Computer System Architecture

H. SOUBRA

Disclaimer

This course contains copyrighted material the use of which has not always been specifically authorized by the copyright owner. They are used strictly for educational purposes. The principle of fair use applies.

Lecture 5: ISA- MIPS32

- Procedures in the MIPS32 ISA?

Basic block

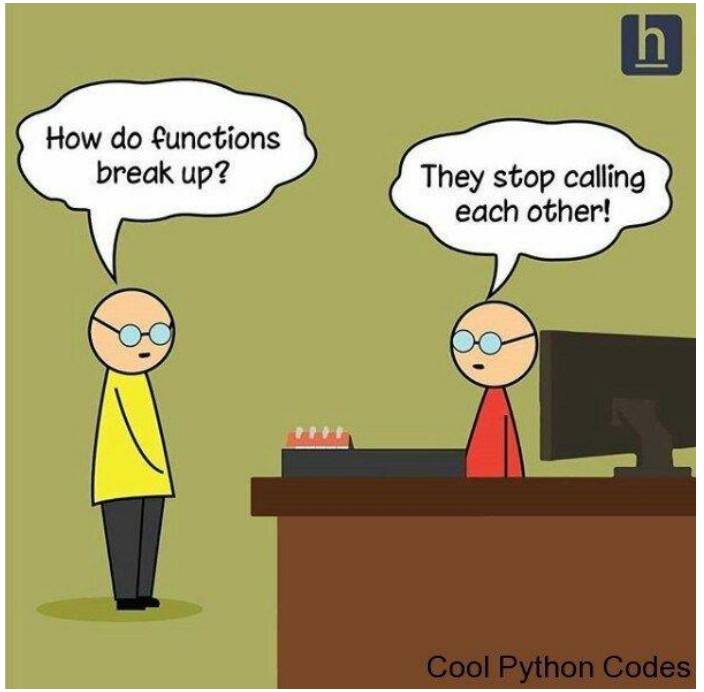
In the last lecture we talked about Conditions and loops.

- Basic block= a sequence of instructions without
  - branches, except possibly at the end,
  - branch targets or branch labels, except possibly at the beginning.

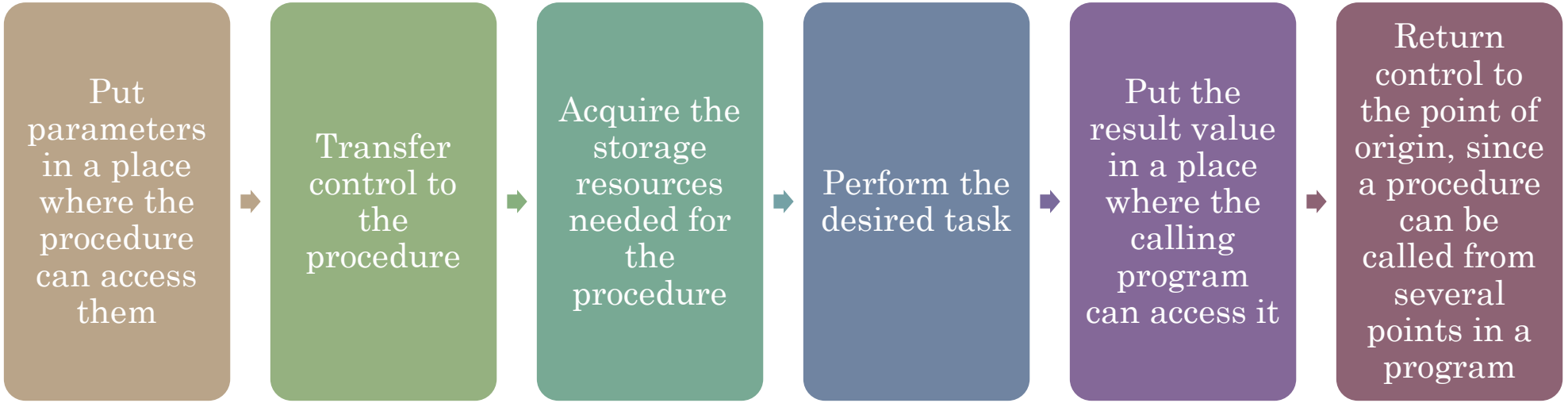
- One of the first early phases of compilation is breaking the program into basic blocks.

