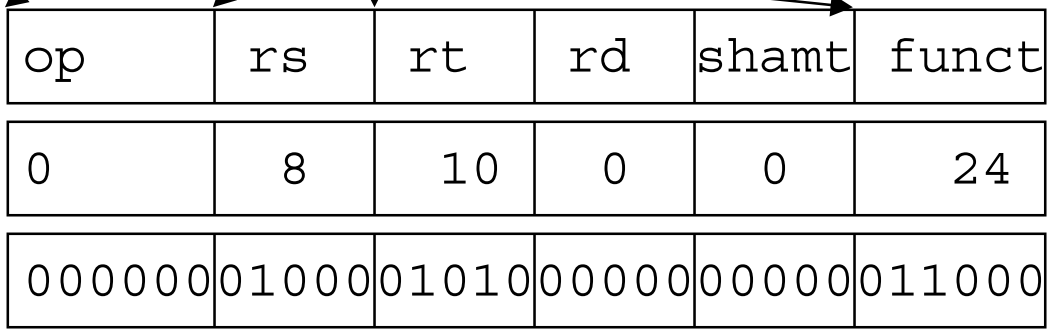
Jumping To Procedures

It beats jumping to conclusions

Multiply

- Multiply mult is an R-type instruction, but is complicated by the fact that the product of 2 32-bit numbers can be 64-bits long
- There is a special multiply result register

```
mult  $8,$10        # Go forth and multiply
```



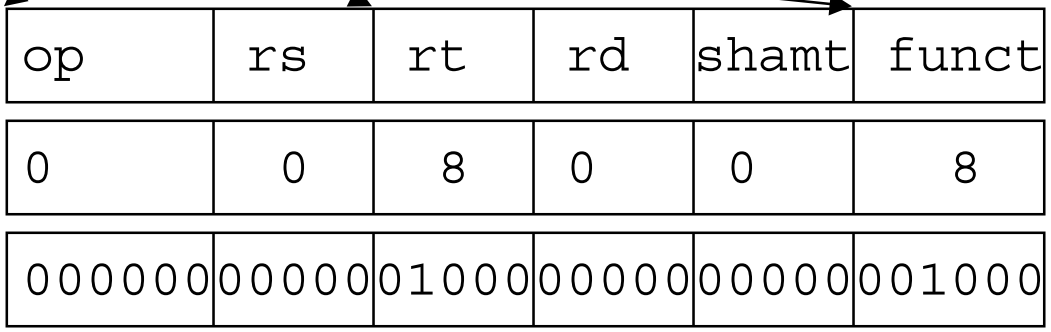
op	rs	rt	rd	shamt	funct
0	8	10	0	0	24
000000	01000	01010	00000	00000	011000

The result can be removed using **move from high**, **mfhi** to get the MSB's and **move from low**, **mflo**, to get the LSBs

Jumping

- The jr instruction has an R-type format and jumps to the address in the operand register

```
jr  $8        # Jump to address in Reg 8
```



