EE 357 Unit 8

Stack Frames

Stack Frames

- Compilers use the stack:
  - to pass parameters to a subroutine
  - to store the return address
  - for storage of local variables declared in the subroutine and a place to save register values
- Every call to a subroutine will create a data structure called a “frame” on the stack to store this information
- To access this data structure on the stack an address register pointer called the “frame pointer” (FP) is used in addition to the normal stack pointer (SP)

Stack Frames

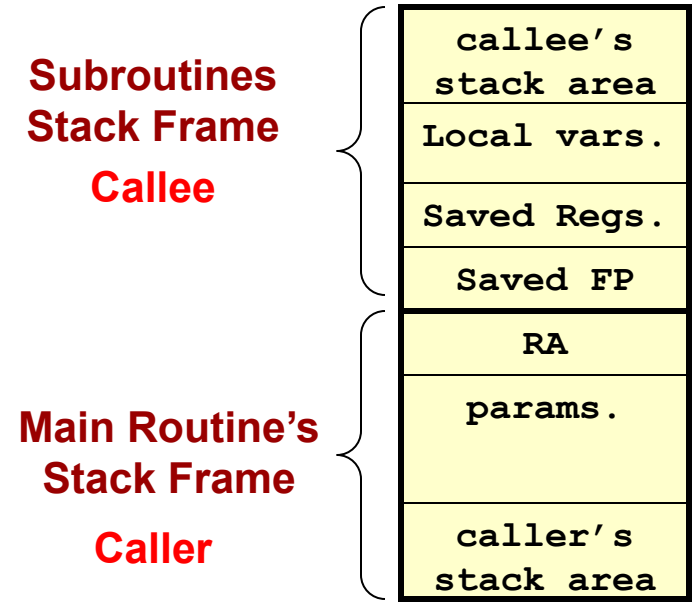
- Caller's code pushes parameters (arguments)
- Call to subroutine pushes RA
- Callee saves registers it will overwrite (including caller's FP)
- Callee allocates space for local variables

Caller

```
void main() {
  ...
  ans = avg(param1, param2);
}
```

Callee

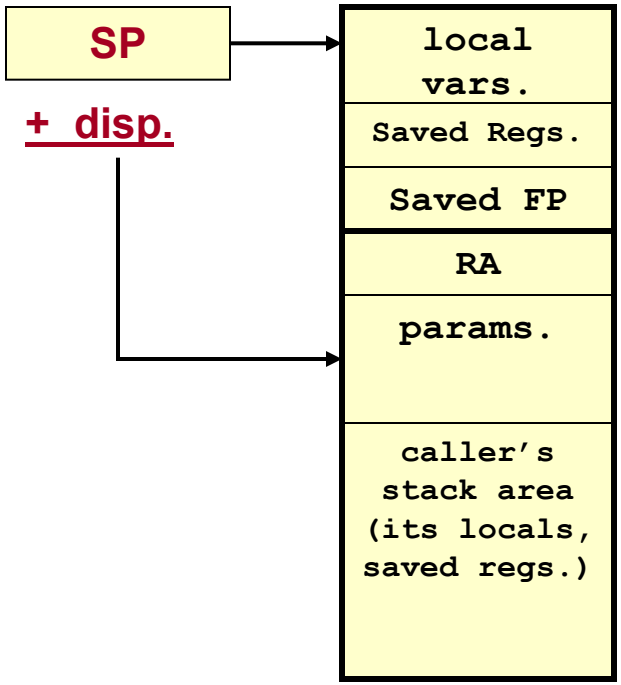
```
int avg(int a, int b) {
  int temp=1;   // local var's
  ...
}
```



Stack Frame Organization

Accessing Values on the Stack

- Stack pointer (SP) is usually used to access only the top value on the stack
- To access parameters / arguments, we need to access values buried on the stack
- We could try to use an offset from SP, but other push operations by the callee may change the SP requiring variable displacements
- Need a better solution

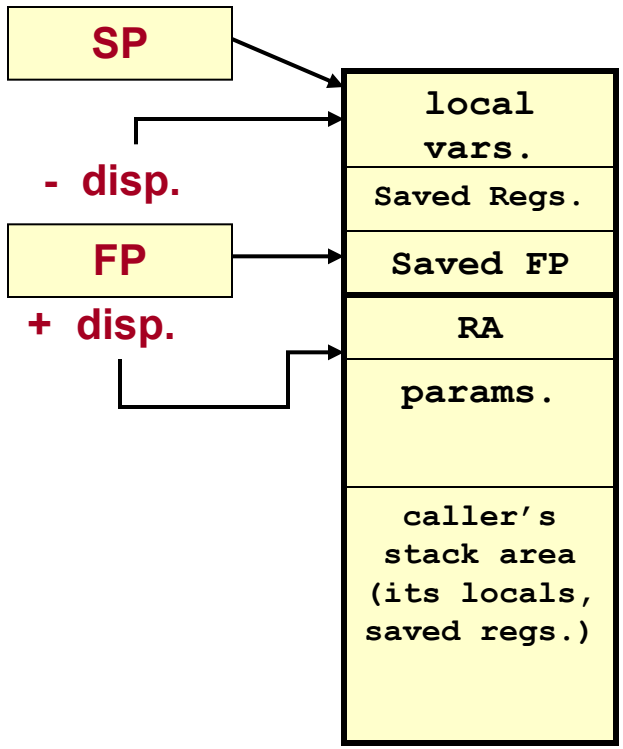


To access parameters we could try to use some displacement

[i.e. d(SP)]

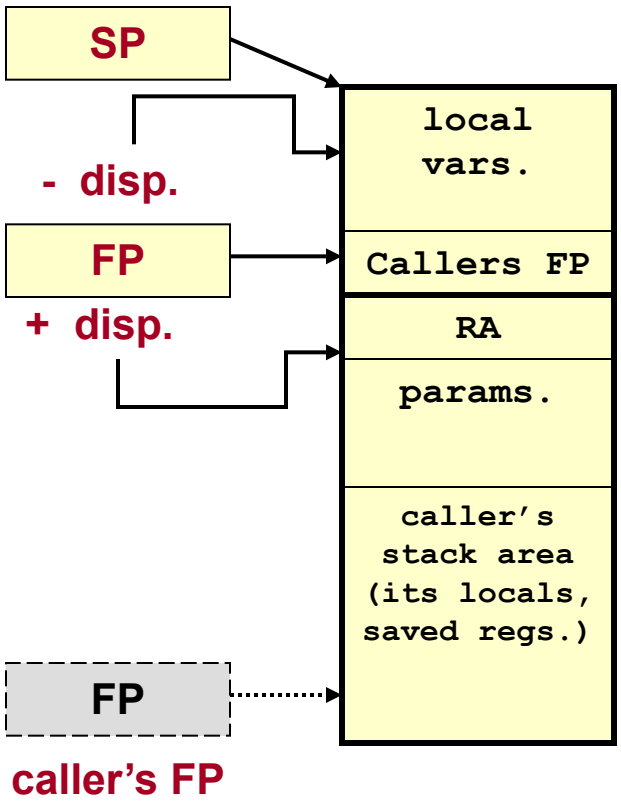
Frame Pointer

- Use a new pointer called Frame Pointer (FP) to point to the base of the current routines frame
  - A6 is used as FP
- FP will not change during the course of subroutine execution
- Can use constant displacements from FP to access parameters or local variables
  - Key 1: FP doesn't change during subroutine execution
  - Key 2: Number of parameters and local variables is a known value
    - Locals accessed @ negative disp. from FP
    - Arguments accessed @ positive disp. from FP



