COMP 303
Computer Architecture
Lecture 3

Supporting procedures in computer hardware

- The execution of a procedure
  - Place parameters in a place where the procedure can access
  - Transfer control to the procedure
  - Acquire the storage resources needed for the procedure
  - Perform the desired task
  - Place the result value in a place where the calling program can access
  - Return control to the point of origin, since a procedure can be called from several points in a program

Register usage conventions

- \$a0-\$a3: four argument registers in which to pass parameters
- \$v0-\$v1: two value registers in which to return values
- \$ra: one return address register to return to the point of origin

- The jump-and-link instruction (`jal`): jumps to an address and simultaneously saves the address of the following instruction in register \$ra

```
jal ProcedureAddress
```

Program counter

- We need a register to hold the address of the current instruction being executed
  - “Program Counter” (due to historical reasons) PC in MIPS
- jal saves `PC+4` in register $ra
- At the end of the procedure we jump back to the $ra (an unconditional jump)

```
jr $ra
```

- The caller puts the parameter values in $a0-$a3
- The caller uses `jal X` to jump to procedure X
- The callee performs the calculations, places the results in $v0-$v1
- Returns control to the caller by `jr $ra`

Stack

- Suppose the procedure needs more than 4 arguments
- We store the values in Stack (a last-in-first-out queue)
- A stack needs a pointer to the most recently allocated address in the stack: stack pointer
- Placing data onto the stack is called a Push. Removing data from the stack is called a Pop.
- The stack pointer in MIPS is $sp. <u>By convention stacks “grow” from higher addresses to lower addresses!!!</u> (You push values onto the stack by subtracting from the stack pointer)

Procedure call

- When making a procedure call, it is necessary to
  1. Place inputs where the procedure can access them
  2. Transfer control to procedure
  3. Acquire the storage resources needed for the procedure
  4. Perform the desired task
  5. Place the result value(s) in a place where the calling program can access it
  6. Return control to the point of origin
- MIPS
  - Provides instructions to assist in procedure calls (jal) and returns (jr)
  - Uses software conventions to
    - place procedure input and output values
    - control which registers are saved/restored by caller and callee
  - Uses a software stack to save/restore values

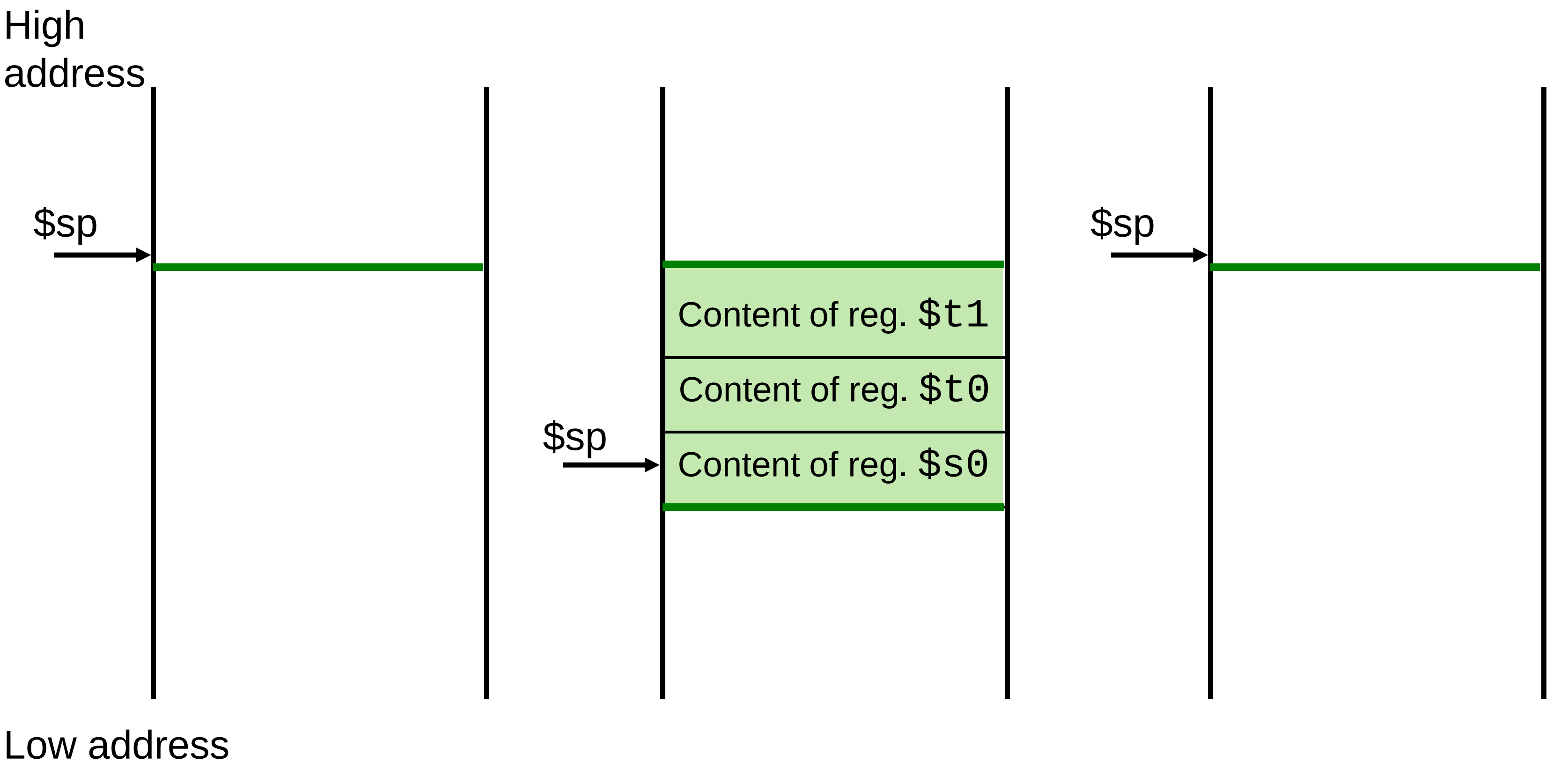
A procedure call with a stack

```
int leaf-example (int g, int h, int i, int j)
{
   int f;
   f = (g+h)-(i+j);
   return f;
}
```