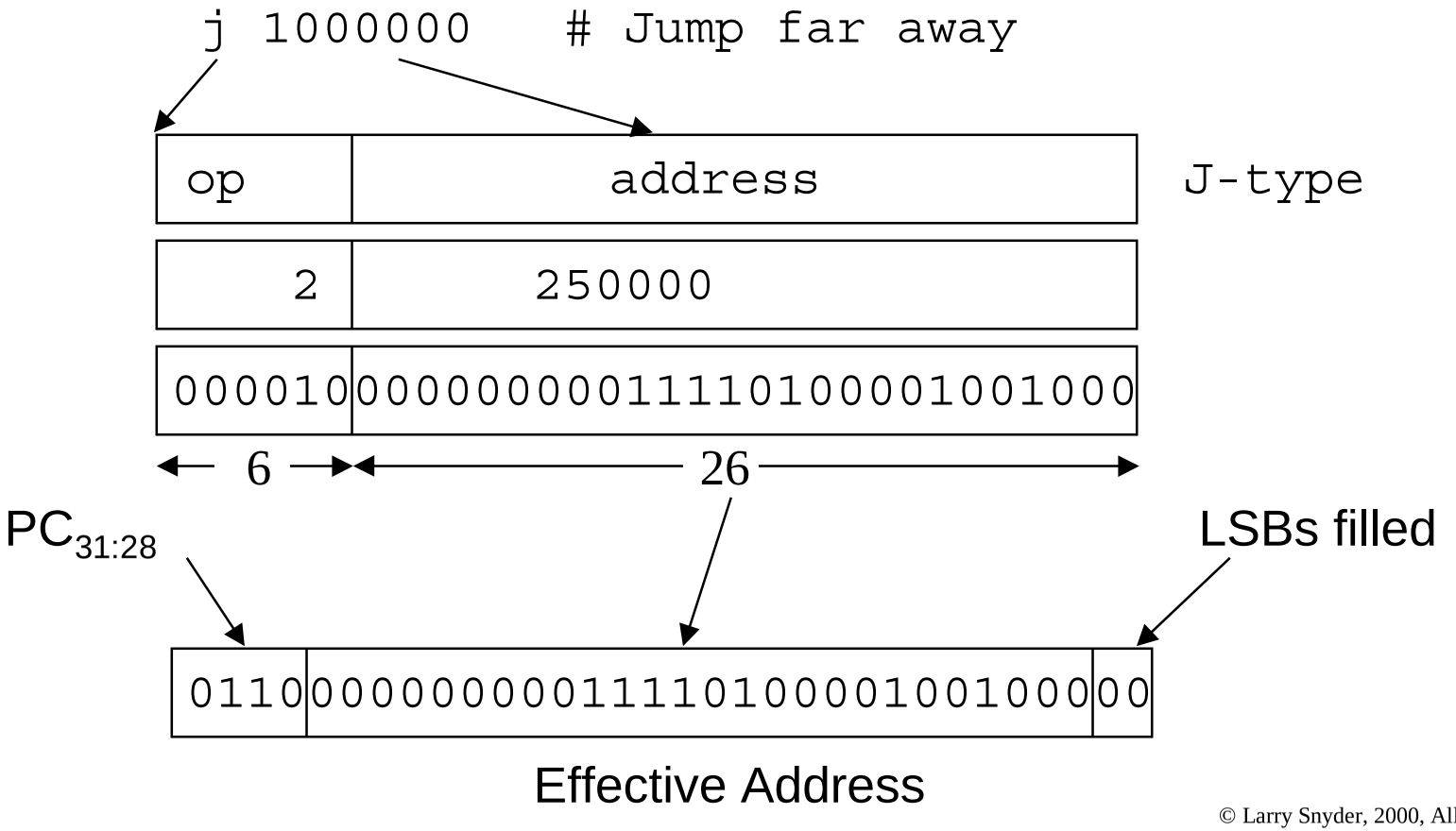
op	rs	rt	rd	shamt	funct
0	0	8	0	0	8
000000	00000	01000	00000	00000	001000

- **jr** is mostly used for procedure return

> The address in the register is a true memory address, not PC-relative instruction address