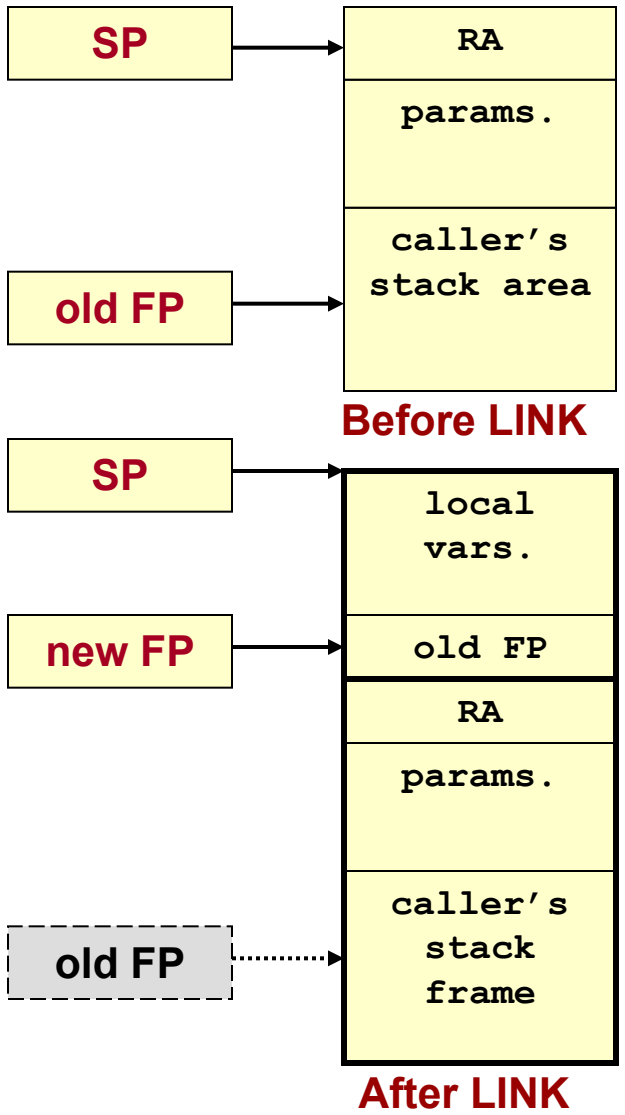
Frame Pointer and Subroutines

- Problem is that each executing subroutine needs its own value of FP
- The called subroutine must save the caller's FP and setup its own FP as the first thing it does when it is called.
- The called subroutine must restore the caller's FP before it returns
- LINK and UNLK instructions are used to save/restore old FP and allocate space for locals

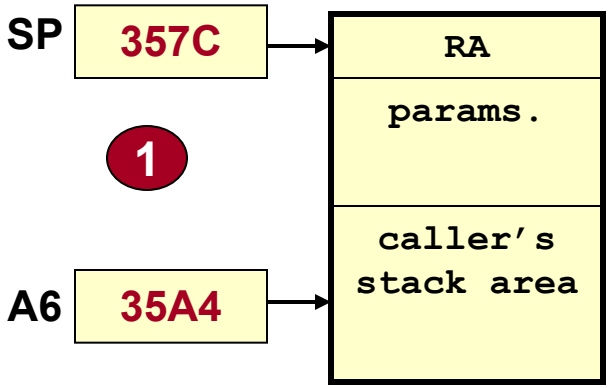


LINK Instruction

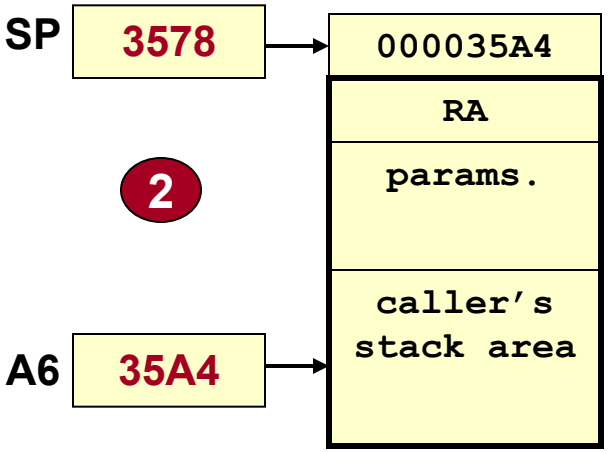
- LINK should be first instruction of a subroutine
  - Pushes FP on top of stack and can also allocates space for that routines stack frame (local var's, saved reg's, etc.)
- LINK An, #imm
  - An should be the FP, imm = # of bytes for locals and other saved values
  - Saves current An (FP) onto the stack
    - SP = SP – 4; M[SP] = An
  - Sets FP to current SP (top of stack)
    - SP = An
  - Adds imm. to SP to allocate space on the stack for local variables
    - SP = SP + imm.
    - imm. should be negative to allocate space



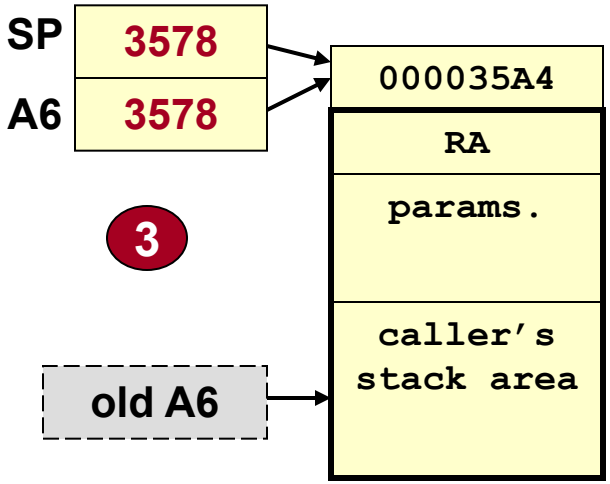
LINK Instruction Example



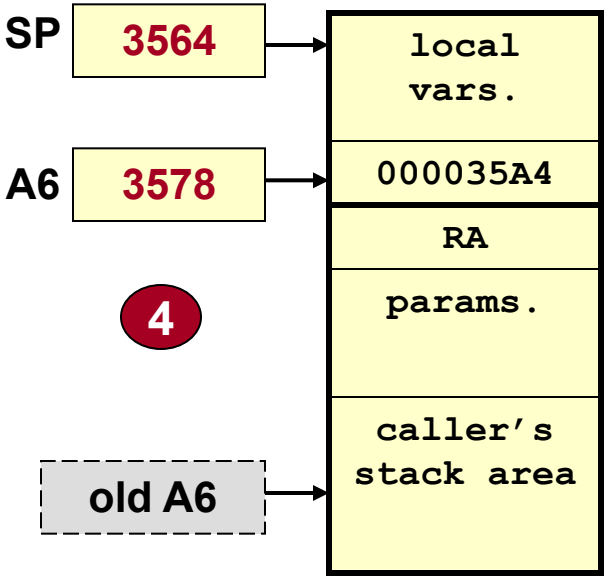
1.) Before LINK



2.) old FP is pushed onto stack



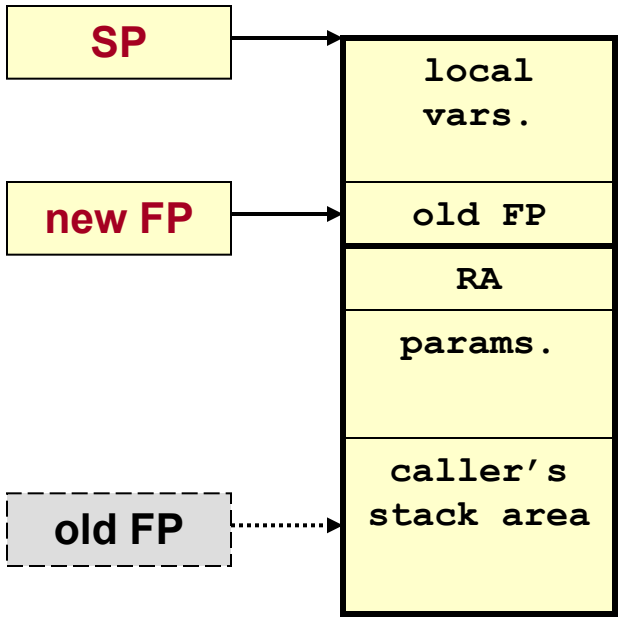
3.) SP is copied into FP so now FP points at old FP



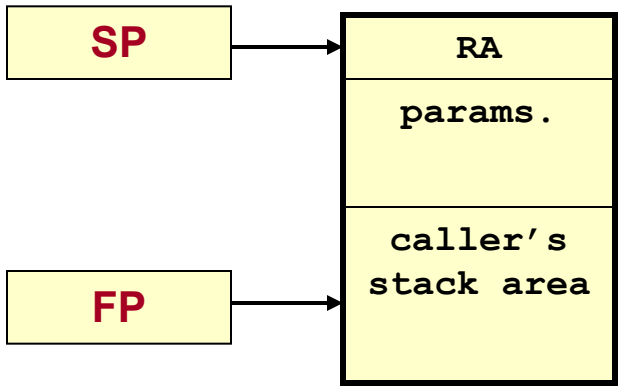
4.) negative immediate value is added to SP to allocate local var. space

