Arithmetic/Logic instructions

“easy”: two operands or one operand and a constant (immediate)

like `sll`, `add`, `xor`, `slt`, etc.

because immediate are limited in terms of space, we can use a register with `lui` (12 bits), then `addiu` (add unsigned) and `xor` to copy

Assembler directives

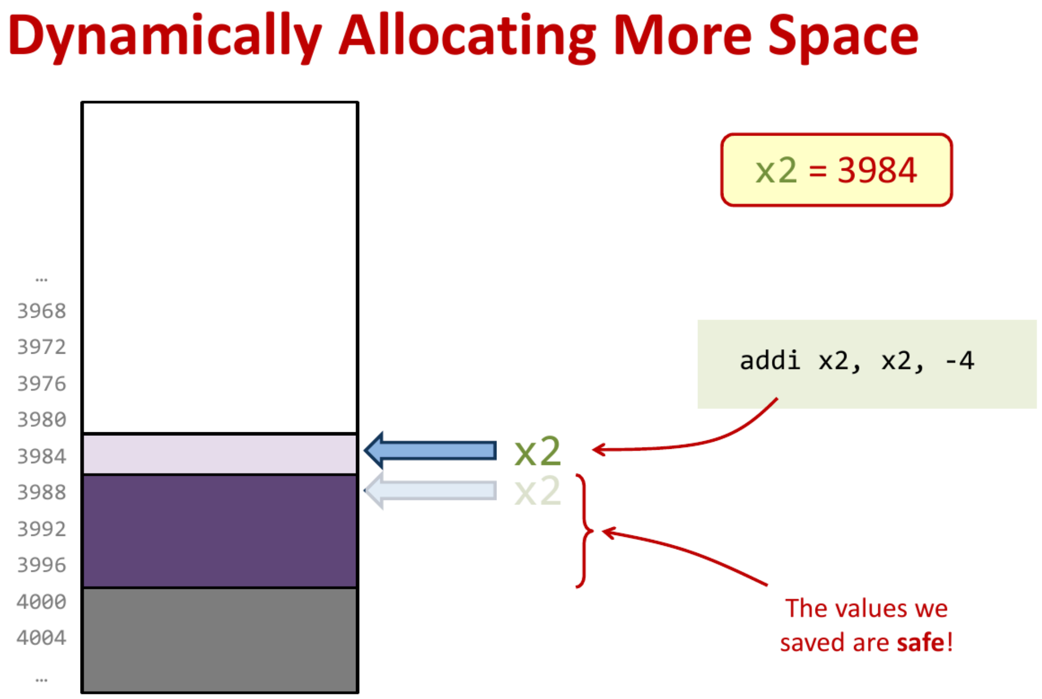
Like `.text`, `.data`, `.asciiz`..

Functions

`jal x1, sqrt` –> leaves `PC+4` into the `x1` register so the funtion can callback `x1` with a simple `jr x1` when finishing.

With RISCV we can simply use `jal offset` and `ret` instead of specifying the register `x1`.