What's a procedure?

- A.K.A subroutine, subprogram, method or function…
- A block of code that performs a specific task.
- Generally speaking functions require:
  - arguments
  - return a value.



Procedures in Hardware



Some definitions

- **Caller:** The program that instigates a procedure and provides the necessary parameter values.
- **Callee:** A procedure that executes a series of stored instructions based on parameters provided by the caller and then returns control to the caller.
- **Stack** A data structure for spilling registers organized as a LIFO queue.
- **Push/Pop** Add/remove element to/from stack.
- **Stack Pointer** A value denoting the most recently allocated address in a stack that shows where registers should be spilled or where old register values can be found. Where can it be found in MIPS…?

Procedures in Hardware

Registers are the fastest place to hold data.. But which ones to use?

- $a0–$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

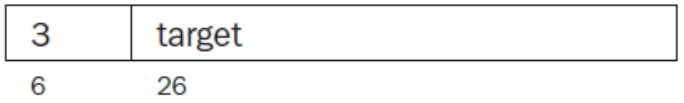
Where to store local variables in the procedure?

- $s0–$s7: saved registers that must be preserved on a procedure call (if used, the callee saves and restores them)

How to go to/get back from a procedure call?

Jump and link

`jal target`



Unconditionally jump to the instruction at target. Save the address of the next instruction in register `$ra`.