UNLK Instruction

- UNLK should be used just before RTS
  - Restores old FP by popping it off the stack as well as deallocating local variable space on the stack.
- UNLK An
  - Copies An (i.e. FP) to SP so that SP now points at old FP.
    - SP = An
  - Pops old FP into An
    - An = M[SP]
    - SP = SP + 4

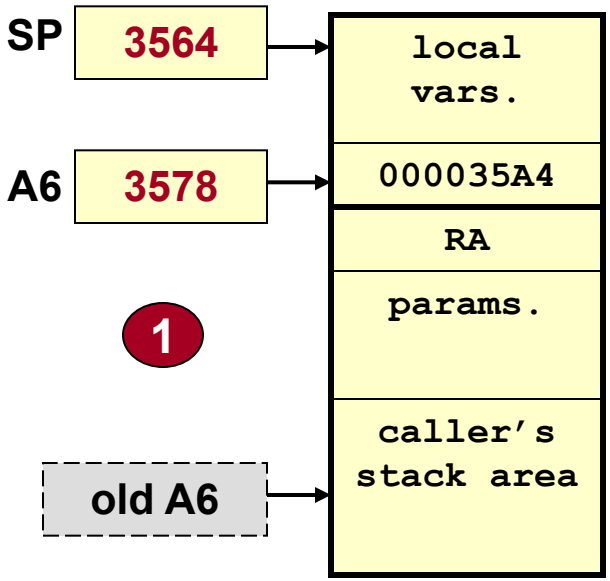


Before UNLK



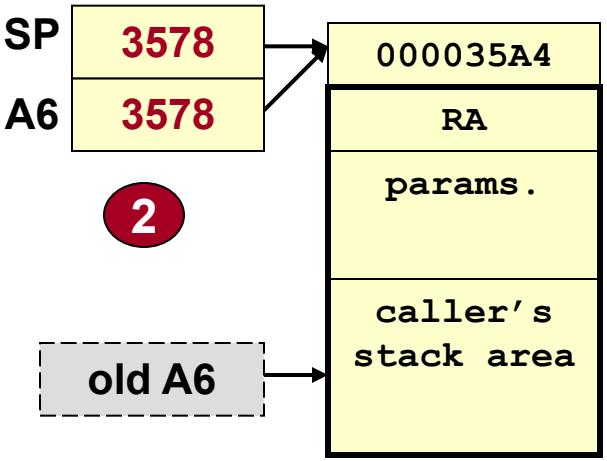
After UNLK

UNLK Instruction Example

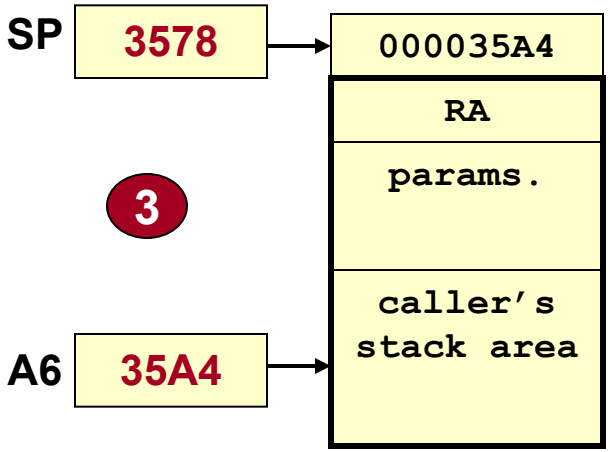


1.) Before UNLK

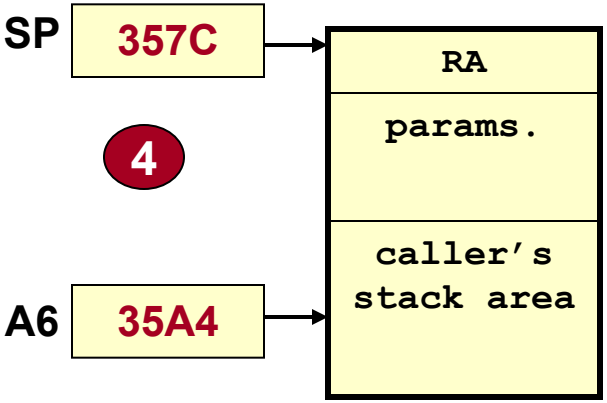
2.) FP is copied to SP so that SP now points below the locals



3.) old FP on the stack is copied into FP restoring the callers FP value



4.) SP is incremented to now point at RA



LINK & UNLK

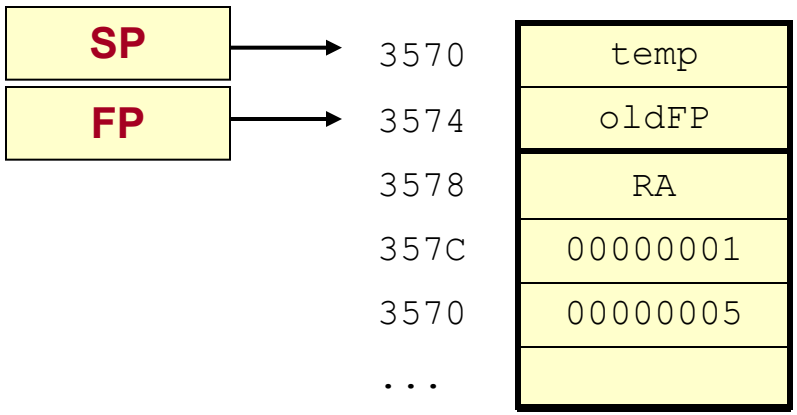
- LINK and UNLK are the single instruction equivalent to the following sequences:
  - LINK An,#imm => MOVE.L An,-(SP)
    MOVEA.L SP,An
    ADDA.L #imm,SP
  - UNLK => MOVEA.L An,SP
    MOVEA.L (SP)+,An

Building a Stack Frame

- Caller's code pushes parameters (arguments)
  - Parameters pushed in reverse order (param. n first)
- Call to subroutine pushes RA
- First instruction of subroutine is LINK
  - Saves old FP on the stack and allocates space for local variables
- Last instruction of subroutine is UNLK
  - Deallocates local variables and restores old FP
- RTS pops RA
- Caller's code pops parameters

