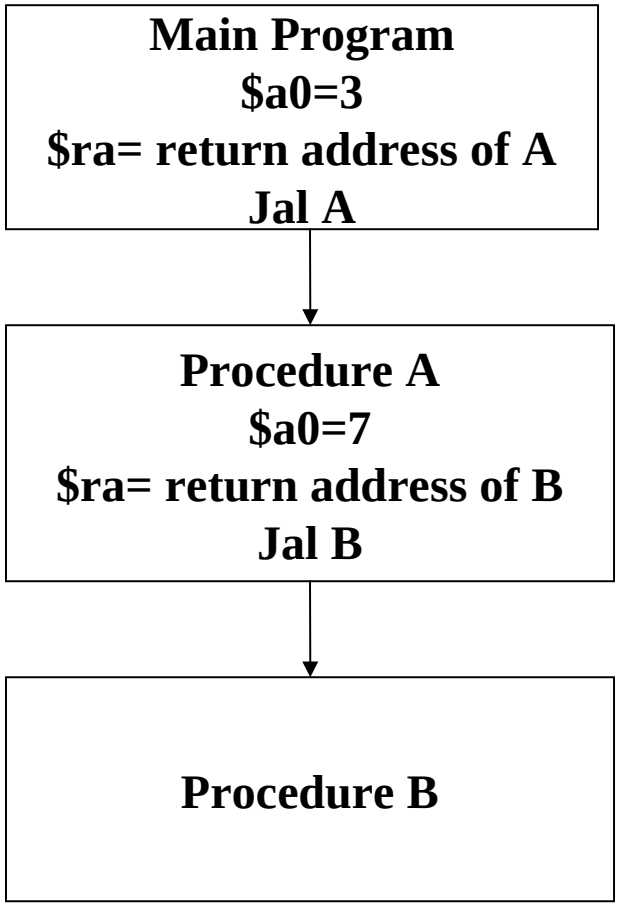
Lecture 4: Instructions: Language of the Computer

CSE-2204: Computer Architecture and Organization

Nested Procedure



Solution:

- ✓ The procedure pushes register values into the stack.
  - Caller pushes argument register and temporary registers.
  - Callee pushes saved registers and $ra.
- ✓ $sp is adjusted to account for the number of registers placed on the stack.

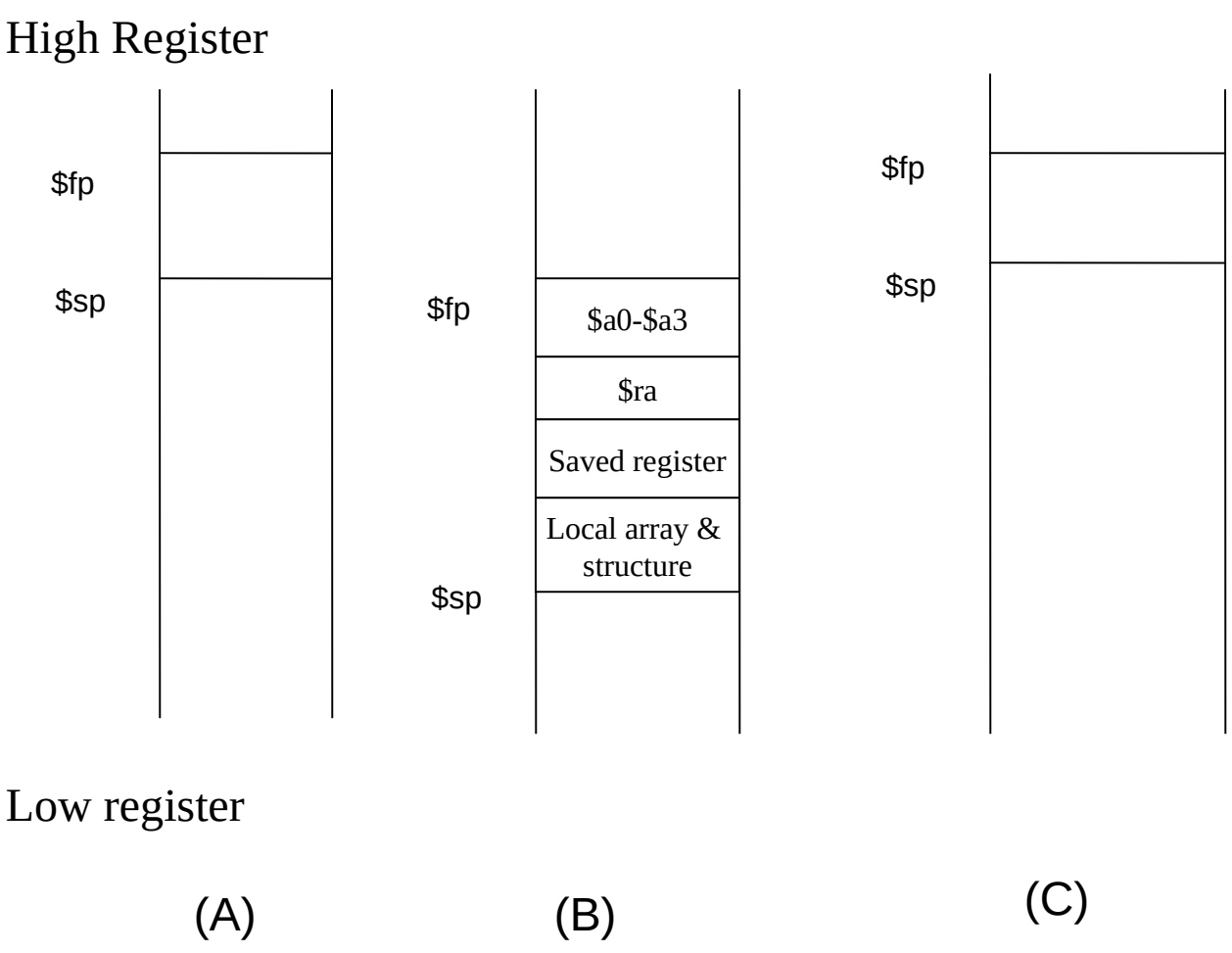
Example

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

fact:

