling program
```

POWER Subroutine Example (Case 2)

Parameter Passing *by Reference*: Using Address Registers
- Main Program -

```
MAIN    ORG         $400            Main Program origin
        MOVEA.L     #$07FFE,SP      Initialize  Stack Pointer
        LEA         B,A1            A1  points to base number
        LEA         E,A2            A2  points to exponent
        BSR         POWER           Call subroutine  POWER
        LEA         A,A5             put address of  answer into A5
        MOVE.L      D3,(A5)         save  answer in memory
        STOP        #$2700          Done
        ORG         $600
B       DC.B        4               Base number stored here
E       DC.B        2               Exponent number stored here
A       DS.L        1               answer to be stored  here
```

POWER Subroutine Example (Case 2)

Parameter Passing *by Reference*: Using Address Registers Continued - Subroutine

```
        ORG     $800        Subroutine  POWER origin
POWER   MOVE.B  (A1),D1     copy base number to D1
        EXT.W   D1          Sign extend base to word length
        CLR.W   D2          Clear D2 before loading exponent
        MOVE.B  (A2),D2     copy exponent to D2
        MOVE.L  #1,D3       initialize result in D3 to 1
LOOP    MULS    D1,D3       multiply result D3 with base D1
        SUB     #1,D2       decrement power in D2 by one
        BNE     LOOP        and repeat as long as power > 0
        RTS                 Done, return to calling program
```

68000 Addressing Modes Revisited:
Address Register Indirect Addressing with Displacement

- **The addressing notation:**

  d16(A0) or (d16,A0)

  - **Refers to the address contained in register A0 plus a signed 16 bit displacement d16**
  - **Some assembles accept only one of the above syntax forms.**

- **Examples:**

MOVE.L (12,A4),D3 or MOVE.L 12(A4),D3

MOVE.W (-$4,A1),D0 or MOVE.W -$4(A1),D0

POWER Subroutine Example (Case 3)

Parameter Passing *by Value*: Using The Stack - Main Program -