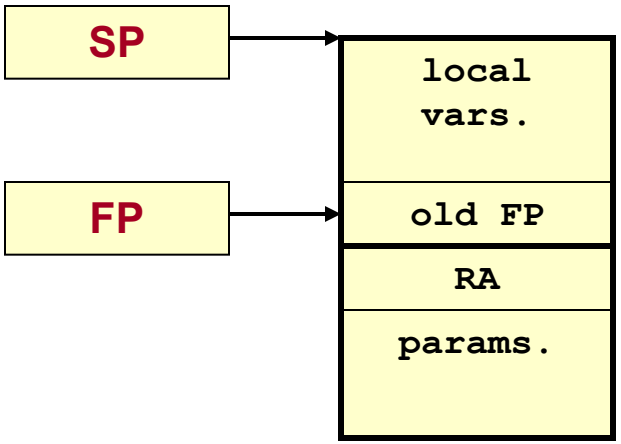
```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



SP and FP Summary

- A6 is used as FP
- FP always points at old FP
  - Local variables are at lower addresses (above) where FP points [i.e. -4(FP), -8(FP), …]
  - RA is directly below old FP [i.e. 4(FP)]
  - Parameters are below RA [i.e. 8(FP), 12(FP), 16(FP)…]
- SP is always pointing at current top of stack (above FP)



Stack Frame Example

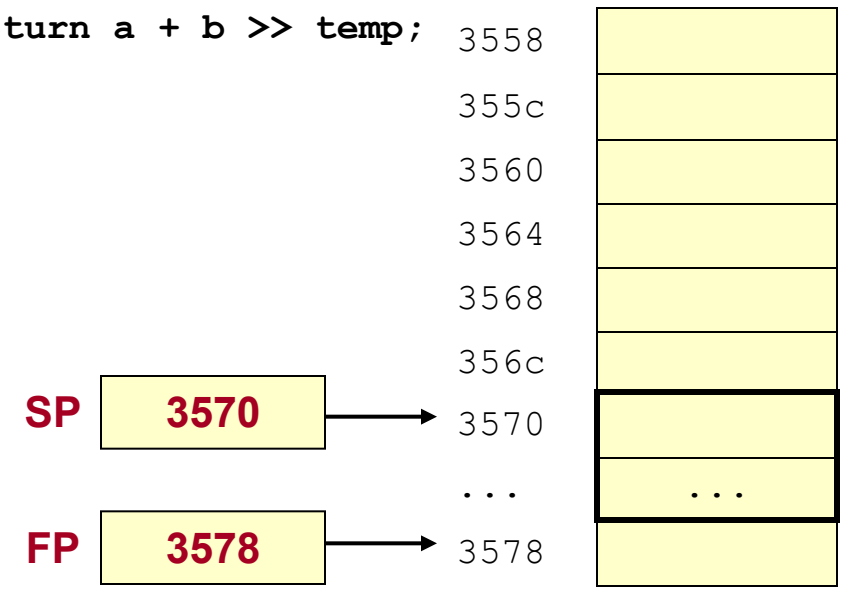
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



Assume for MAIN that FP points to 0x3578 and SP points to 0x3570

Stack Frame Example

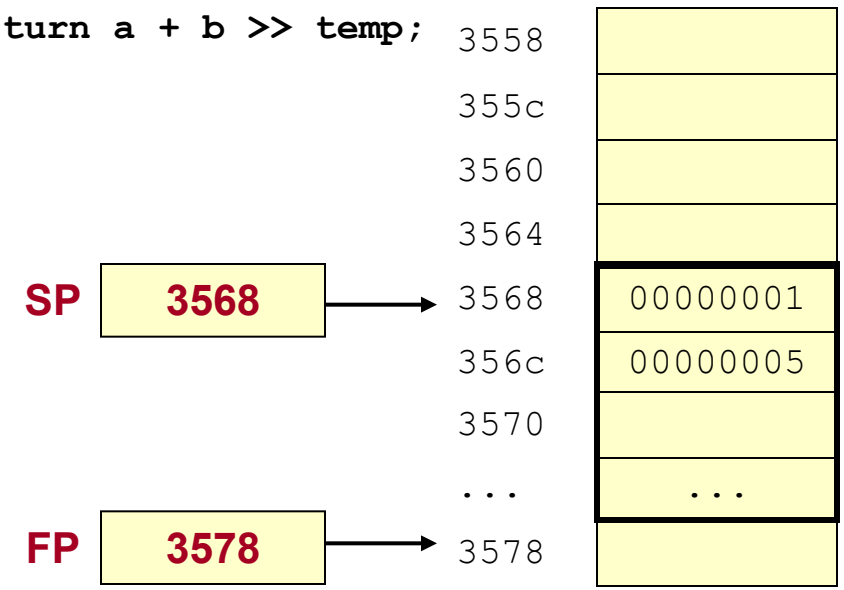
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



Push the parameters on in reverse order

Stack Frame Example

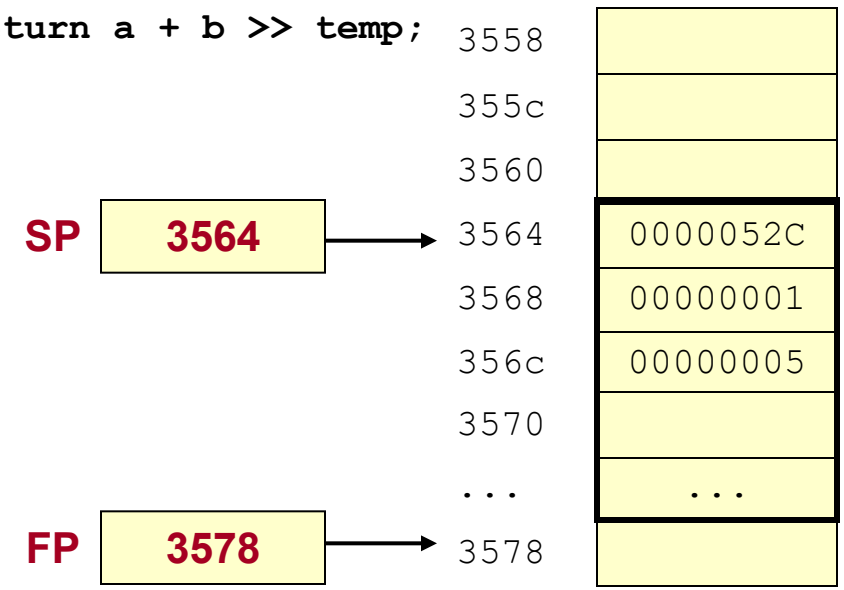
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



BSR pushes return address and branches to AVG

Stack Frame Example

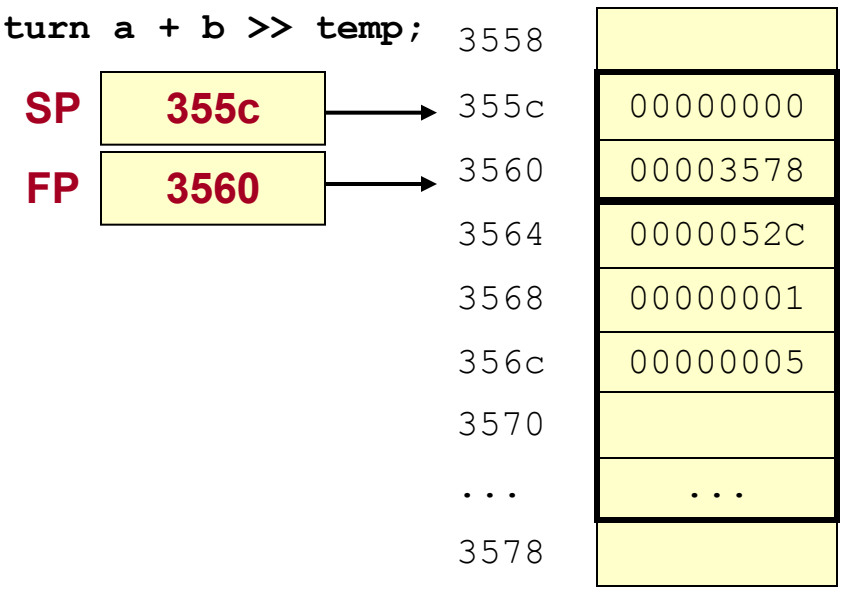
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



LINK causes the old A6 value of 0x3578 to be pushed on the stack, then the SP value which would be 0x3560 is copied into A6 overwriting its old FP value. Finally, -4 is added to the SP to allocate space for the temp variable.

Stack Frame Example

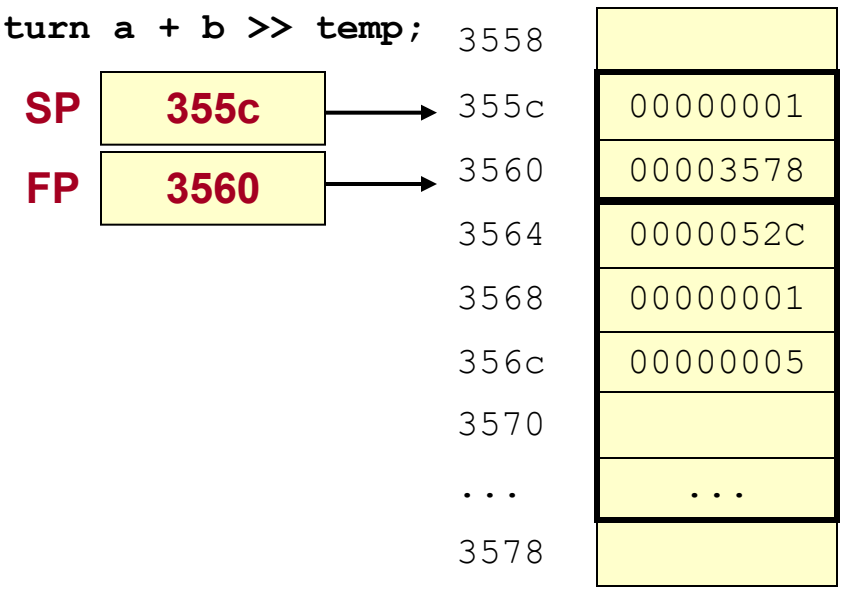
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