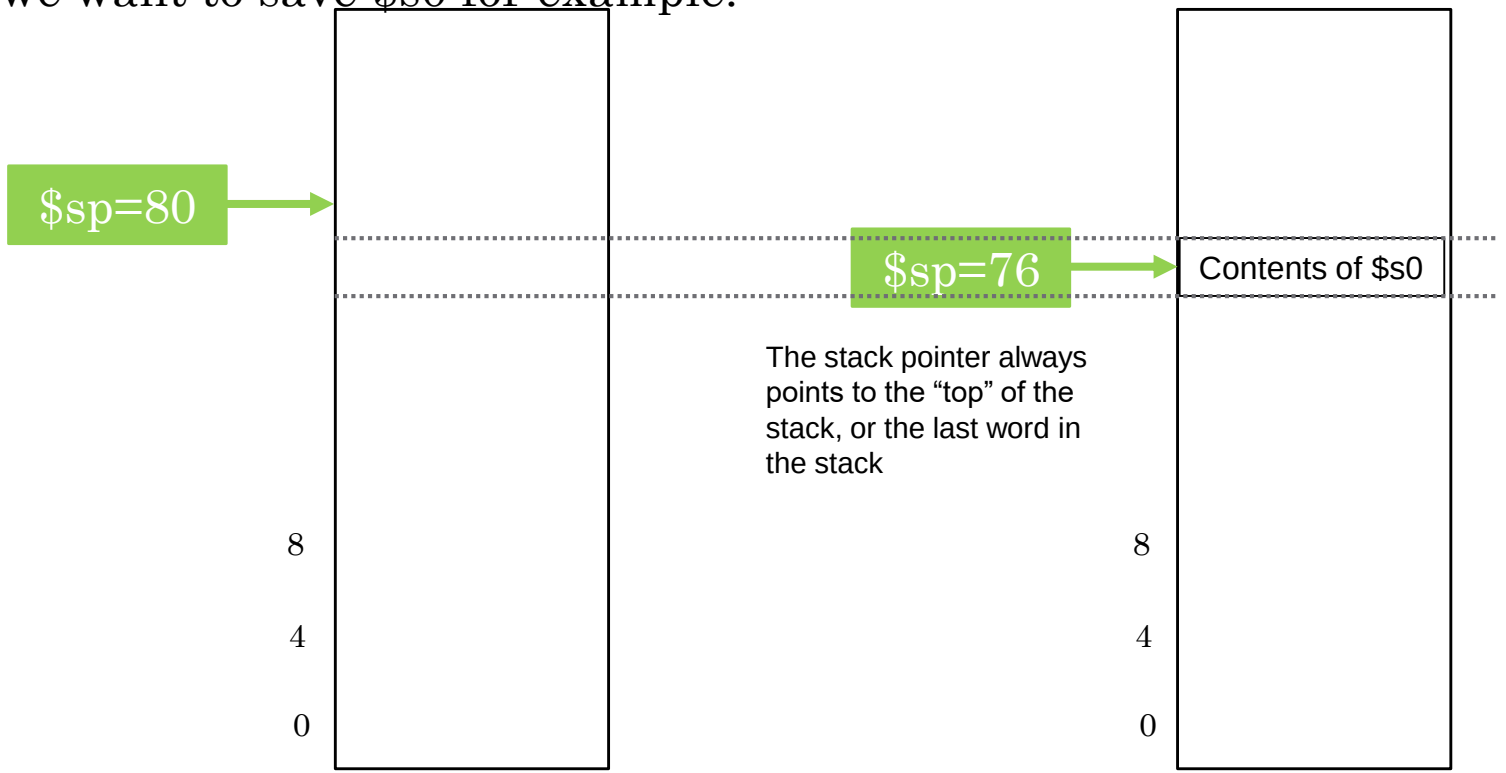
Jump register

`jr rs`

Unconditionally jump to the instruction whose address is in register `rs`.

Procedures in Hardware

- If $s0–$s7 are used they must be saved on the stack and restored at the end
- If we want to save $s0 for example:



And when we’re done?

MIPS Examples: Procedure

- In C:

```
int proc (int a, b) {
int c;
c = a+b; return c; }
```

- Ask yourself:

which registers to use? Do I have to save registers on the stack? How to branch back to the caller?

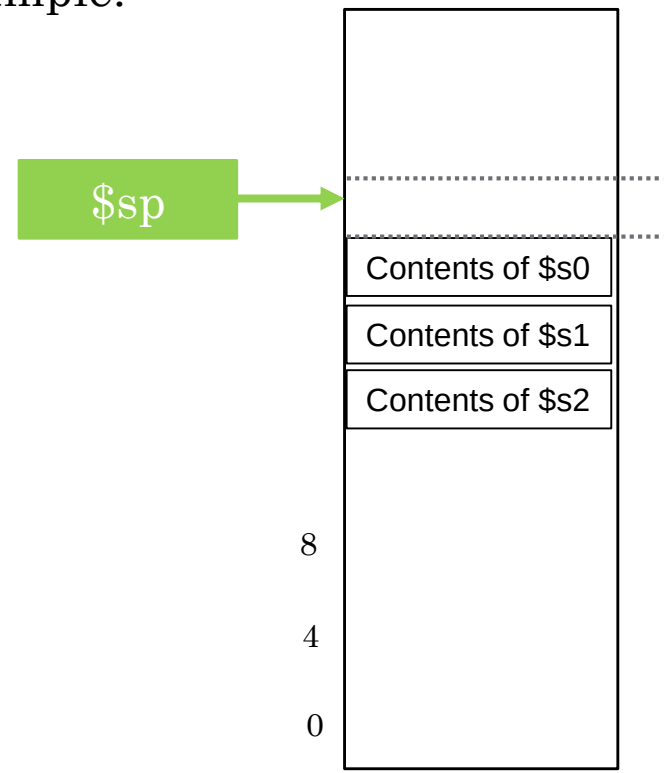
- In MIPS:

```
proc:   addi $sp, $sp, -4
        sw $s0, 0($sp)
        add $s0, $a0, $a1
        add $v0, $s0, $zero
        lw $s0, 0($sp)
        addi $sp, $sp, 4 #pop it out
        jr $ra
```

With what instruction did we call "proc?"