Stack Pointer

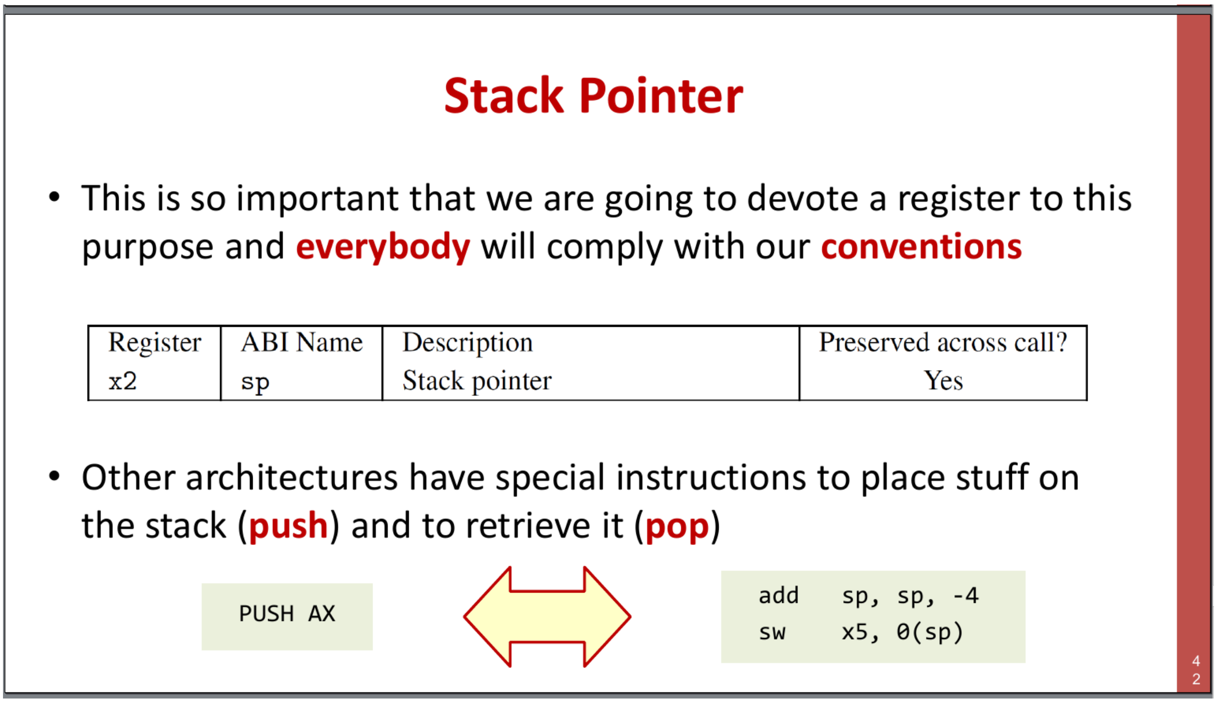


Dynamically Allocating More Space

x2 = 3984

```
addi x2, x2, -4
```

The values we saved are **safe**!



Stack Pointer

- This is so important that we are going to devote a register to this purpose and **everybody** will comply with our **conventions**

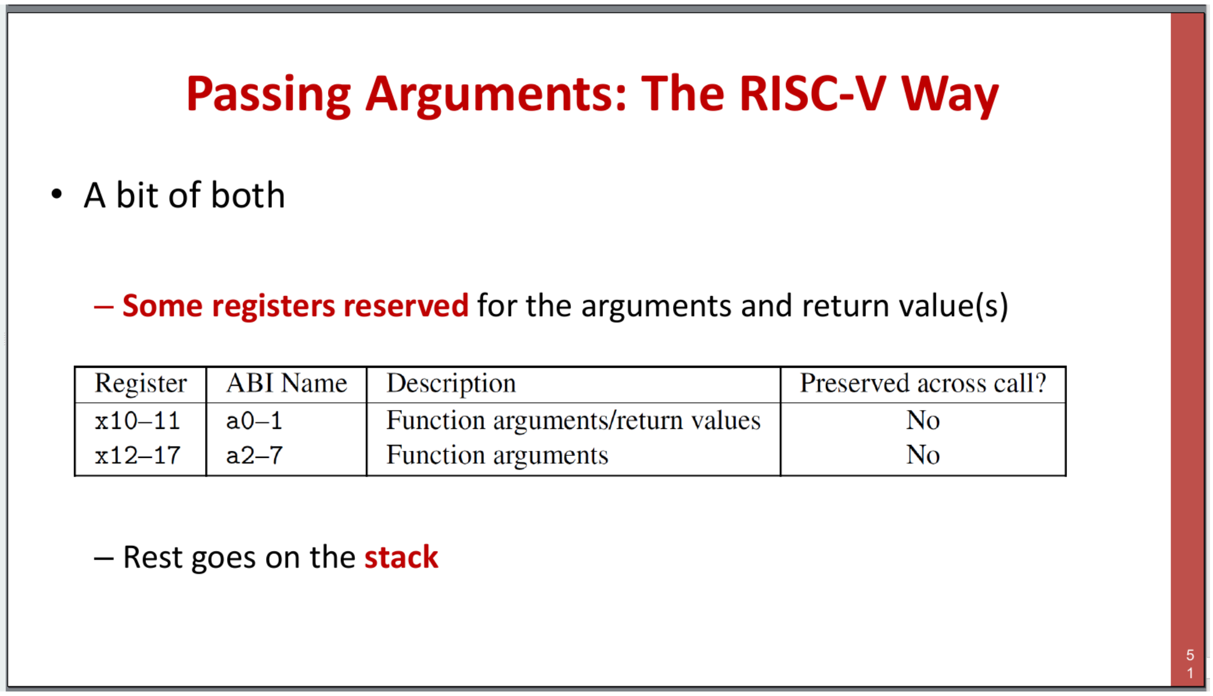
Register	ABI Name	Description	Preserved across call?
x2	sp	Stack pointer	Yes