We can initialize temp by using the FP with a negative displacement.

Stack Frame Example

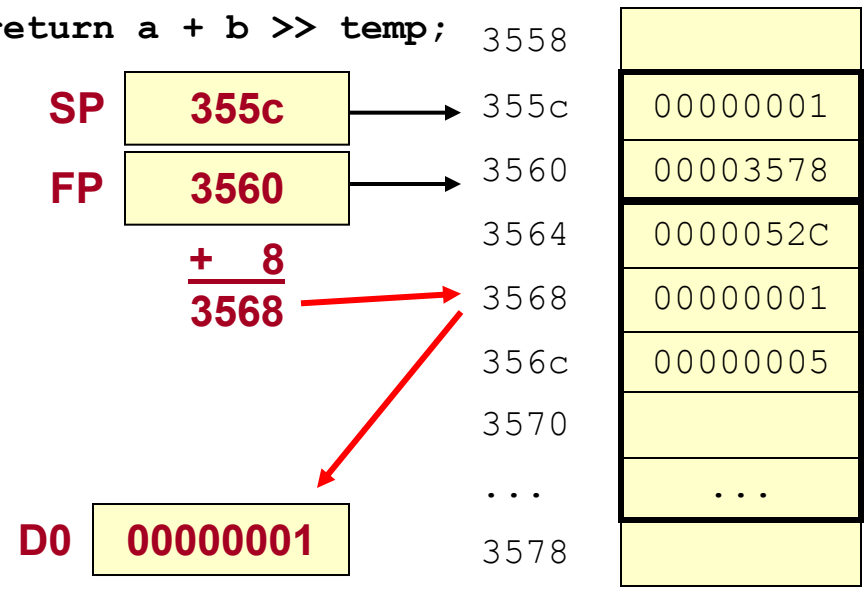
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



8(A6) causes us to pull out the first parameter of 1 and put it into D0

Stack Frame Example

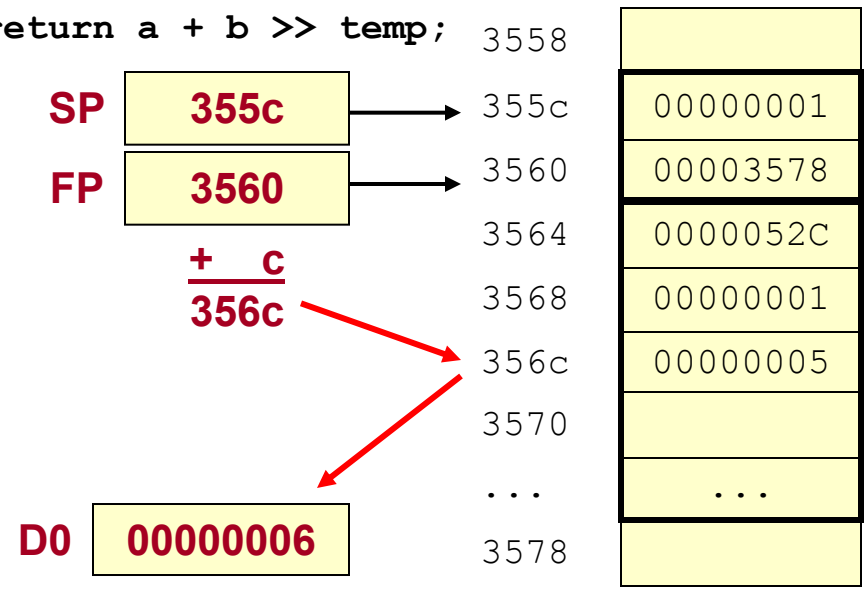
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



Adding 12(A6) to D0 adds the 2nd parameter to the 1st.

Stack Frame Example

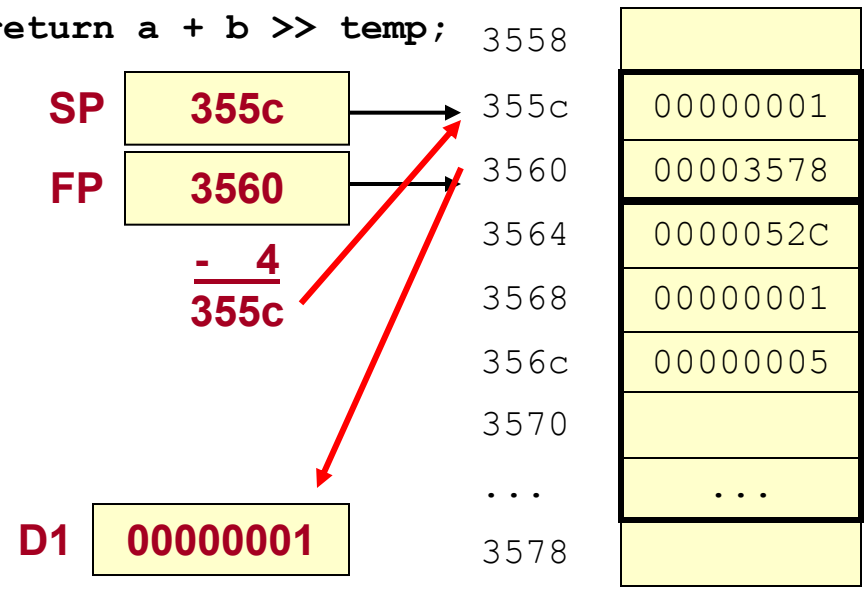
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



Place the temp value of 0001 into D1 to be used as a shift count

Stack Frame Example

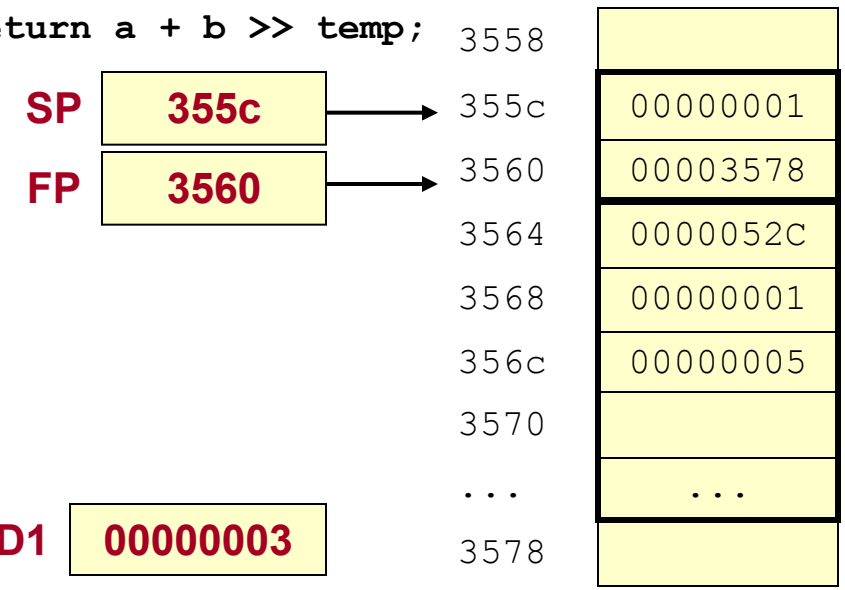
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



Shifting D0 right by 1 bit divides by 2 and we get the average in D0 where it will be expected by MAIN as the return value

Stack Frame Example

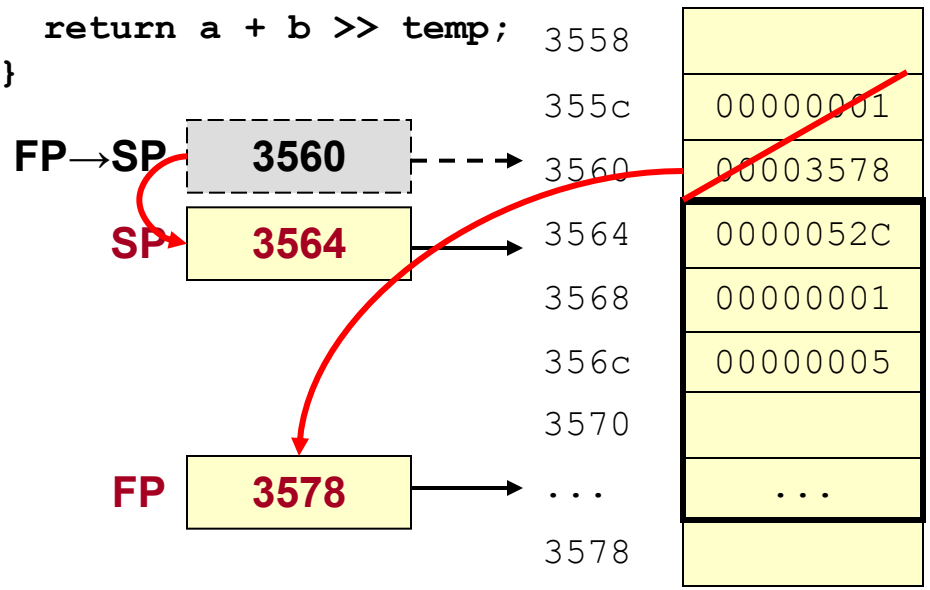
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



UNLK causes the value in A6 to be copied into SP and the old FP stored on the stack to then be popped into A6 followed by a postincrement of SP by 4. This restores the old FP for MAIN

Stack Frame Example

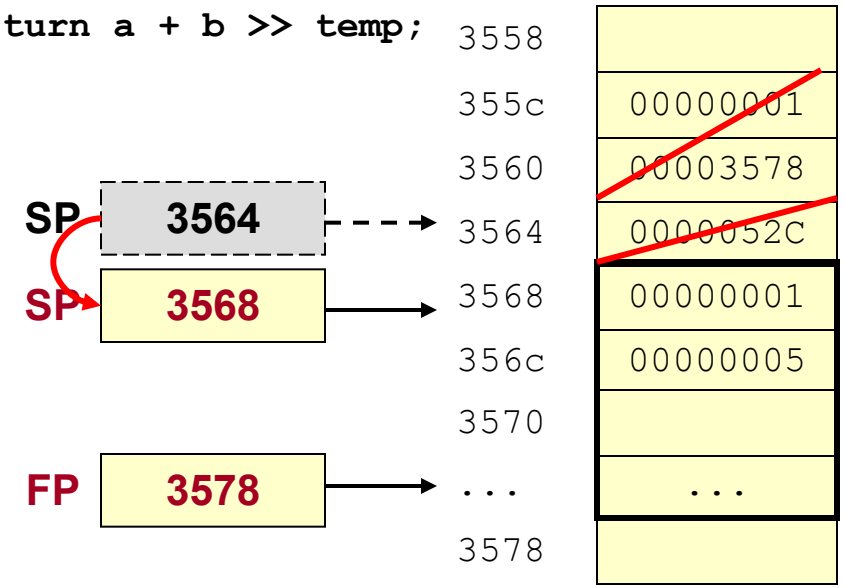
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



RTS causes the RA to be popped off the stack and execution returned to that point in MAIN.

Stack Frame Example

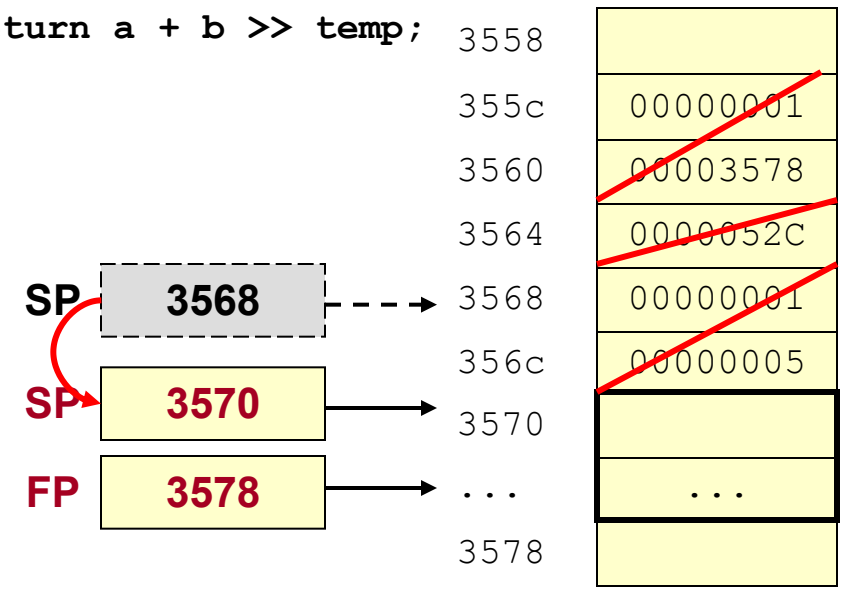
```
ANS:     .space    4