Assume the parameter variables g, h, i, and j correspond to the argument registers $a0, $a1, $a2, and $a3, and f corresponds to $s0.

```
leaf_example:
        sub   $sp, $sp, 12       # adjust stack to make room for 3 items
        sw    $t1, 8($sp)        # save register $t1 for use afterwards
        sw    $t0, 4($sp)        # save register $t0 for use afterwards
        sw    $s0, 0($sp)        # save register $s0 for use afterwards
        add   $t0, $a0, $a1      # register $t0 contains g + h
        add   $t1, $a2, $a3      # register $t1 contains i + j
        sub   $s0, $t0, $t1      # register $s0 contains (g + h) - (i + j)
        add   $v0, $s0, $zero    # register $v0 contains the result
        lw    $s0, 0($sp)        # restore register $s0 for caller
        lw    $t0, 4($sp)        # restore register $t0 for caller
        lw    $t1, 8($sp)        # restore register $t1 for caller
        add   $sp, $sp, 12       # adjust stack to delete 3 items
        jr    $ra                # jump back to calling routine
```

A procedure call with a stack (cont'd)



Before procedure call **During procedure call** **After procedure call**

Some register conventions

Register	Name	Convention
R0	$zero	Constant 0
R1	$at	Reserved for assembler
R2	$v0	Return Values
R3	$v1	
R4	$a0	Procedure arguments
R5	$a1	
R6	$a2	
R7	$a3	
R8	$t0	Caller saved temporaries: may be overwritten by called procedures
R9	$t1	
R10	$t2	
R11	$t3	
R12	$t4	
R13	$t5	
R14	$t6	
R15	$t7	

Register	Name	Convention
R16	$s0	Callee saved temporaries: may not be overwritten by called procedures
R17	$s1	
R18	$s2	
R19	$s3	
R20	$s4	
R21	$s5	
R22	$s6	
R23	$s7	
R24	$t8	Caller save temp
R25	$t9	
R26	$k0	Reserved for operating system
R27	$k1	
R28	$gp	Global pointer
R29	$sp	Stack pointer
R30	$s8	Callee save temp
R31	$ra	Return address

Recursion (Nested procedure call)

```
int fact (int  n)
{
    if (n < 1)
            return 1;
    else
            return (n * fact(n-1));
}
```

