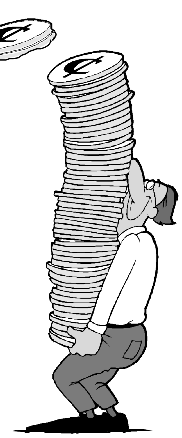
6004 Spring 1998: L24

Stack Organization
(As you push, so shall you pop!)



1

Recap: Procedures

```
int Fact( N )
int N ;
{
  int  F ;

  F = 1 ;
  while ( N > 0) {
    F = F * N ;
    N = N - 1 ;
  }
  return ( F ) ;
}
```

Need mechanisms for:

- **Passing arguments**
- **Local storage**
- **Returning result**
- **Linking control**

2

6004 Spring 1998: L24

Recap: Static Allocation

Given

int x, y, z ;

...

z = x + y ;

Translate as follows:

- **Assign fixed (constant) addresses to variables *x*, *y* and *z***
- **Translate each variable reference to a direct read/write of the assigned memory location**

Storage allocated at <u>compile</u> time!

3

Problems with Static Allocation

```
int Fact( N )
int N ;
{
  if ( N > 0)
    return Fact( N-1 ) * N ;
  else
    return 1 ;
}
```

Fact* assumes *N* is in *0x1024

Overwrite value of *N* with *N-1* when we call *Fact(N-1)* from *Fact (N)* !

Problems:

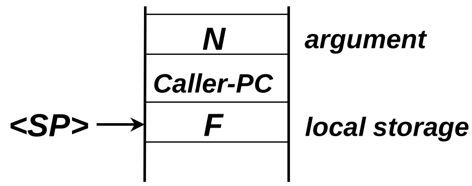
1. **Nested procedure calls**
2. **Recursion**

We need a <u>STACK</u> !

4

Stacks

- 1960’s revolution in storage management



- Easy to implement using contiguous memory and a “stack pointer” register
- Used for:
  – Procedure calls and returns. Stack pointer points to storage locations of active procedure
  – Automatic allocation of local variables

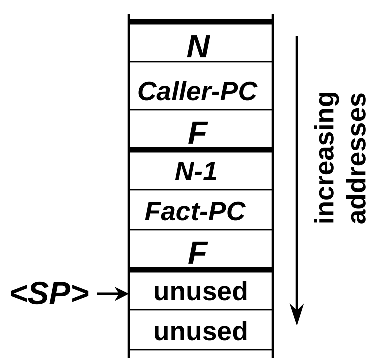
5

Stack Implementation

Conventions:

Builds UP on *push*

Stack pointer points to first unused location



To push <x>

SP ← <SP> + 4

<SP> - 4 ← <x>

To pop( ) a value into x

x ← < <SP> - 4 >

SP ← <SP> - 4

6

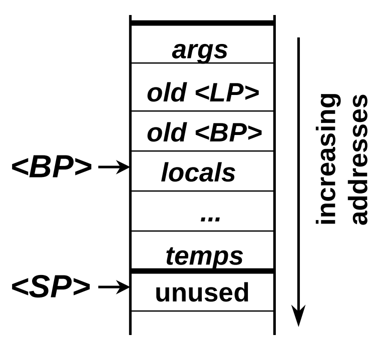
Activation Records

- **Storage needs in a procedure**
  - **Arguments *x*, *y*, *z* in *f(x, y, z)***
  - **Return address of routine that called *f* (caller routine)**
  - **Result passed back to caller routine**
  - **Local variables in procedure *f***
  - **Temporary storage during expression evaluation in *f*, e.g., *k = (c + d) * (c - d)***

- **Each of the above is specific to an *activation* of a procedure, and collectively, they may be viewed as an activation record**