MAIN:    ...
         MOVE.L   #5,-(SP)
         MOVE.L   #1,-(SP)
         BSR.W    AVG
0x052C   ADDA.L   #8,SP
         MOVE.L   D0,ANS

         .org 0x0200
AVG:     LINK     A6,#-4
         MOVE.L   #1,-4(A6)
         MOVE.L   8(A6),D0
         ADD.L    12(A6),D0
         MOVE.L   -4(A6),D1
         LSR.L    D1,D0
         UNLK     A6
         RTS
```

```
int ans;
void main() {
  ans = avg(1,5);
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```



We can deallocate (pop) the parameters stored on the stack by simply pointing the SP below them and then write the return value in D0 to memory at ANS.

Example 2

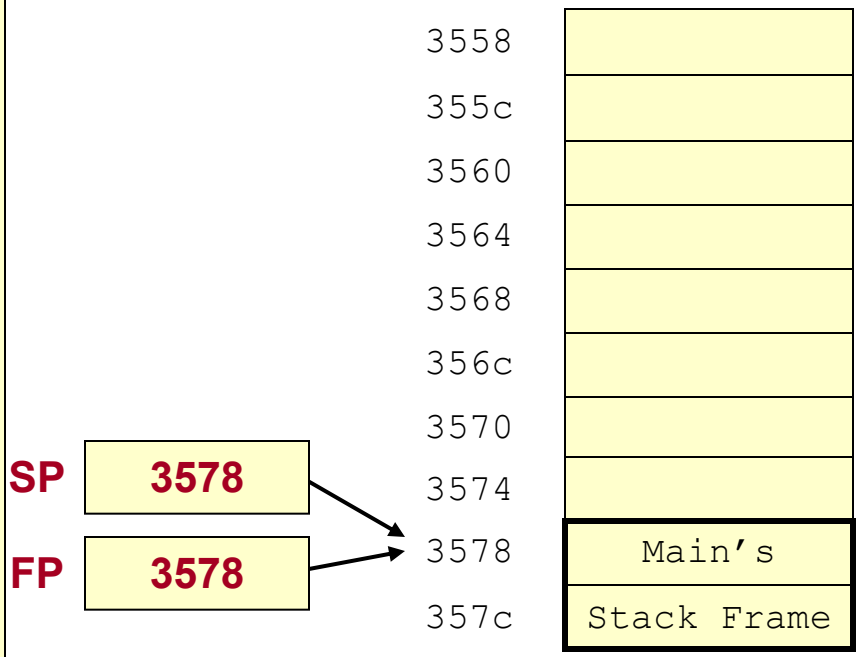
- Subroutine to sum an array of integers

```
int sumit(int dat[], int length)
{
  int sum, i;
  sum = 0;
  for(i=0; i < length; i++)
    sum = sum + data[i];
  return sum;
}
```

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move init i to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

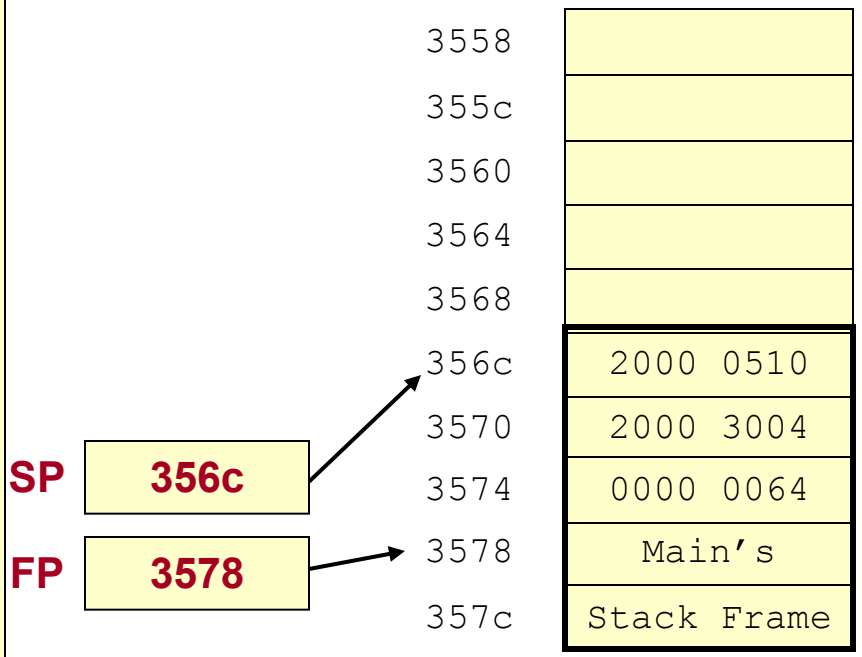


Originally for MAIN, FP points to 0x3578 and SP points to 0x3578

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move init i to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