```
MAIN    ORG       $400              Main Program origin
        MOVEA.L   #$07FFE,SP        Initialize  Stack Pointer
        MOVE.B    B,D1              Put base number into D1
        EXT.W     D1                Sign extend base to word length
        MOVE.W    D1,-(SP)          push base  B onto the stack
        CLR.W     D2                Clear D2 before loading exponent
        MOVE.B    E,D2              Put exponent number into D2
        MOVE.W    D2,-(SP)          push exponent E onto the stack
        BSR       POWER             Call subroutine  POWER
        MOVE.L    (SP)+,D3          pop answer from stack resetting SP
        LEA       A,A5               put address of  answer into A5
        MOVE.L    D3,(A5)           save  answer
        STOP      #$2700            Done
        ORG       $600
B       DC.B      4                 Base number stored here
E       DC.B      2                 Exponent number stored here
A       DS.L      1                 answer to be stored  here
```

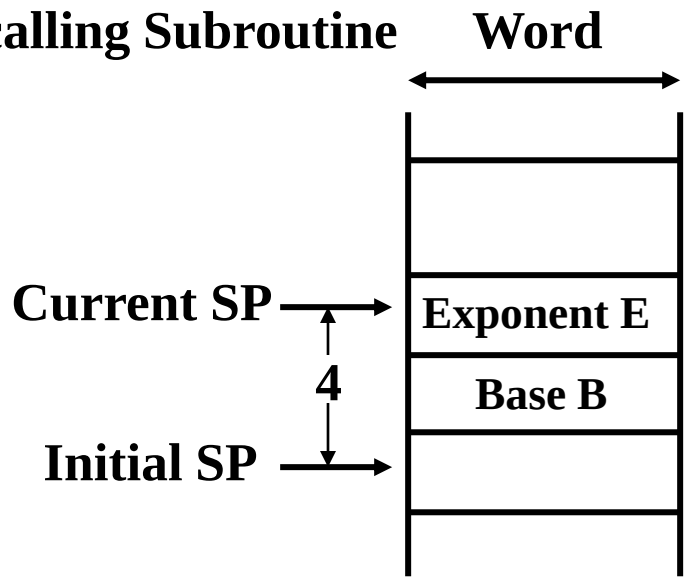
POWER Subroutine Example (Case 3)

Parameter Passing *by Value*: Using The Stack
Continued - Subroutine -

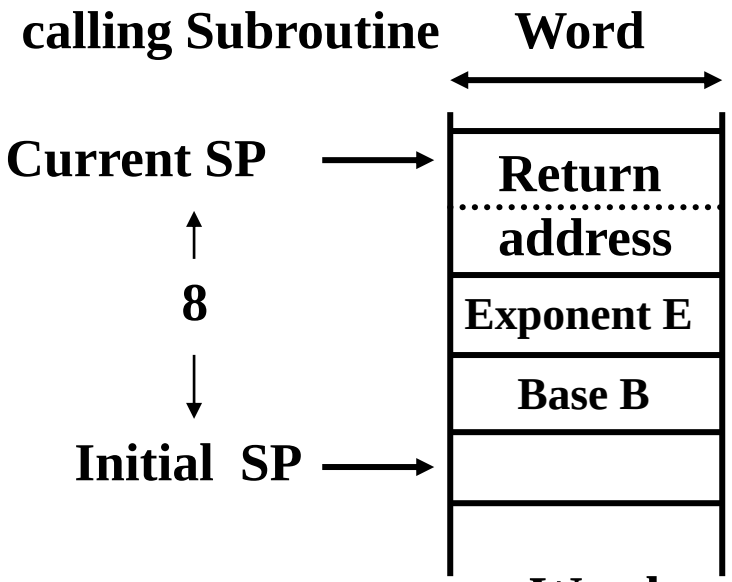
```
        ORG     $800        Subroutine  POWER origin
POWER   MOVE.W  6(SP),D1    copy base  from stack to D1
        CLR.W   D2          Clear D2 before loading exponent
        MOVE.B  4(SP),D2    copy exponent from to D2
        MOVE.L  #1,D3       initialize result in D3 to 1
LOOP    MULS    D1,D3       multiply result D3 with base D1
        SUB     #1,D2       decrement power in D2 by one
        BNE     LOOP        and repeat as long as power > 0
        MOVE.L  D3,4(SP)    Push result onto the stack
        RTS                 Done, return to calling program
```

Effect on The Stack

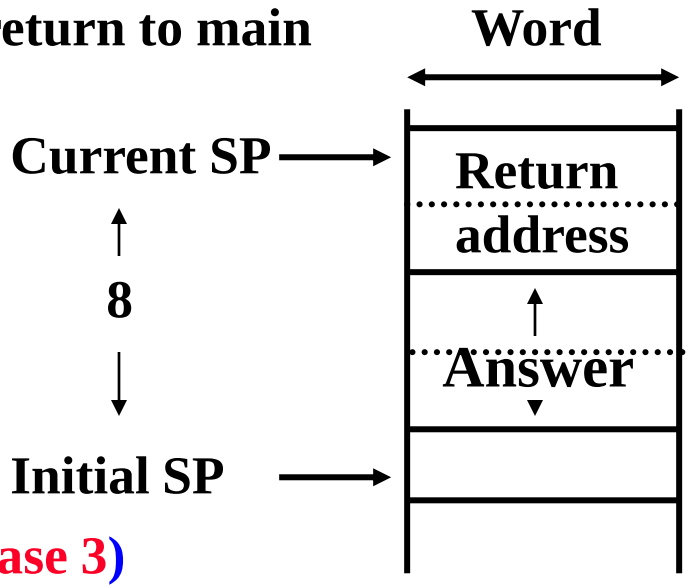
Just before
calling Subroutine



Just after
calling Subroutine

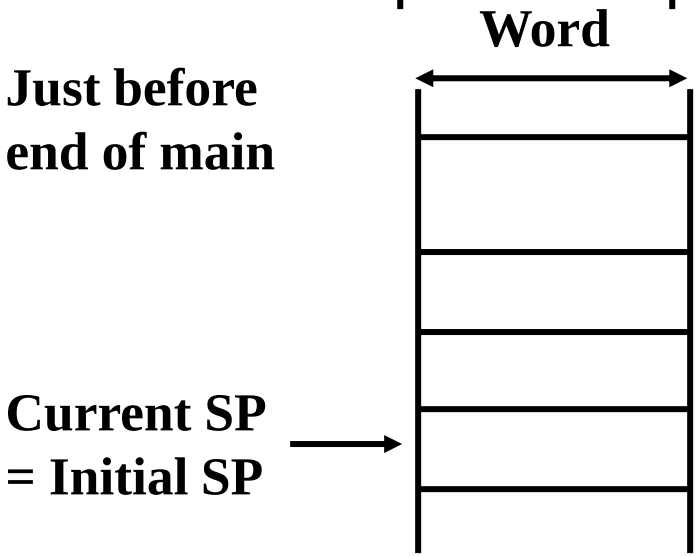


Just before
return to main



(Case 3)

Just before
end of main



POWER Subroutine Example (Case 4)

Parameter Passing *by Reference*: Using The Stack

- Main Program -