7

Activation Record a.k.a. Stack Frame

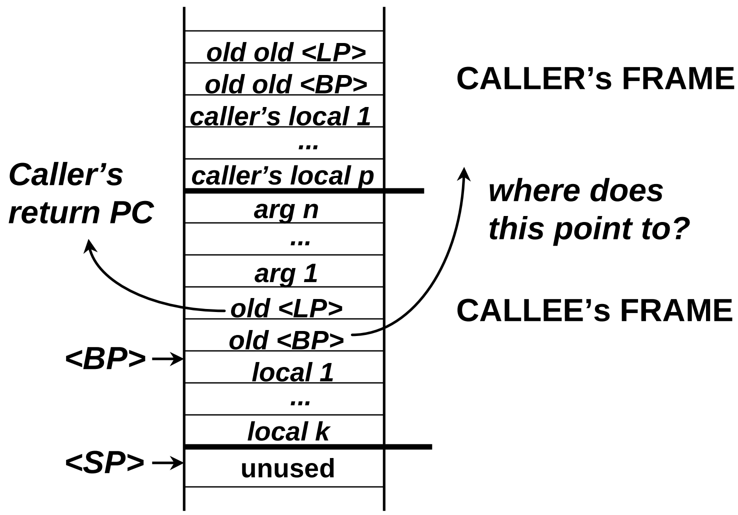


reserved registers

r27 = BP Base pointer: ***Points to 1st local.***
r28 = LP Linkage pointer: ***Saved <PC>.***
r29 = SP Stack pointer: ***Points to 1st unused location.***
r30 = XP Exception pointer: ***Saved <PC>.***

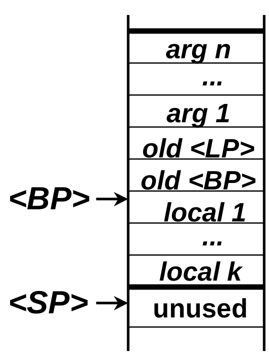
8

Stack Frame Details



9

Access to Local Frame



To access jth local variable:

LD(BP, _______, rx)
ST(rx, ________, BP)

To access jth argument:

LD(BP, _______, rx)
ST(rx, ________, BP)

Push arguments in reverse order so we can accommodate variable # arguments

10

6004 Spring 1998: L24

Stack Macros

PUSH(rx)	**push 4-byte value onto stack**
ADDC(SP, 4, SP)	
ST(rx, -4, SP)	
POP(rx)	**pop 4-byte value into *rx***
LD(SP, -4, rx)	
SUBC(SP, 4, SP)	
ALLOCATE(k)	**reserve *k* words of stack**
ADDC(SP, 4*k, SP)	
DEALLOCATE(k)	**give back *k* words**
SUBC(SP, 4*k, SP)	

11

Procedure Linkage Contract

Calling Sequence:

PUSH(arg n)	**Push arguments in**
…	**right to left (reverse)**
PUSH(arg 1)	**order.**
BEQ(r31, f, LP)	**Call *f*.**
DEALLOCATE(n)	**Clean up.**

Entry Sequence:

PUSH(LP)	**Save *<LP> <BP>***
PUSH(BP)	**for new calls.**
ADDC(SP, 0, BP)	**Set *BP* to frame base.**
ALLOCATE(locals)	**Allocate locals.**
(PUSH other registers)	**Preserve registers used.**

12

Procedure Linkage Contract - II

Exit and Return Sequence:

(POP other registers)	**Restore registers.**
ADDC(rval, 0, r0)	**Result in *r0*.**
ADDC(BP, 0, SP)	**Strip locals.**
POP(BP)	**Back to caller frame.**
POP(LP)	**Obtain return address.**
JMP(LP, r31)	**Return to caller.**

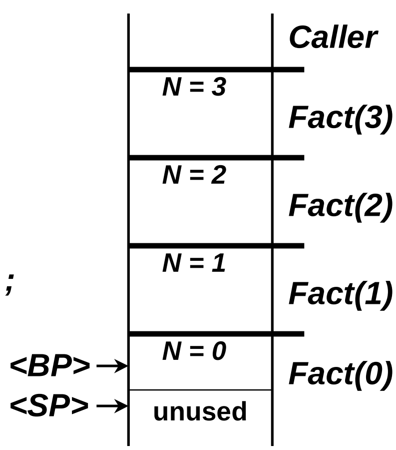
Stack pointer before and after Calling sequence should be the same!