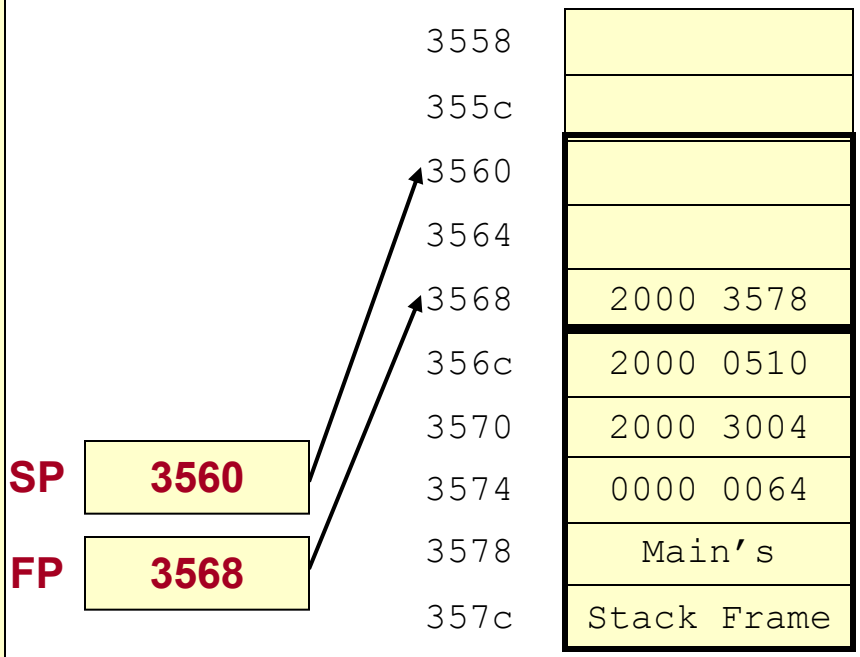


Parameters are pushed onto stack in reverse order (1st the length = 100 = 0x64, then the DAT pointer) and then the subroutine is called which pushes the RA

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move init i to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

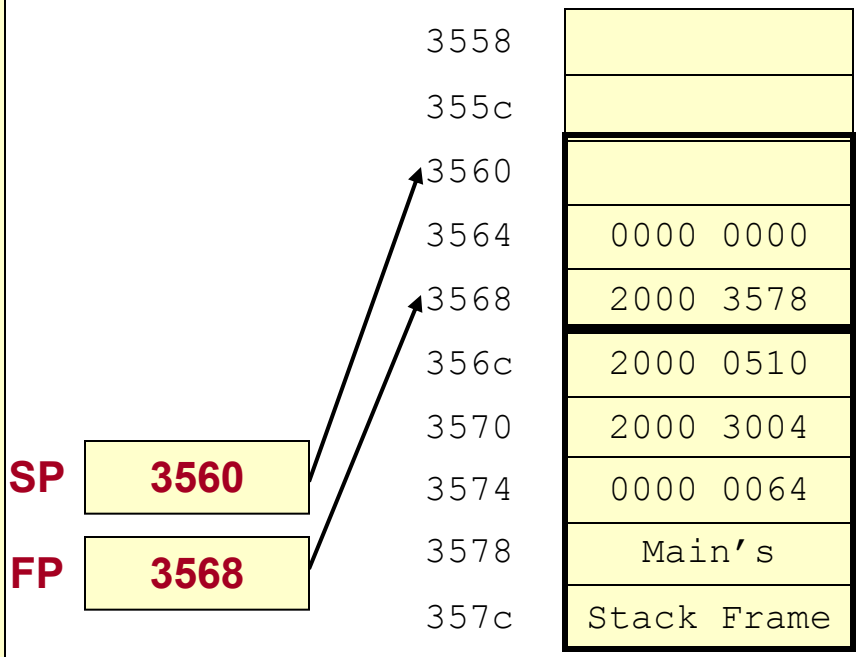


LINK pushes the old FP on the stack, sets FP = SP, and then allocates the total space for the locals (int sum and int i) to the SP

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move length to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

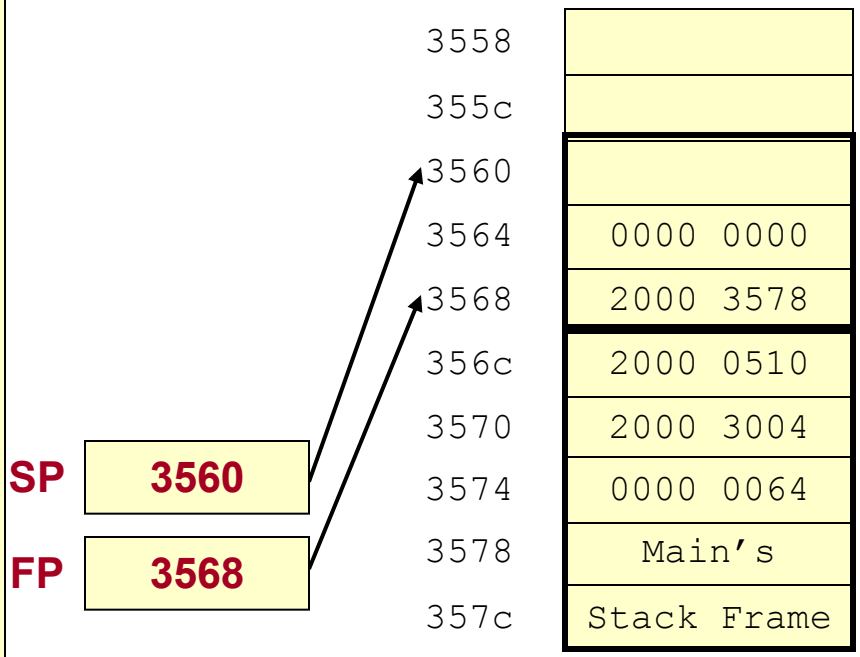


Initialize sum = 0 and then move it to D0 for faster access

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move length to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

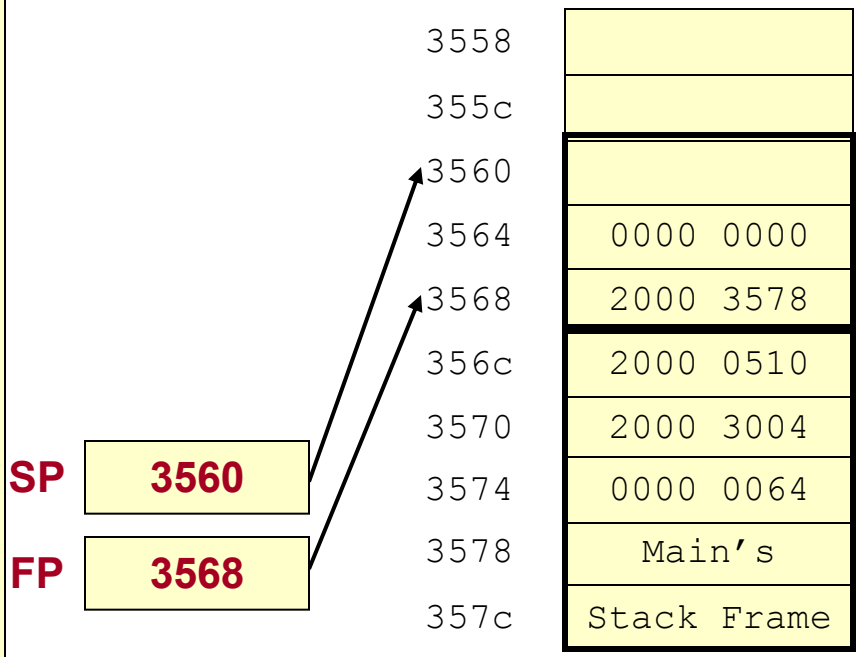


Initialize pointer to DAT, init i with length and move it to D1 for fast access. We will start at 100 and decrement until we get to 0. Note that -8(A6) is reserved in stack for i, however the optimizer of the compiler does not see the need here to initialize -8(A6)

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move length to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