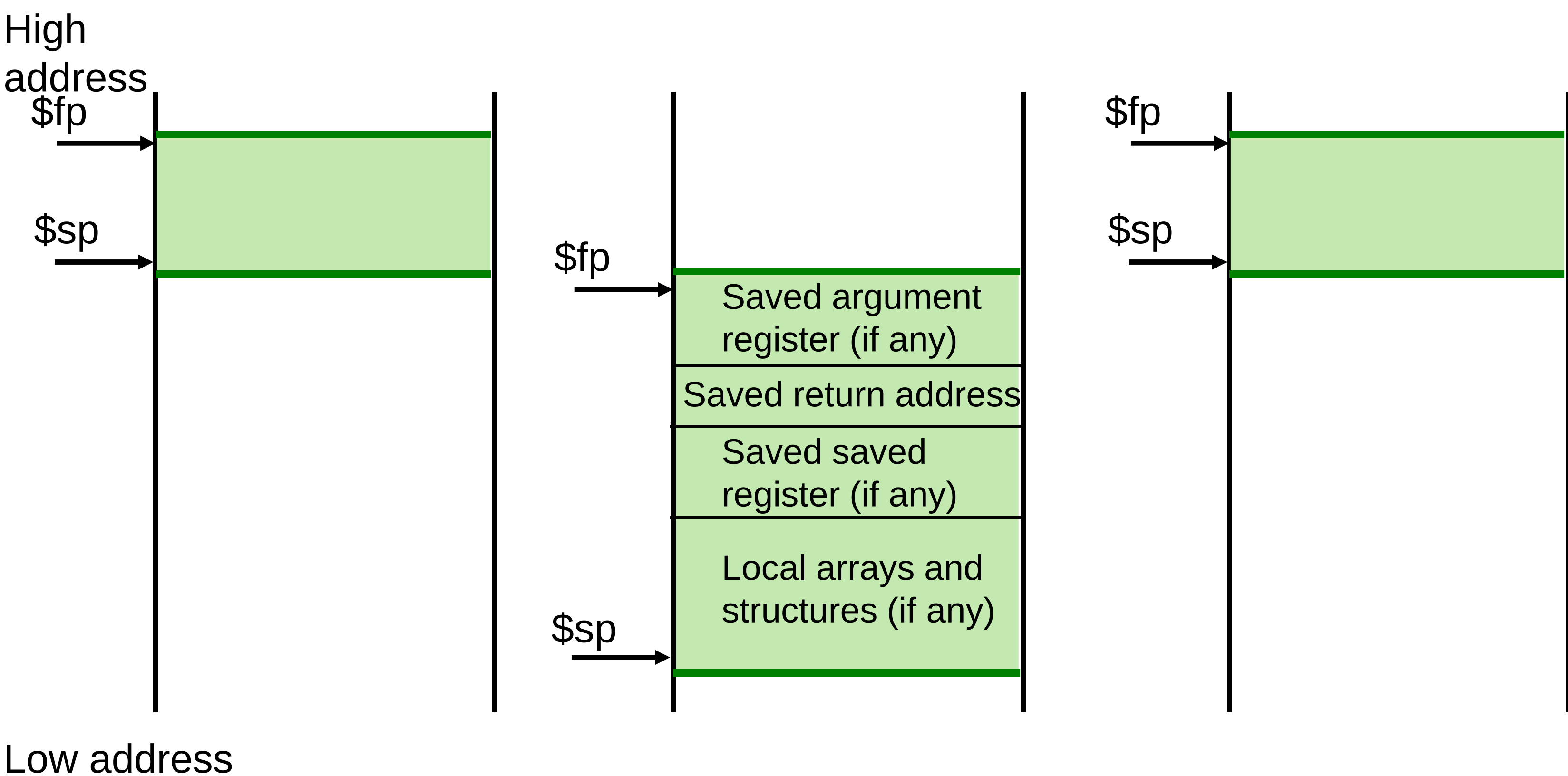
Recursion

```
fact:
    addi $sp, $sp, -8       # adjust stack for 2 items
    sw   $ra, 4($sp)        # save the return address
    sw   $a0, 0($sp)        # save the argument n

    slti $t0, $a0, 1        # test for n<1
    beq  $t0, $zero, L1     # if n>=1, goto L1
    addi $v0, $zero, 1      # return 1
    addi $sp, $sp, 8        # pop 2 items off stack
    jr   $ra                # return to after jal
L1:
    addi $a0, $a0, -1       # n>=1: argument gets (n-1)
    jal  fact               # call fact with (n-1)
    lw   $a0, 0($sp)        # return from jal: restore argument n
    lw   $ra, 4($sp)        # restore the return address
    addi $sp, $sp, 8        # adjust stack pointer to pop 2 items

    mul  $v0, $a0, $v0      # return n*fact(n-1)
    jr   $ra
```