13

Recursive Factorial Stack Frames

```
        ...
z = Fact(3) ;
        ...

int Fact( N )
int N ;
{
   if ( N > 0)
     return Fact( N-1 ) * N ;
   else
     return 1 ;
}
```



14

Recursive Factorial Assembly - I

z = Fact(N) ;

Calling sequence	0	*PUSH(arg 1)*	Push argument
		BEQ(r31, Fact, LP)	Call *Fact*.
		DEALLOCATE(1)	Clean up.
	Fact:	*PUSH(LP)*	
Entry sequence	1	*PUSH(BP)*	
		ADDC(SP, 0, BP)	
	2	*PUSH(r1)*	
Body of *Fact*		*LD(BP, -12, r1)*	*r1 = N*
		BNE(r1, recur, r31)	*if (N > 0)*
		ADDC(r31, 1, r0)	
		BEQ(r31, rtn, r31)	*else return 1 ;*

Assume ***N*** is non-negative!

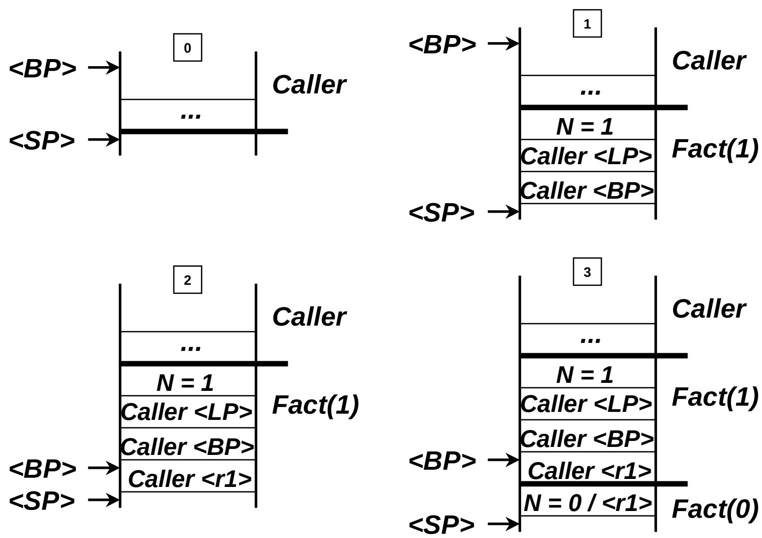
15

Recursive Factorial Assembly - II

Body of *Fact* (contd.)	*recur:*	*SUBC(r1, 1, r1)*	*r1 = N - 1*
	3	*PUSH(r1)*	push arg 1
		BEQ(r31, Fact, LP)	call *Fact(N - 1)*
	4	*DEALLOCATE(1)*	
		LD(BP, -12, r1)	*r1 = N*
		MUL(r1, r0, r0)	*r0 = Fact(N - 1) * N*
Exit/Return sequence	*rtn:*	*POP(r1)*	restore register
		ADDC(BP, 0, SP)	strip locals
		POP(BP)	
	5	*POP(LP)*	Restore link
		JMP(LP, r31)	return

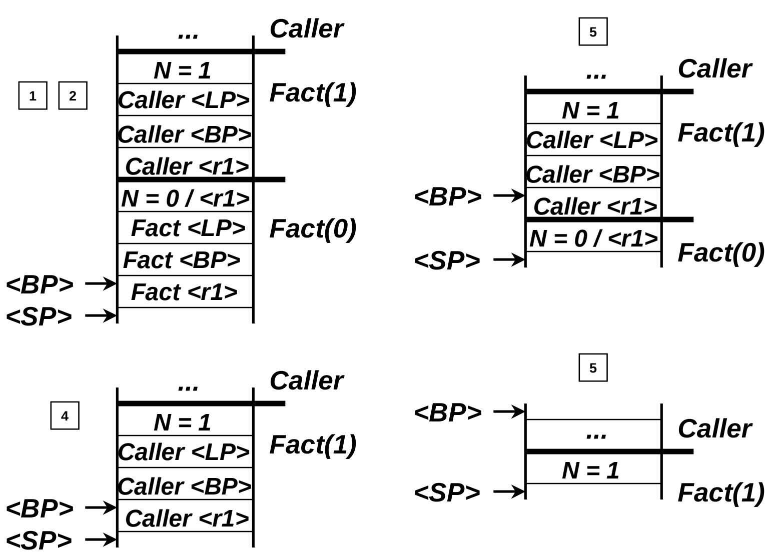
16

Snapshots of Stack Frames - I



17

Snapshots of Stack Frames - II



18