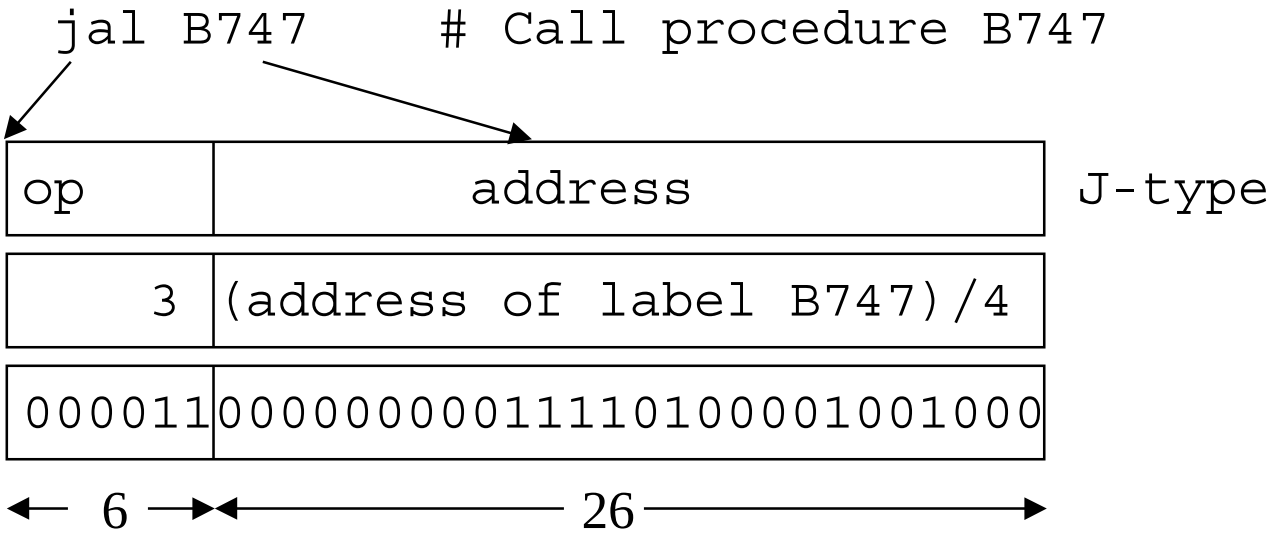
The Long Jump

- The jump instruction **j** is a J-format instruction



Jump and Link

- The jump and link **jal** instruction is J-type
- It jumps to the address like **j**, but saves the address of the next instruction in **$ra**