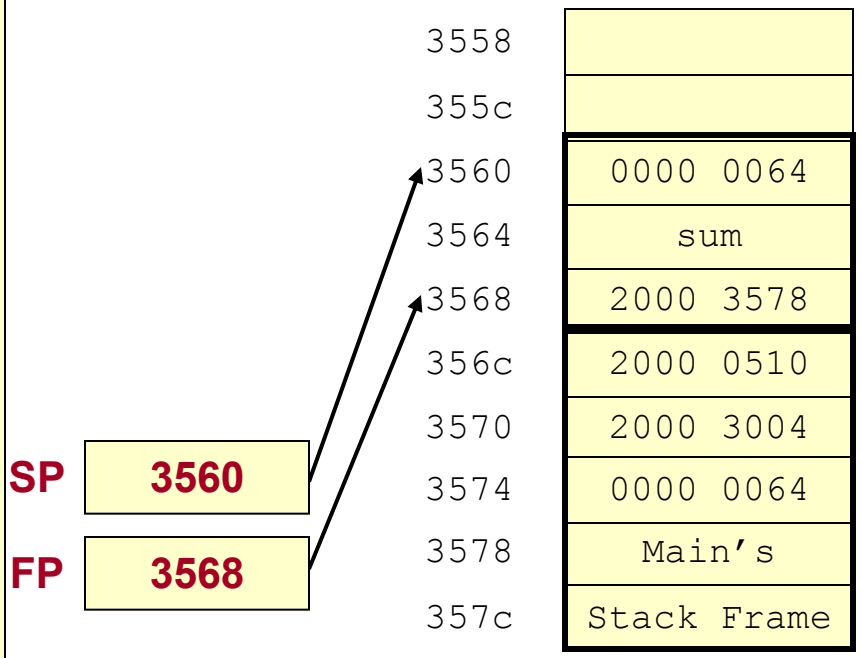


Iterate through the loop 100 times

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move length to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

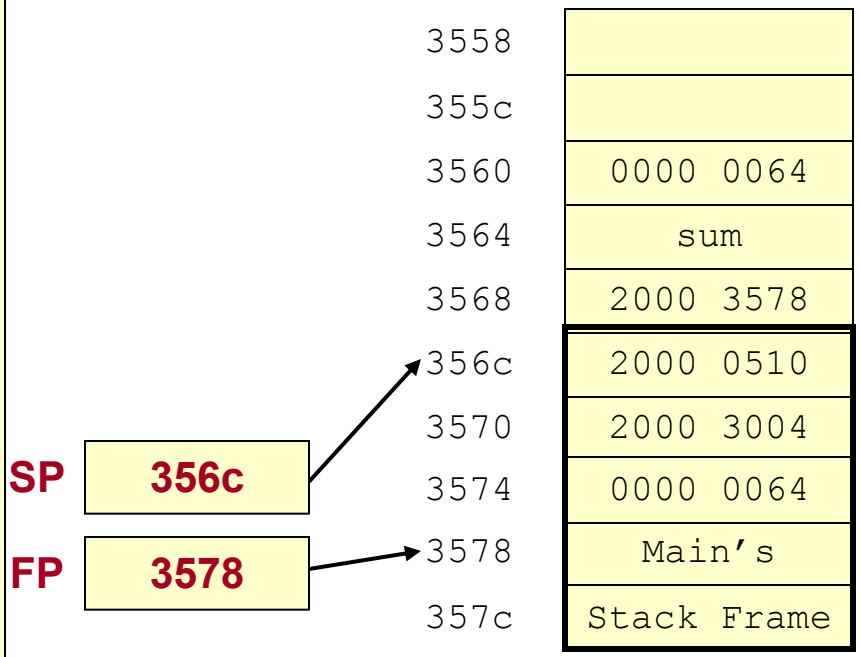


Update the memory versions of the variables. Here -8(A6) is updated with length of i, i.e., 0x64 and -4(A6) with sum

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move length to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

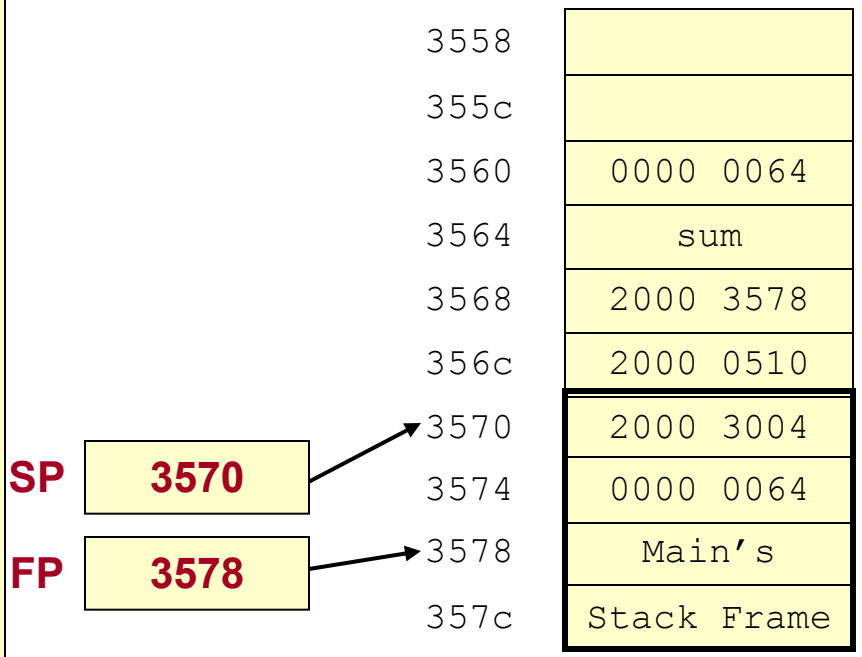


UNLK sets the SP = FP (deallocating locals), then pops the saved FP on the stack into the FP register

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move length to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```

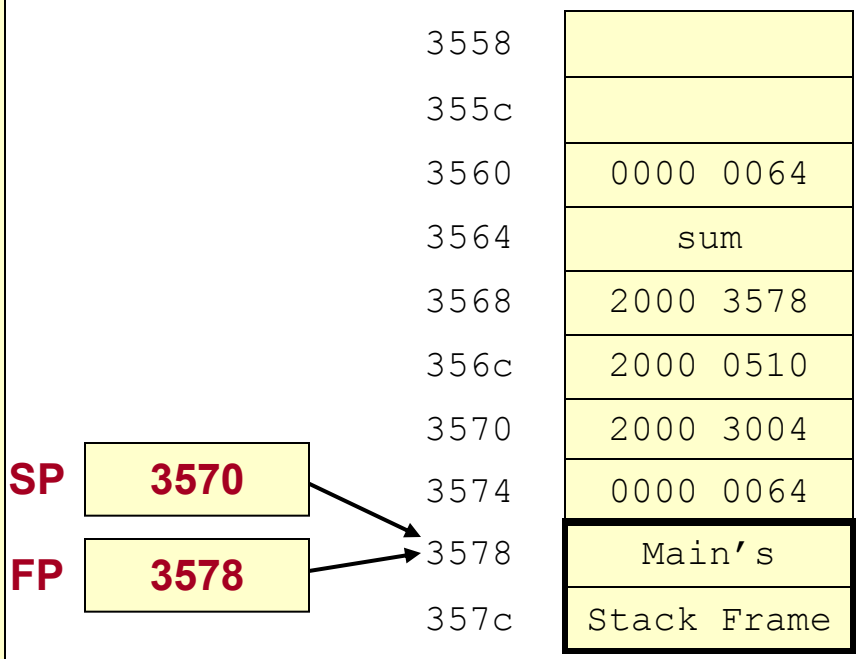


Pop the RA and return to MAIN

Example 2

```
        .data
DAT:    .space   400
        .text
MAIN:   MOVE.L   #100,-(SP)
        MOVE.L   #DATA,-(SP)
0x50c   BSR.W    SUMDAT
        ADDA.L   #8,SP

        .org     0x0300
SUMIT:  LINK     A6,#-8
        CLR.L    -4(A6)
        MOVE.L   -4(A6),D0      ; move sum to D0
        MOVE.L   8(A6),A0       ; move pointer
        MOVE.L   12(A6),D1      ; move length to D1
LOOP:   ADD.L    (A0)+,D0
        SUBI.L   #1,D1
        BNE      LOOP
DONE:   MOVE.L   #100,-8(A6)
        MOVE.L   D0,-4(A6)
        UNLK     A6
        RTS
```



Pop the parameters off the stack

What It Means To You…

- A compilers class (CS410) will go into more detail of how to organize the stack frame
- For now, make sure you realize and understand…
  - When you look at assembly output of compiled C code, all functions will access values from their stack frame
  - Local variables (those declared inside a function) are allocated to the stack (may not want to create that 2 MB array inside your function)
  - A register usually called the frame pointer will be often be used as a pointer to some fixed location in the frame, with values being accessed at a particular displacement