- Other architectures have special instructions to place stuff on the stack (**push**) and to retrieve it (**pop**)

```
PUSH AX
```

```
add   sp, sp, -4
sw    x5, 0(sp)
```

Passing Arguments



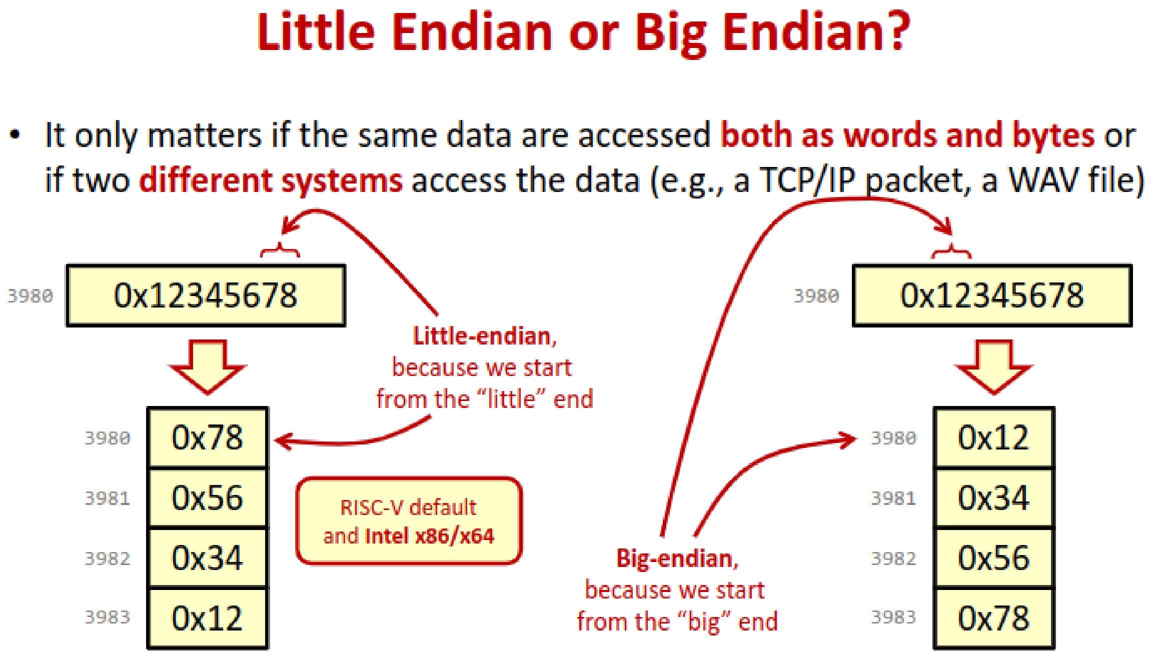
Passing Arguments: The RISC-V Way

- A bit of both
  - **Some registers reserved** for the arguments and return value(s)

Register	ABI Name	Description	Preserved across call?
x10–11	a0–1	Function arguments/return values	No
x12–17	a2–7	Function arguments	No

  - Rest goes on the **stack**

Little/Big Endian



Little Endian or Big Endian?

- It only matters if the same data are accessed **both as words and bytes** or if two **different systems** access the data (e.g., a TCP/IP packet, a WAV file)

3980
0x12345678
3980
0x78
3981
0x56
3982
0x34
3983
0x12
Little-endian, because we start from the “little” end
RISC-V default and Intel x86/x64
Big-endian, because we start from the “big” end
3980
0x12345678
3980
0x12
3981
0x34
3982
0x56
3983
0x78

Little endian is used by RISCV and is cool because when using: if `t0 = 1`:

```
sw t0, 0(t1)
```

writes the same in memory as:

```
sb t0, 0(t1)
```

!