jal is used to call procedures, since saving the link allows the procedure to return to the point of the call

Register Usage

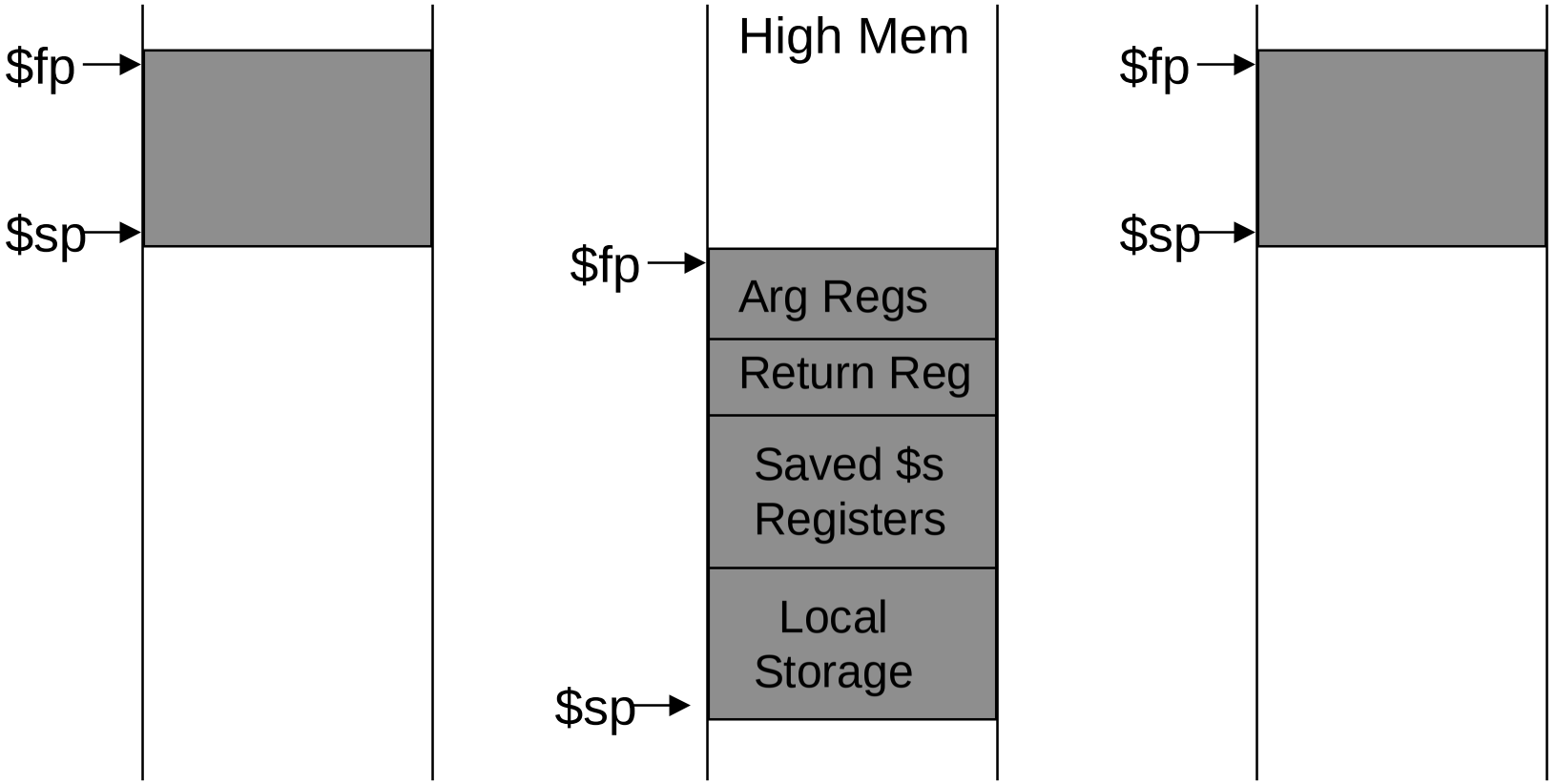
The assembler conventions on register usage