```
MAIN    ORG       $400          Main Program origin
        MOVEA.L   #$07FFE,SP    Initialize  Stack Pointer
        PEA       B             Push address of Base  onto the stack
        PEA       E             Push address of Exponent  onto the stack
        PEA       A             Push address of Answer  onto the stack
        BSR       POWER         Call subroutine  POWER
        LEA       12(SP),SP     Stack clean-up:  stack pointer reset
        STOP      #$2700        Done
        ORG       $600
B       DC.B      4             Base number stored here
E       DC.B      2             Exponent number stored here
A       DS.L      1             answer to be stored  here
```

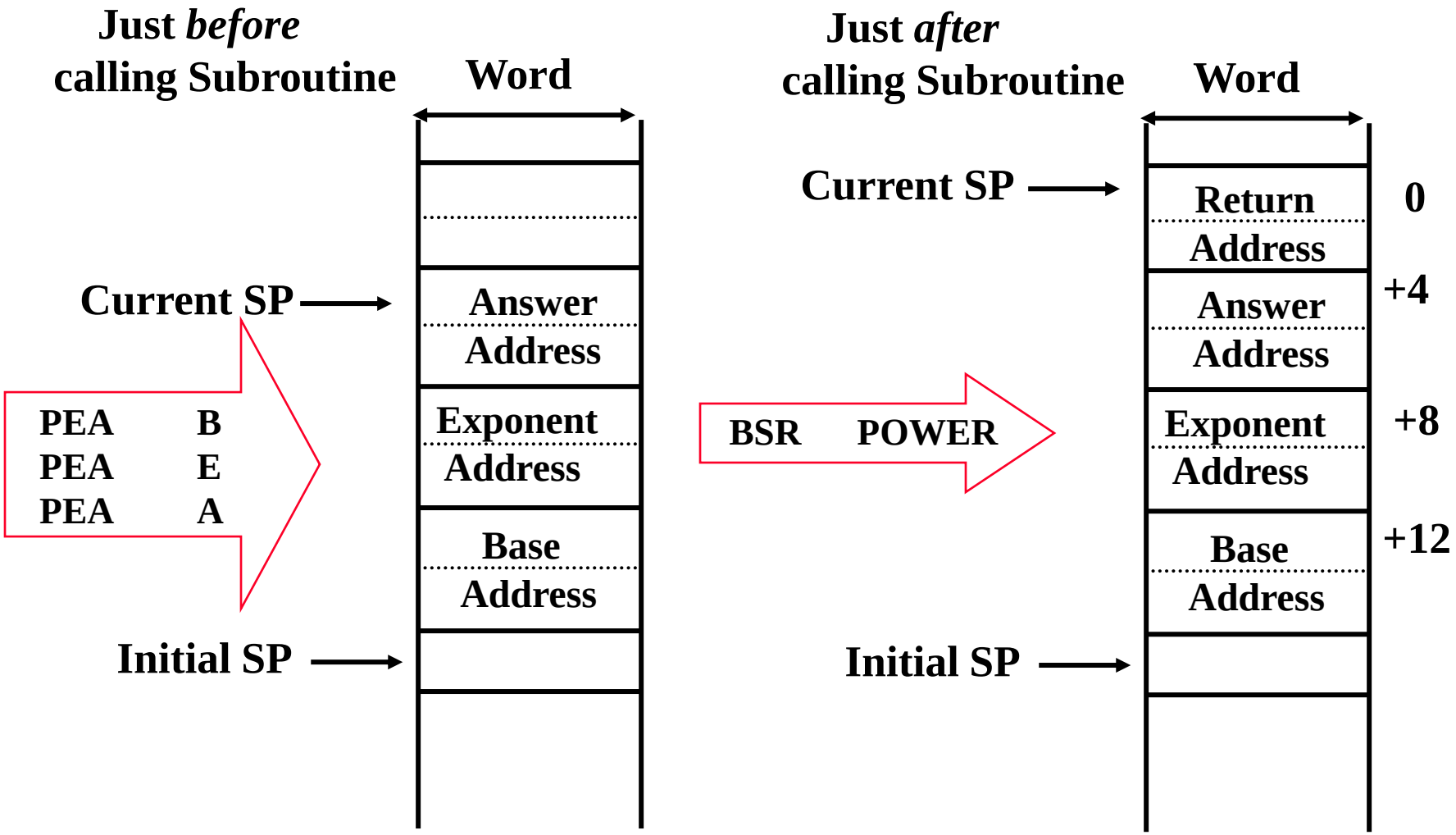
POWER Subroutine Example (Case 4)

Parameter Passing *by Reference*: Using The Stack

Continued - Subroutine -

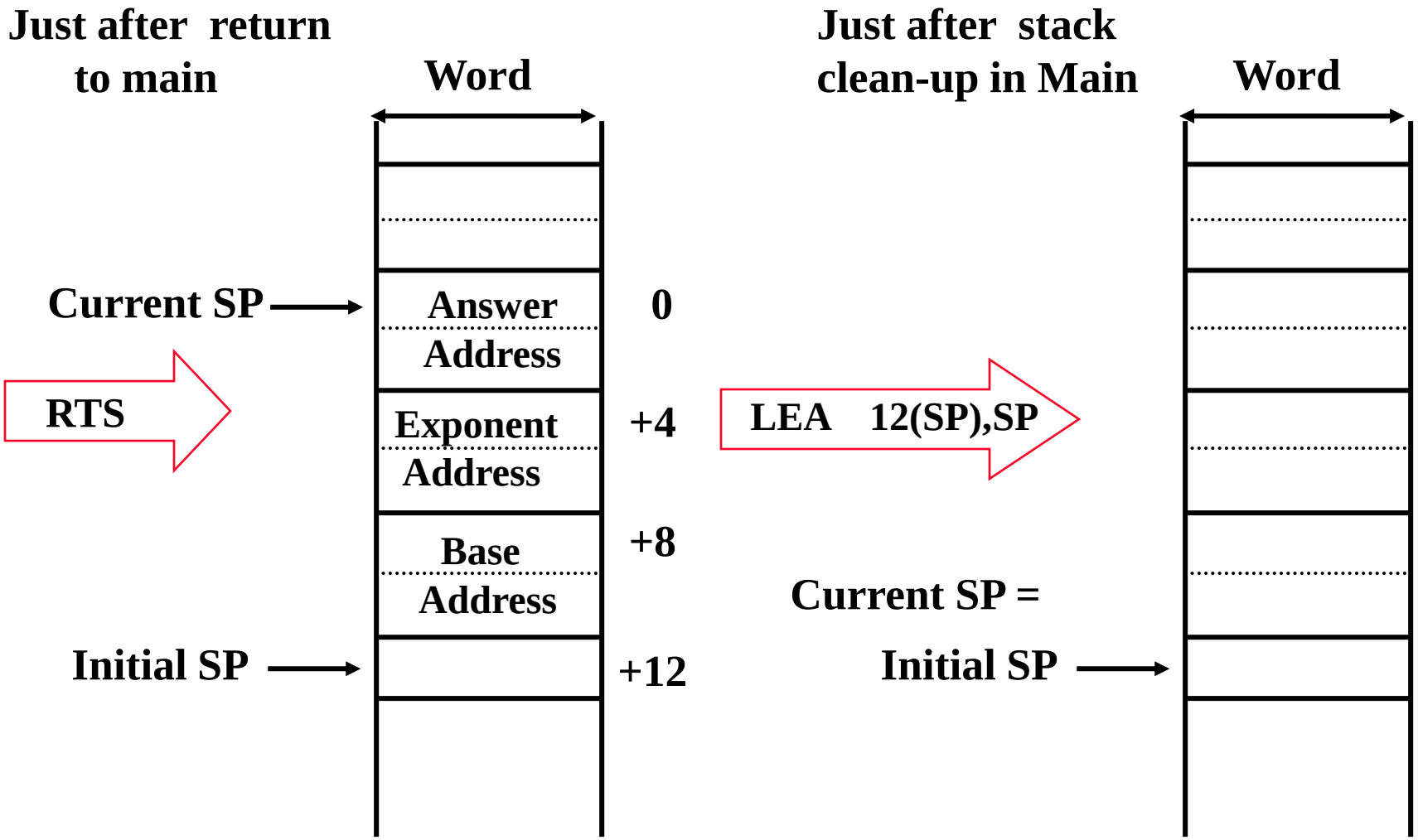
```
        ORG     $800        Subroutine  POWER origin
POWER   MOVEA.L 12(SP),A1   load Base address  in A1
        MOVEA.L 8(SP),A2    load Exponent address  in A2
        MOVEA.L 4(SP),A3    load Answer address address in A3
        MOVE.B  (A1),D1     Put base number into D1
        EXT.W   D1          Sign extend base to word length
        CLR.W   D2          Clear D2 before loading exponent
        MOVE.B  (A2),D2     copy exponent from to D2
        MOVE.L  #1,D3       initialize result in D3 to 1
LOOP    MULS    D1,D3       multiply result D3 with base D1
        SUB     #1,D2       decrement power in D2 by one
        BNE     LOOP        and repeat as long as power  > 0
        MOVE.L  D3,(A3)     Save result in memory
        RTS                 Done, return to calling program
```

Effect on The Stack



(Case 4)

Effect on The Stack



(Case 4)

Questions?

- **How does a subroutine returns immediately after the call site?**
- **Where and how does it return a value?**
- **Where and how are we passing arguments to a subroutine?**
- **What happens to register values once a subroutine is called. Do we require that the subroutine preserves their values or is it OK to overwrite some registers?**
- **Where and how are we allocating storage for any local variables (i.e., variable that belong to the subroutine)?**

ASCII-Encoded Decimal To Binary Conversion

- **A useful subroutine**