The Stack

- If $s0–$s7 are used they must be saved on the stack and restored at the end
- If we want to save $s0, $s1, $s2 for example:
- Which one will be popped out first?



The stack during

Non-Leaf Procedures

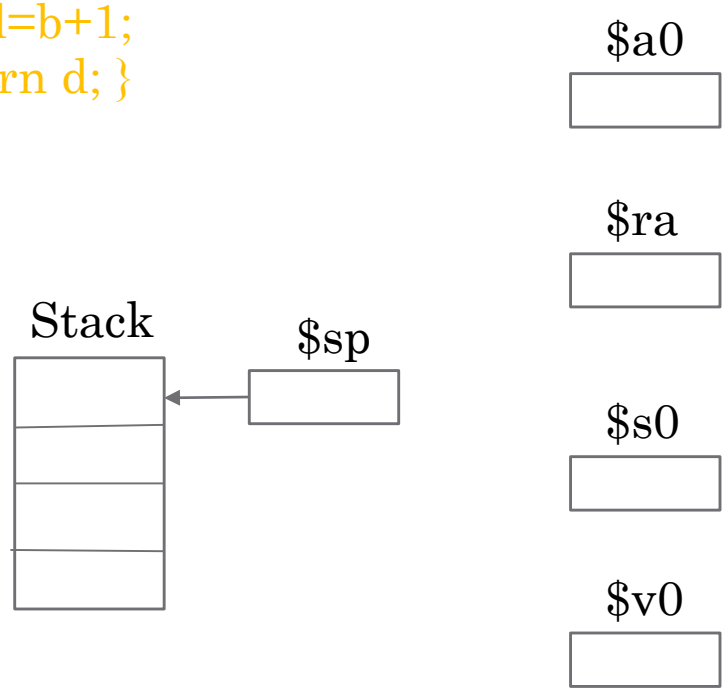
- Main prog calls Procedure A:
  - 2 into register $a0 and then using JAL A
- While executing Procedure A calls Procedure B!
  - 4 into register $a0 and then using JAL B
- Conflict!
  - $a0?
  - $ra?
- Solution?
  - The caller pushes any argument registers ($a0–$a3) or temporary registers ($t0–$t9) that are needed after the call. Return address register $ra as well.
  - The callee pushes any saved registers ($s0–$s7) used by the callee.
  - When returned, the registers are restored and SP adjusted

MIPS Examples: Non-Leaf Procedure

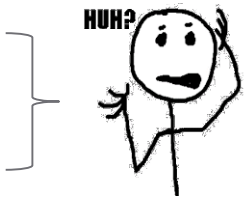
- In C:

```
int proc1 (int a) {
proc2 (a+1);
int c=a-1;
return c; }


int proc2 (int b) {
int d=b+1;
return d; }
```



- In MIPS:




```
        addi $a0, $0, 4
        jal proc1
        j exit
proc1:  addi $sp, $sp, -8
        sw $ra, 4($sp)
        sw $a0, 0($sp)
        addi $a0, $a0, 1
        jal proc2
        lw $a0, 0($sp)
        addi $s0, $a0, -1
        add $v0, $s0, $0
        lw $ra, 4($sp)
        addi $sp, $sp, 8 #pop it out
        jr $ra

proc2:  addi $sp, $sp, -4
        sw $s0, 0($sp)
        addi $s0, $a0, 1
        add $v0, $s0, $0
        lw $s0, 0($sp)
        addi $sp, $sp, 4
        jr $ra
exit:
```

Research

- What are the different *pseudoinstructions* of MIPS?