- addi $sp, $sp, -8
- sw $ra, 4($sp)
- sw $a0, 0($sp)
- slti $t0, $a0, 1 #test for n< 1
- beq $t0, $zero, L1 #if n>=1, go to L1
- addi $v0,$zero, 1 # return 1
- addi $sp, $sp, 8 #pop 2 items of stack
- jr $ra #$ra and $a0 do not change
- L1: addi $a0, $a0, -1 #n>=1, $a0 = n-1
- jal fact
- lw $a0, 0($sp)
- lw $ra, 4($sp)
- addi $sp, $sp, 8
- mul $v0, $a0, $v0
- jr $ra

Allocating Space for New Data on the Stack



Procedure Frame / Activation Record:

The segment of the stack containing a procedure's saved registers and local variables.

Frame Pointer:

It points to the first word of a frame.

MIPS Register Convention

Name	Register Number	Usage
$zer0	0	The constant value 0
$v0-$v1	2-3	Values for results and expression evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved
$t8-$t9	24-25	More temporaries
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

MIPS Instruction for String Manipulation

1. Load Byte (lb):

   Loads a byte from memory, placing it in the rightmost 8 bits of a register.

   lb $t0, 0($sp)

2. Store Byte (sb):

   Takes a byte from the rightmost 8 bits of a register and writes it to memory.

   sb $t0, 0($gp)

MIPS Addressing for 32-bit Immediates and Addresses

32-bit Immediate Operands

Load Upper Immediate (lui) is used to set the upper 16-bits of a constant in a register, allowing a sub-sequent instruction to specify the lower 16-bits of the constant.

Example:

Consider the following 32-bit constant

0000 0000 0011 1101 0000 1001 0000 0000

1. lui $s0, 61 #61 = 0000 0000 0011 1101
2. The content of $s0 is

0000 0000 0011 1101 0000 0000 0000 0000

3. ori $s0, $s0, 2304 #2304 = 0000 1001 0000 0000

4. The final value of $s0 is

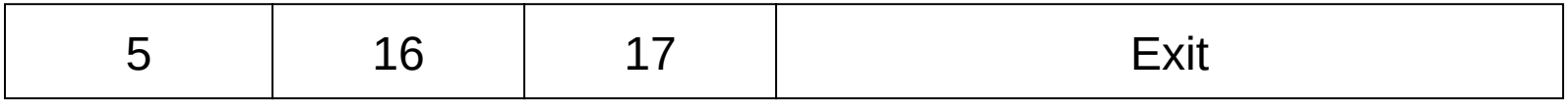
0000 0000 0011 1101 0000 1001 0000 0000

- Compiler or assembler must break large constants into pieces and then resembles them into a register.
- Creating 32-bit constant requires care.

32-bit Addressing in Branches and Jumps

Conditional Jump (Used in if and loop statements)

bne $s0,$s1, Exit



6 bits 5 bits 5 bits 16 bits

- ✓Branch instruction is calculated as: Program Counter = Register + Branch address
- ✓16 bit field represent number of words instead of number of bytes.

PC relative Addressing:

An addressing mode in which the address is the sum of the program counter (PC) and a constant in the instruction.

- ✓ MIPS address is relative to the address of the following instruction (PC+4)

32-bit Addressing in Branches and Jumps

Unconditional Jump (J) and Jump-and-link (JAL) (Used in procedure call)

JAL invokes procedures that have no reason to be near the call. So, MIPS uses long addresses for procedure call using j-type format.

6 bits 26 bits

Pseudodirect Addressing:

Temp:= PC[31:28].ADR.00

PC:=Temp

✓MIPS instruction always starts with a location address ended with 00.

32-bit Addressing in Branches and Jumps

Branching Far Away:

beq $s0, $s1, L1

Replaced with:

bne $s0, $s1, L2

j L1

L2:

Example

Consider the following loop,

```
while ( save [i]==k)
    i+=1;
```

MIPS Representation:

i and k correspond to $s3 and $S5 and the base of the array save is in $s6.

```
Loop: sll $t1, $s3, 2
        add $t1, $t1, $s6
        lw $t0, 0($t1)
        bne $t0, $s5, Exit
        addi $s3, $s3, 1
        j Loop
Exit:
```

Continuation of the Example

200	0	0	19	9	2	0
204	0	9	22	9	0	32
208	35	9	8	0		
212	5	8	21	2		
216	8	19	19	1		
220	2	50				
224	…….					

Addressing Modes in MIPS

1. Register addressing
2. Base or displacement or effective addressing
3. Immediate addressing
4. PC-relative addressing
5. Pseudo-direct addressing