```
* Subroutine DECBIN
* A0 points to the highest character of a valid five character
* ASCII-encoded decimal number with a maximum value 65535
* The decimal number is converted to a one word binary value
* stored in the low word of D0

DECBIN    CLR.L     D0          Clear result register
          MOVEQ     #5,D6       Initialize loop counter to get 5 digits
NEXTD     CLR.L     D1          Clear new digit holding register
          MOVE.B    (A0)+,D1    Get one ASCII digit from memory
          SUB.B     #$30,D1     Subtract ASCII bias $30
          MULU      #10,D0      Multiply D0 by 10
          ADD.W     D1,D0       Add new digit to binary value in D0
          SUB.B     #1,D6       Decrement counter
          BNE       NEXTD       If not done get next digit
          RTS
```

ASCII-Encoded Decimal To Binary Conversion

- **A better version**

```
* Subroutine DECBIN
* A0 points to the highest character of a valid five character
* ASCII-encoded decimal number with a maximum value 65535
* The decimal number is converted to a one word binary value
* stored in the low word of D0

DECBIN    MOVEM.L   D1/D6,-(SP) Save the registers we're borrowing
          MOVEQ     #0,D0       MOVEQ faster than CLR.L
          MOVEQ     #5,D6       Initialize loop counter to get 5
digits
NEXTD     MOVEQ     #0,D1       Clear new digit holding register
          MOVE.B    (A0)+,D1    Get one ASCII digit from memory
          SUB.B     #$30,D1     Subtract ASCII bias $30
          MULU      #10,D0      Multiply D0 by 10
          ADD.W     D1,D0       Add new digit to binary value in D0
          SUB.B     #1,D6       Decrement counter
          BNE       NEXTD       If not done get next digit
          MOVEM.L   (SP)+,D1/D6 Restore registers
          RTS
```

Review Problem #1

```
; ******** Main Program *********
1000   START   MOVE.L   NUM1,-(SP)
1006           MOVE.L   NUM2,-(SP)
100C           MOVE.L   NUM3,-(SP)
1012           JSR      SUB1
1018           Next instr…
…              …
; ******* Subroutine SUB1 *******
1022   SUB1    …
1026           …
102E           RTS
               ORG      $2000
2000   NUM1    DC.L     $150
2004   NUM2    DC.L     $180
2008   NUM3    DC.L     $12
               END
```

Review Problem #2

```
; ******** Main Program *********
1000   START   MOVE.L   NUM1,-(SP)
1006           MOVE.L   NUM2,-(SP)
100C           MOVE.L   NUM3,-(SP)
1012           JSR      SUB1
1018           Next instr…
…              …
; ******* Subroutine SUB1 *******
1022   SUB1    MOVEM.L  D2-D3/A4,-(SP)
1026           … ; show stack
102E           MOVEM.L  (SP)+,D2-D3/A4
1032           RTS
               ORG      $2000
2000   NUM1    DC.L     $150
2004   NUM2    DC.L     $180
2008   NUM3    DC.L     $12
               END
```