Name	Reg. No.	Usage	Preserved On Call
$zero	0	Constant value 0	N.A.
	1	Reserved for Assm	N.A.
$v0-$v1	2-3	Result registers	No
$a0-$a3	4-7	Arguments	Yes
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Locals	Yes
$t8-$t9	24-25	More temporaries	No
	26-27	Operating System	N.A.
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

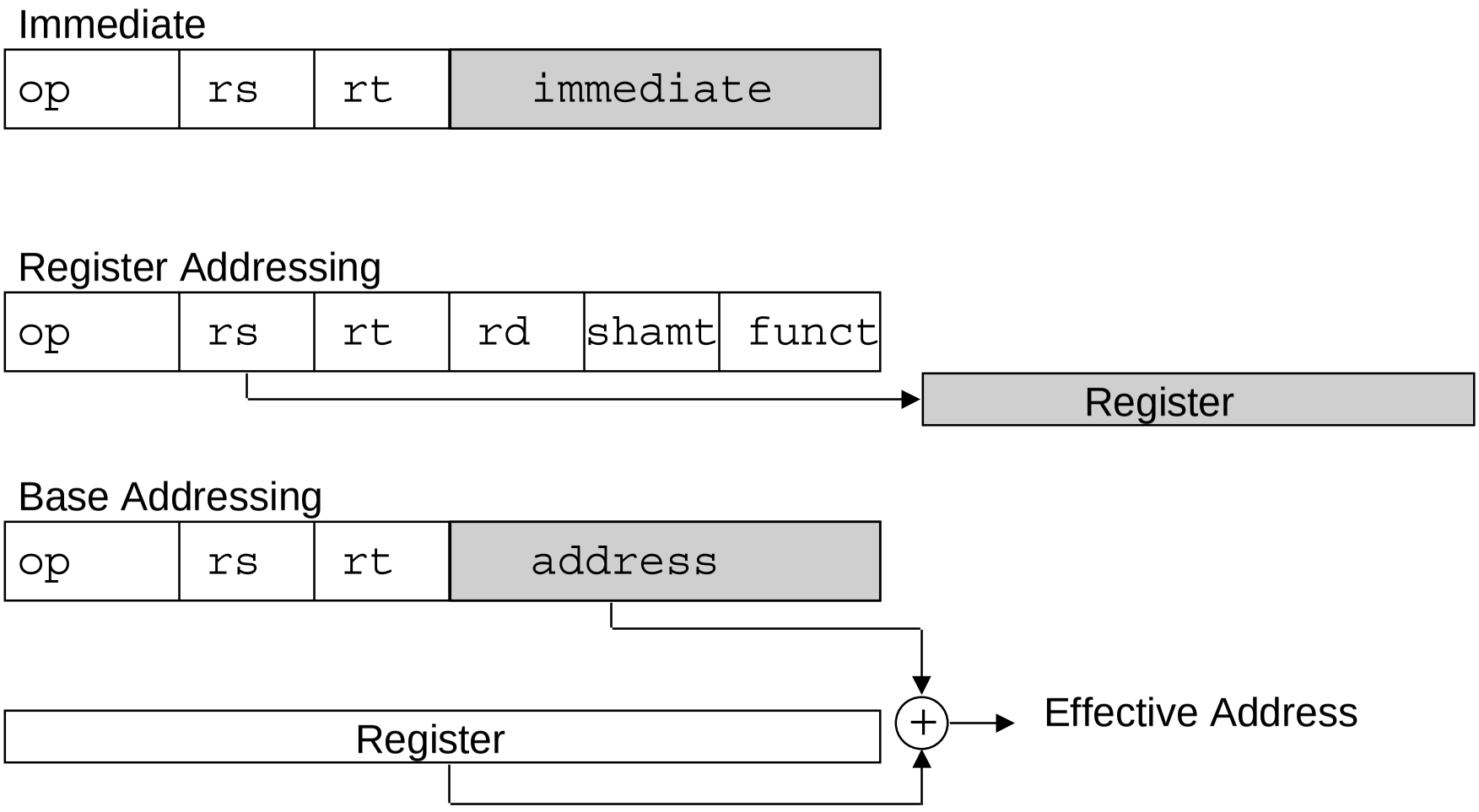
See Appendix A-6 for more information

The Stack Allocation

The stack provides the dynamic strorage for saving state during procedure calls

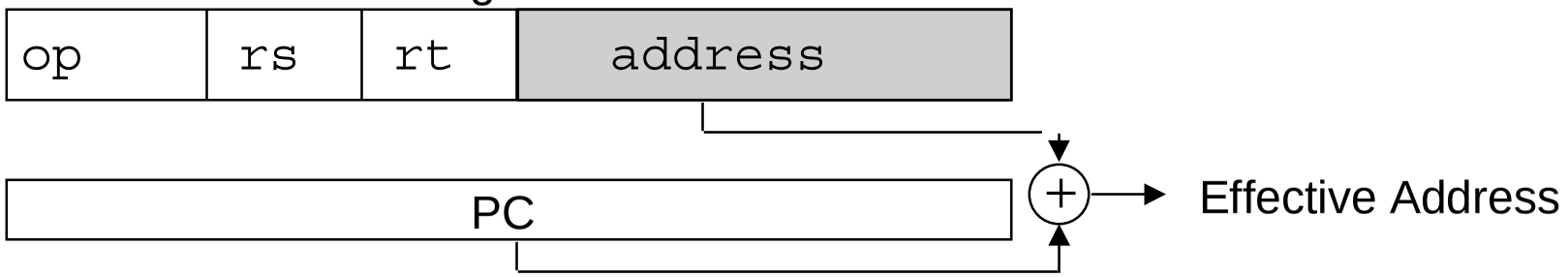


Addressing Modes



Addressing Modes (Continued)

PC-relative Addressing



Pseudo-direct Addressing