Stack allocation in MIPS

- The stack is also used to store variables that are local to the procedure that do not fit in registers (local arrays or structures)
- The segment of the stack containing a procedure's saved registers and local variables is called a procedure frame or activation record.
- Some MIPS software use a frame pointer ($fp) to point to the first word of the frame of a procedure

Stack allocation in MIPS



Before procedure call **During procedure call** **After procedure call**

Policy of use conventions

Name	Register number	Usage	Preserved on call?
`$zero`	0	the constant value 0	n.a.
`$v0-$v1`	2-3	values for results and expression evaluation	no
`$a0-$a3`	4-7	arguments	yes
`$t0-$t7`	8-15	temporaries	no
`$s0-$s7`	16-23	saved	yes
`$t8-$t9`	24-25	more temporaries	no
`$gp`	28	global pointer	yes
`$sp`	29	stack pointer	yes
`$fp`	30	frame pointer	yes

Register 1 ($at) reserved for assembler, 26-27 for operating system

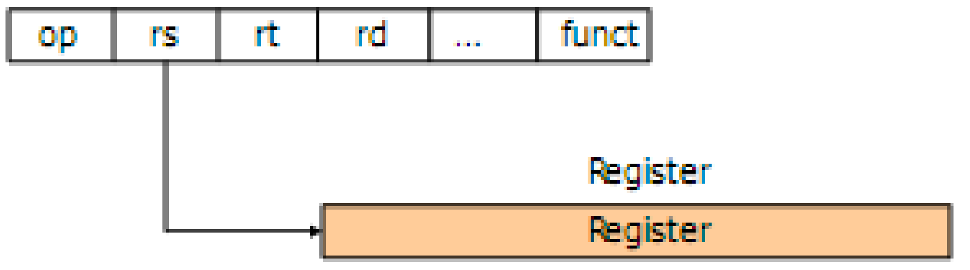
Global pointer

- C has two storage classes: automatic and static
  - Automatic: variables that are local to a procedure and are discarded when the procedure exits
  - Static: they exist across exits from and entries to procedures. C variables declared outside all procedures are considered static (or those declared with keyword `static`)
- To simplify access to static data MIPS uses global pointer or $gp

MIPS addressing

- Register addressing where the operand is a register

1. Register addressing (R-Type)



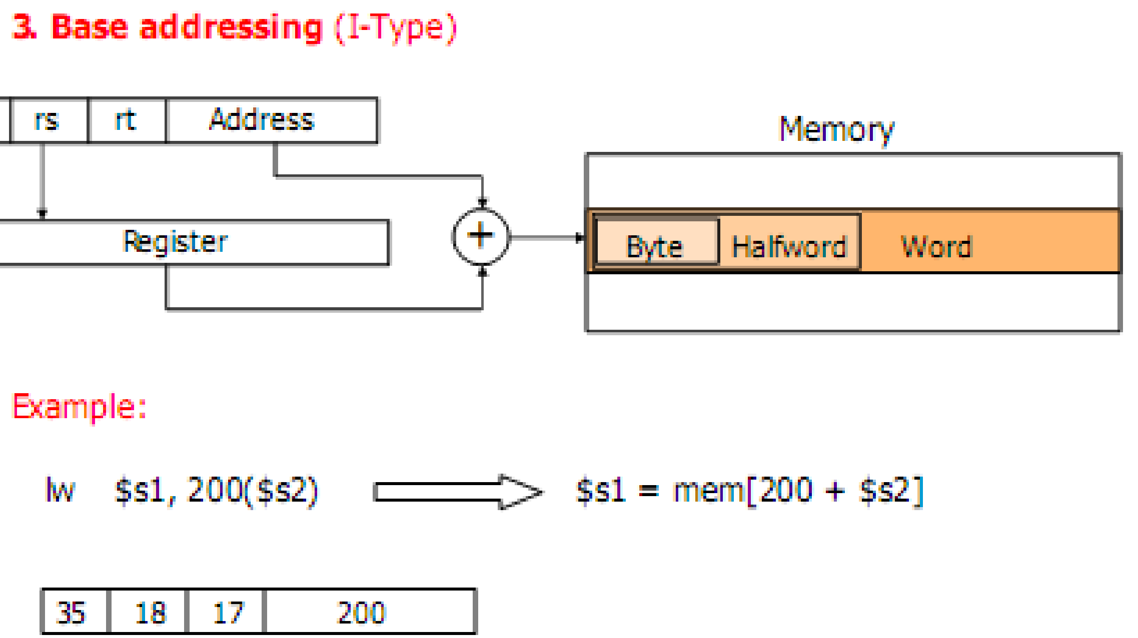
Example:

add \$s1, \$s2, \$s3 ⇒ \$s1 = \$s2 + \$s3

0	18	19	17	0	32

MIPS addressing

- Base or displacement addressing where the operand is at the memory location whose address is the sum of a register and a constant in the instruction



MIPS addressing

- Immediate addressing where the operand is a constant within the instruction itself

2. Immediate addressing (I-Type)

op	rs	rt	immediate

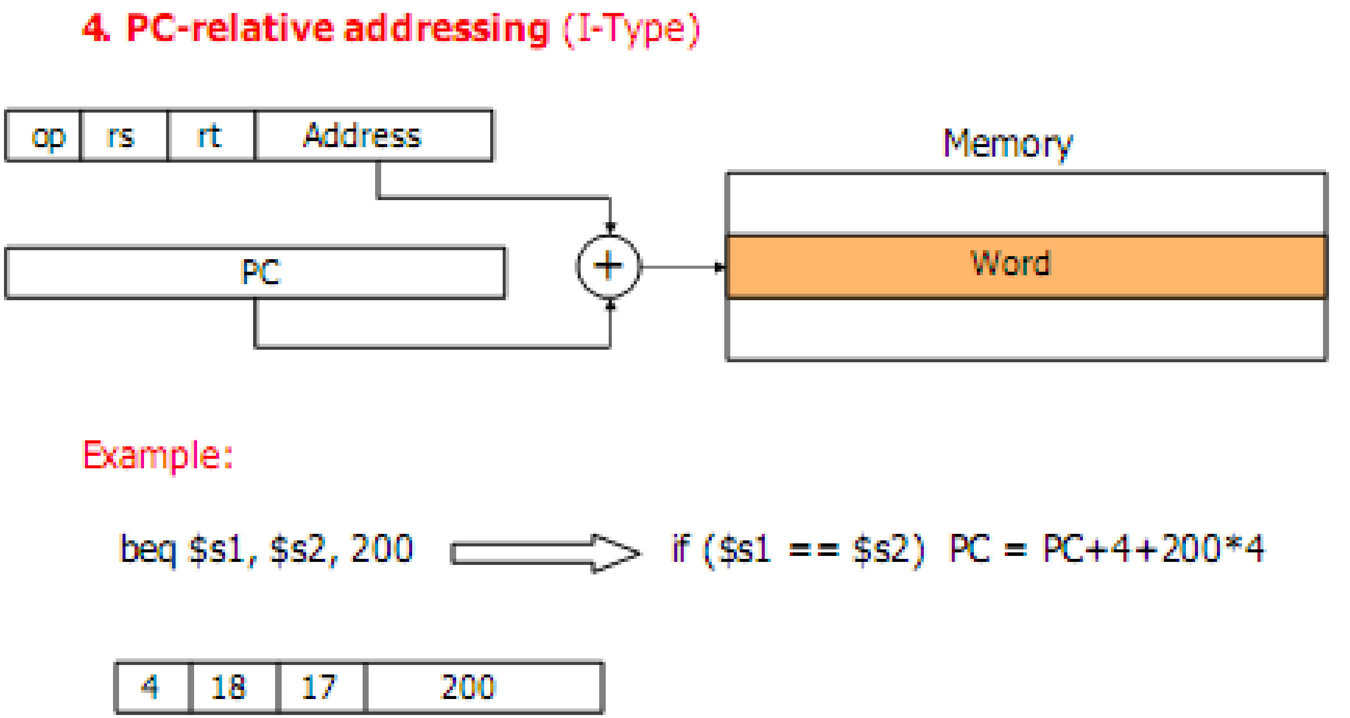
Example:

addi $s1, $s2, 200 ⇒ $s1 = $s2 + 200

8	18	17	200

MIPS addressing

- PC-relative addressing where the address is the sum of the PC and a constant in the instruction



MIPS addressing

- Pseudodirect addressing where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC