MACHINE LEVEL PROGRAMMING 2: CONTROL

CS 045

Computer Organization and Architecture

Prof. Donald J. Patterson

Adapted from Bryant and O'Hallaron,
Computer Systems:
A Programmer's Perspective, Third Edition

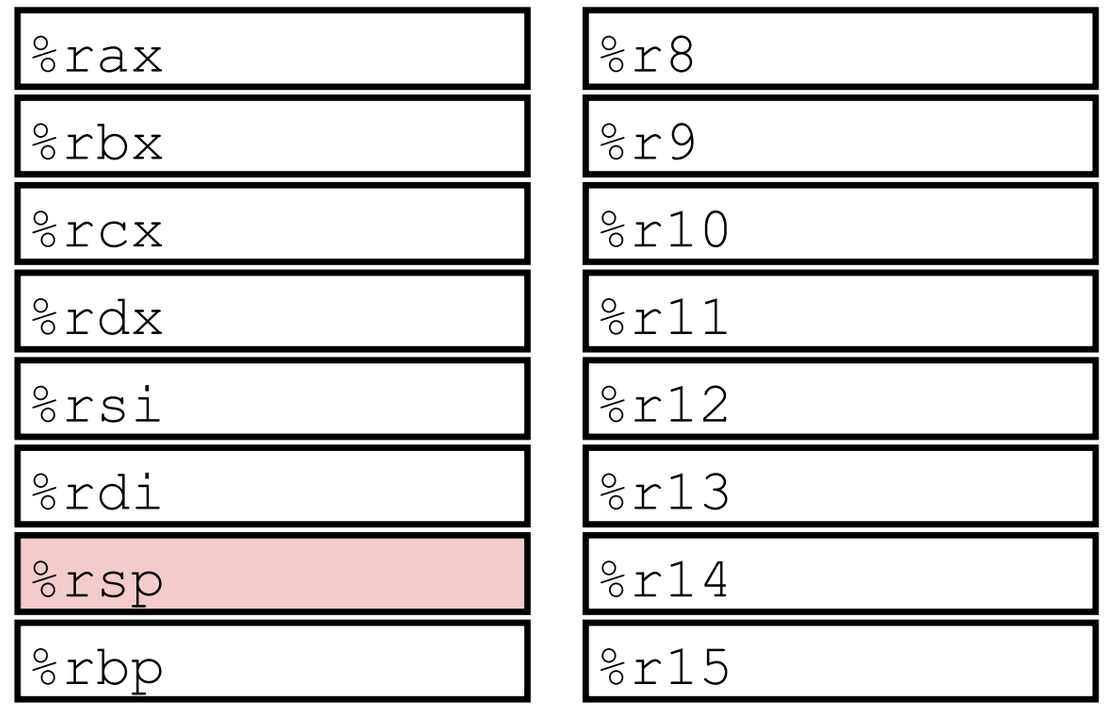
- CONTROL: CONDITION CODES
- CONDITIONAL BRANCHES
- LOOPS
- SWITCH STATEMENTS

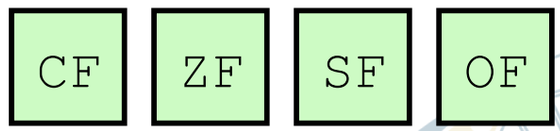
- CONTROL: CONDITION CODES
- CONDITIONAL BRANCHES
- LOOPS
- SWITCH STATEMENTS

CONTROL

PROCESSOR STATE

- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)

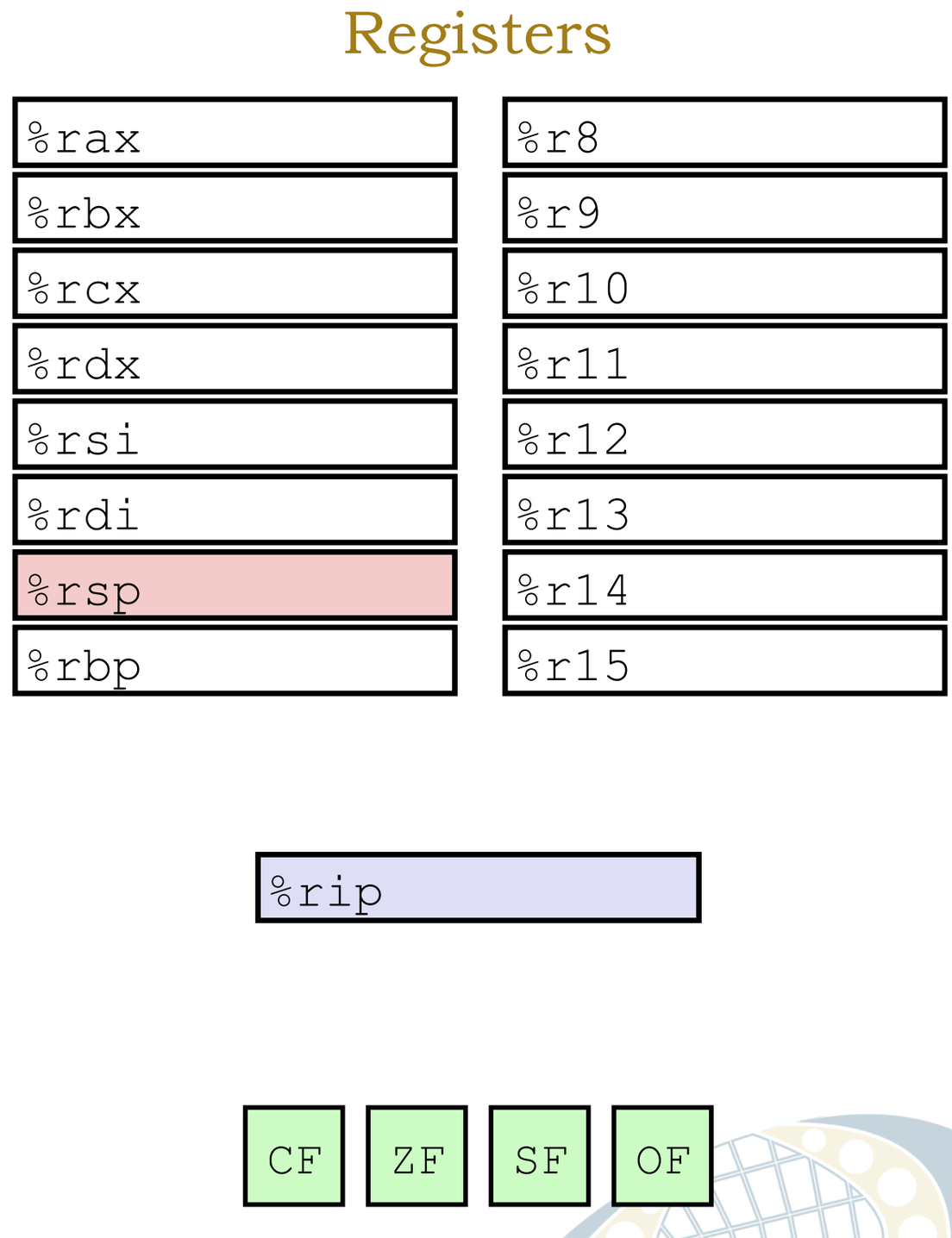




CONTROL

PROCESSOR STATE

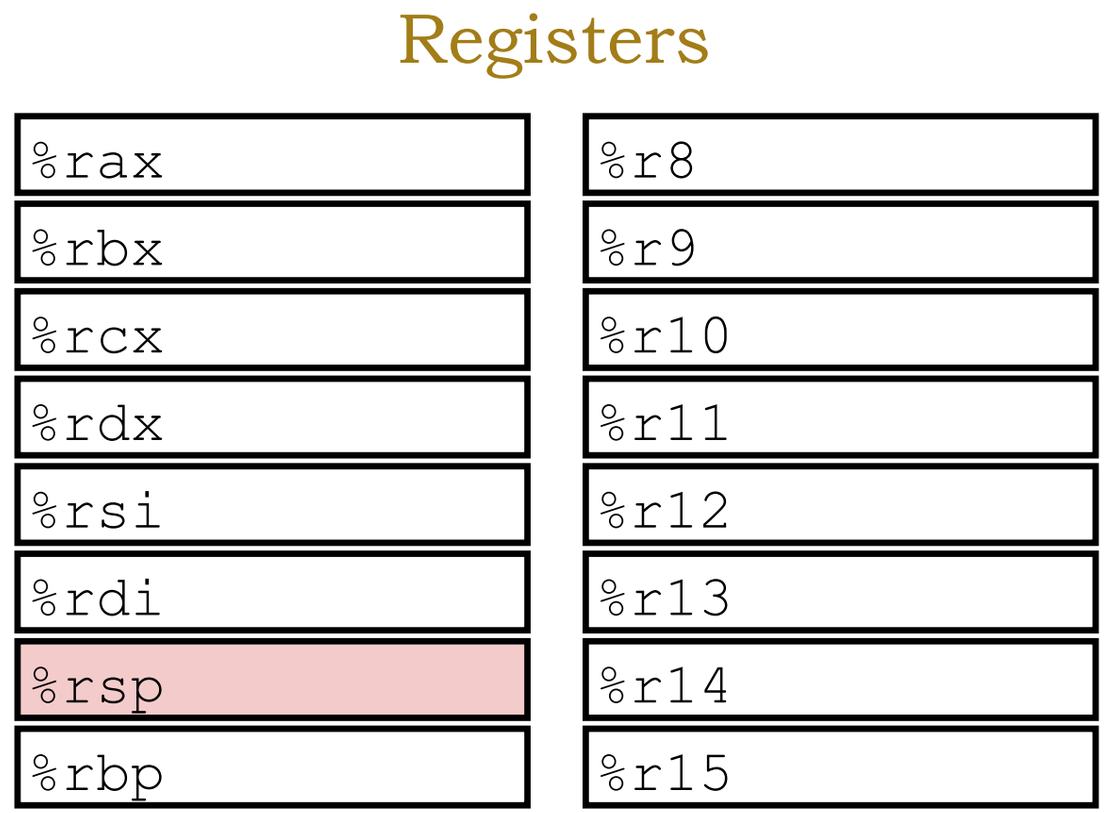
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

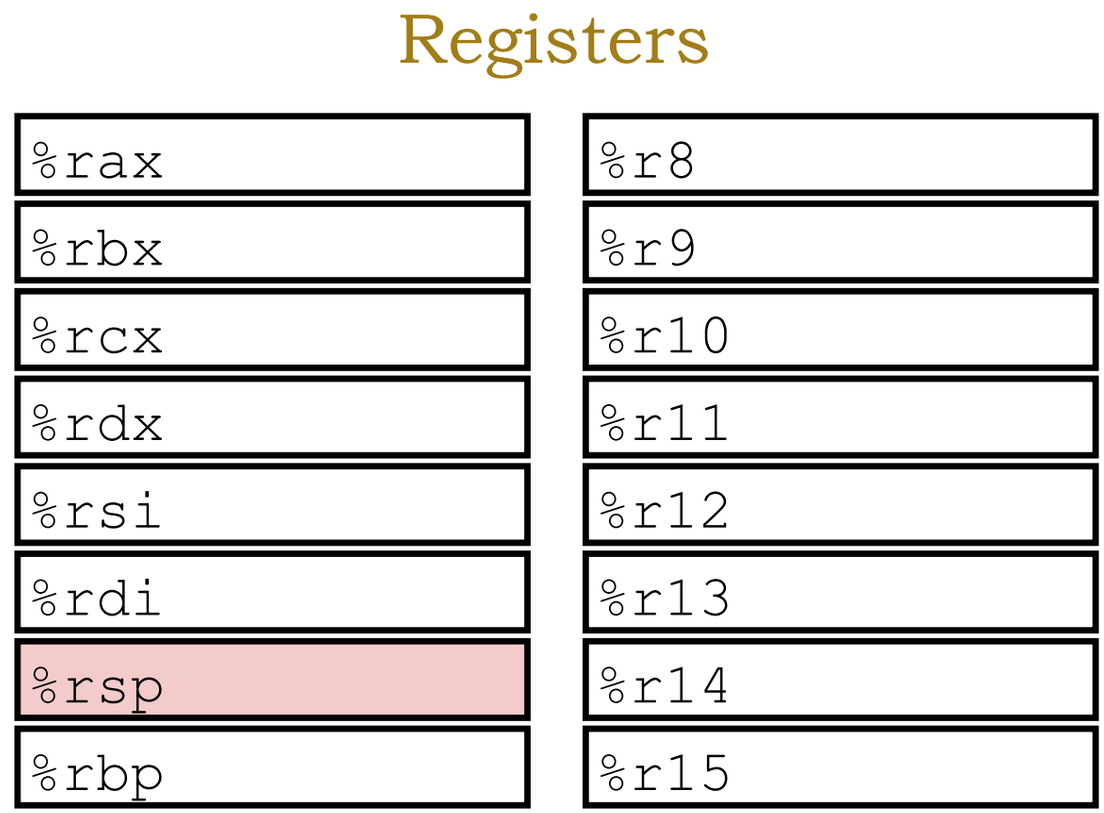
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

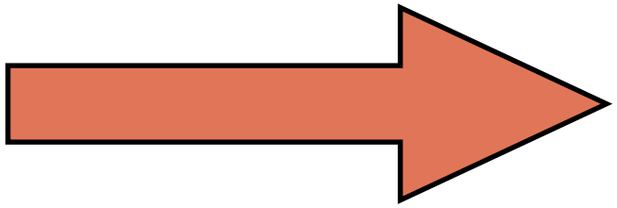
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



Registers

%rax	%r8
%rbx	%r9
%rcx	%r10
%rdx	%r11
%rsi	%r12
%rdi	%r13
%rsp	%r14
%rbp	%r15

Instruction Pointer

%rip

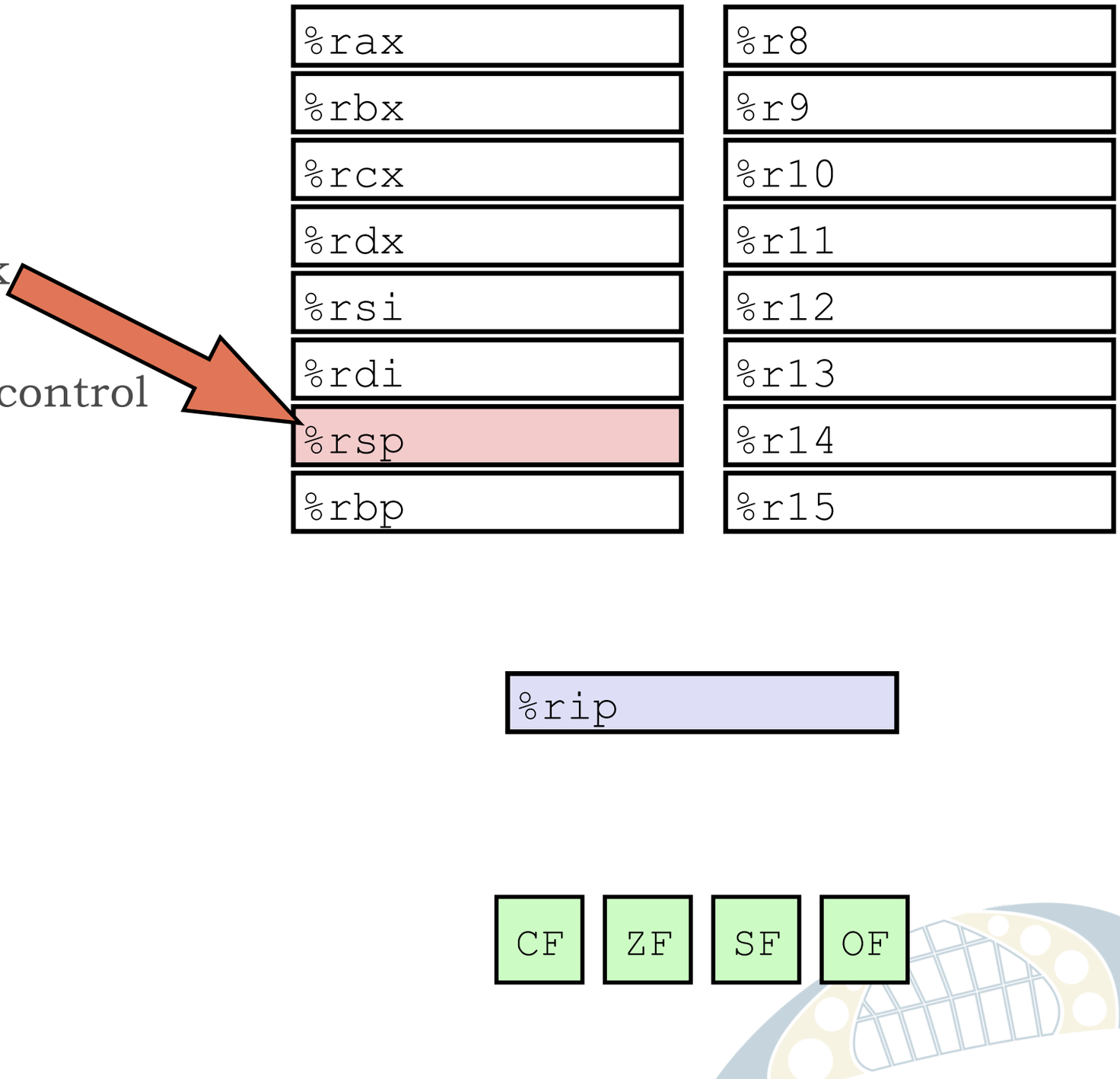
Condition Codes

CF ZF SF OF

CONTROL

PROCESSOR STATE

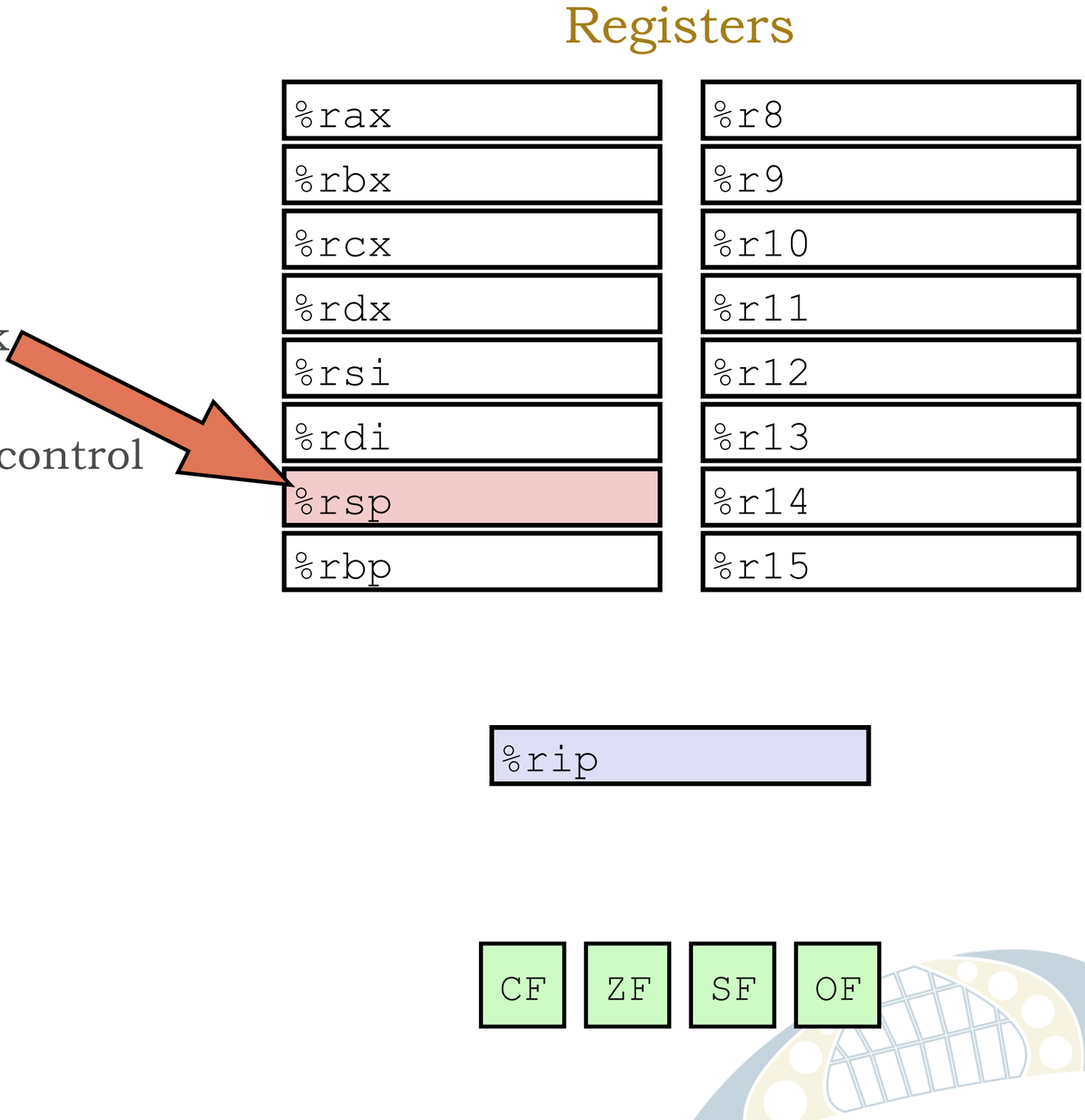
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
  - Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

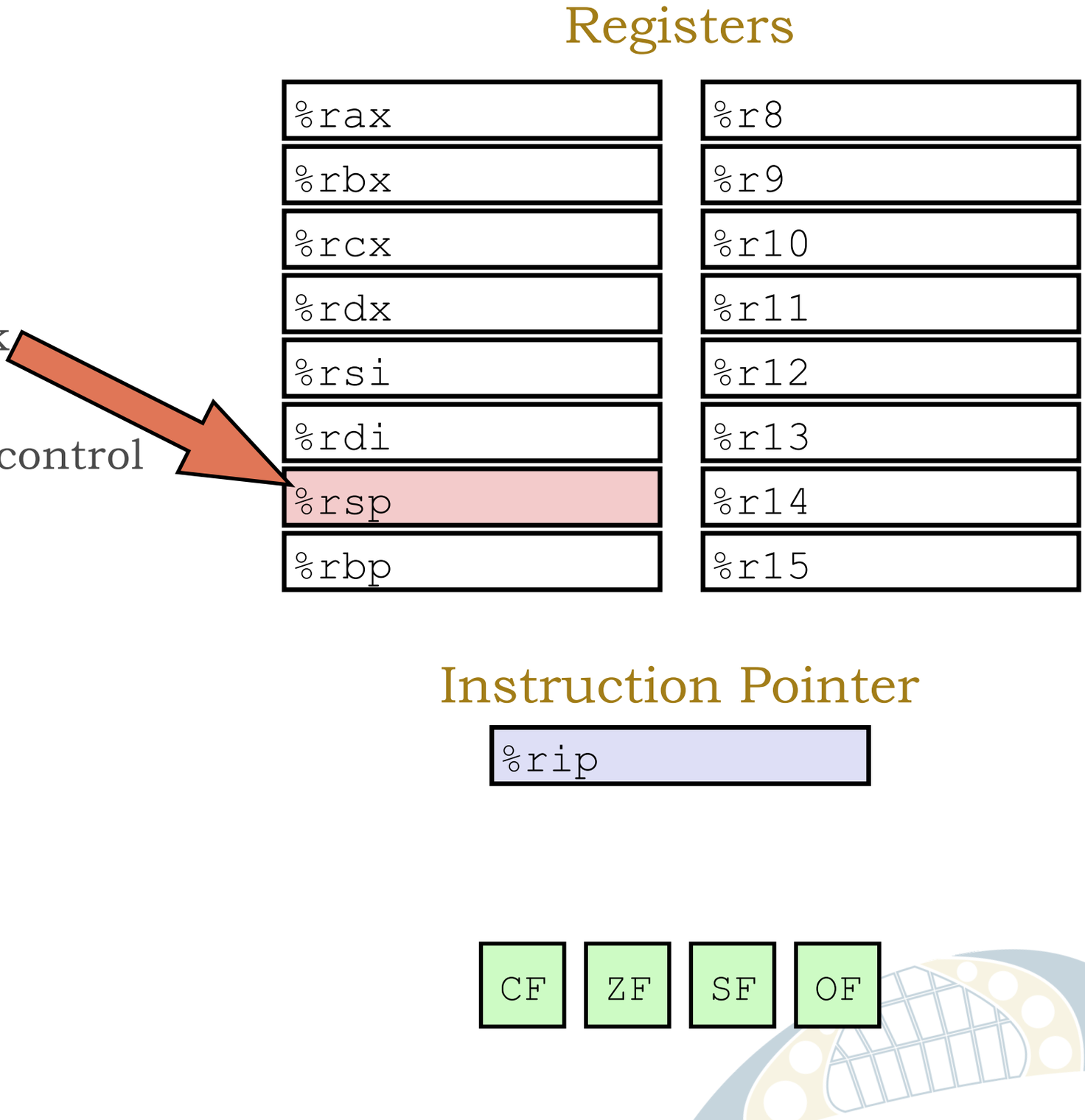
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
  - Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

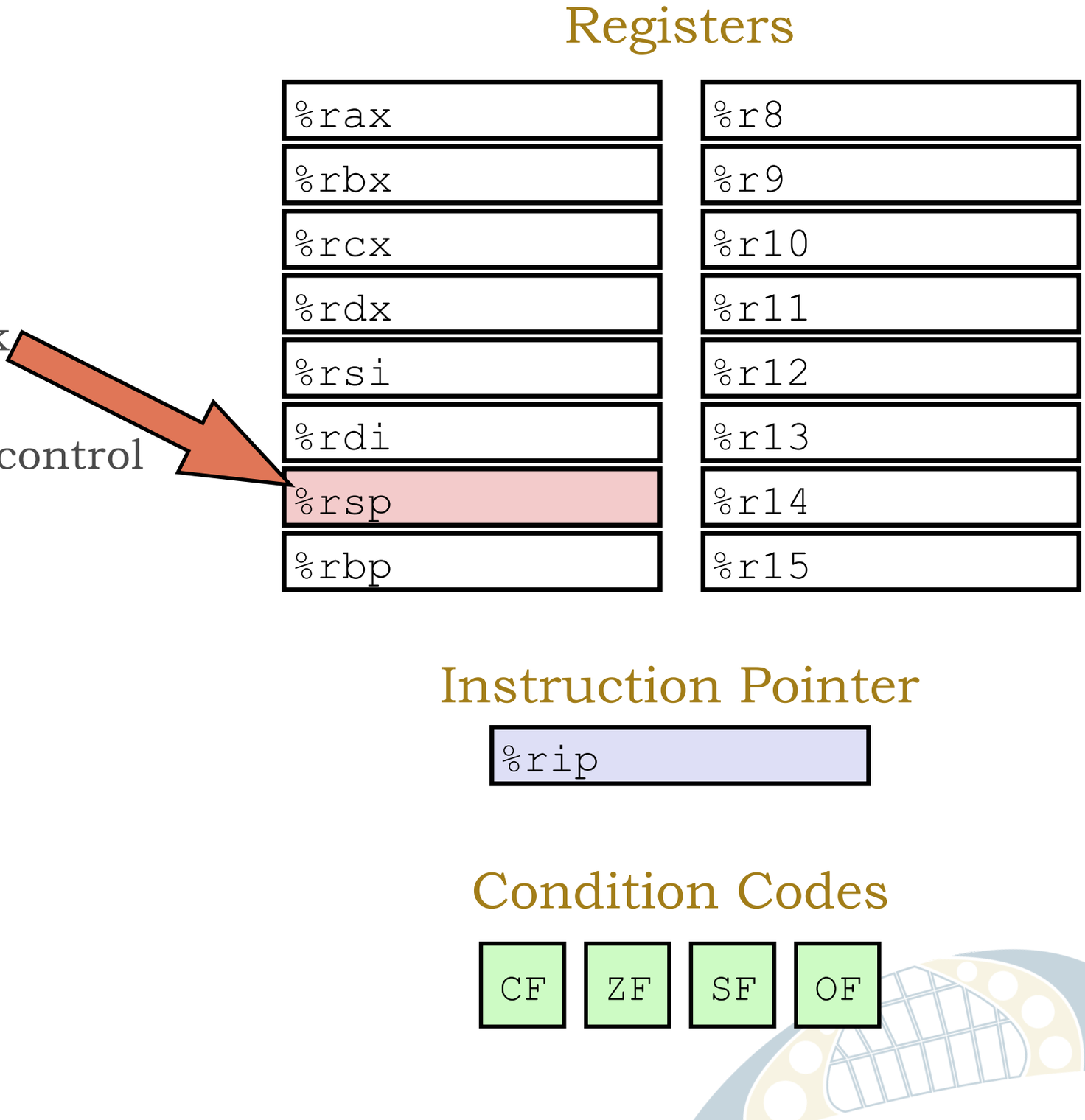
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
  - Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

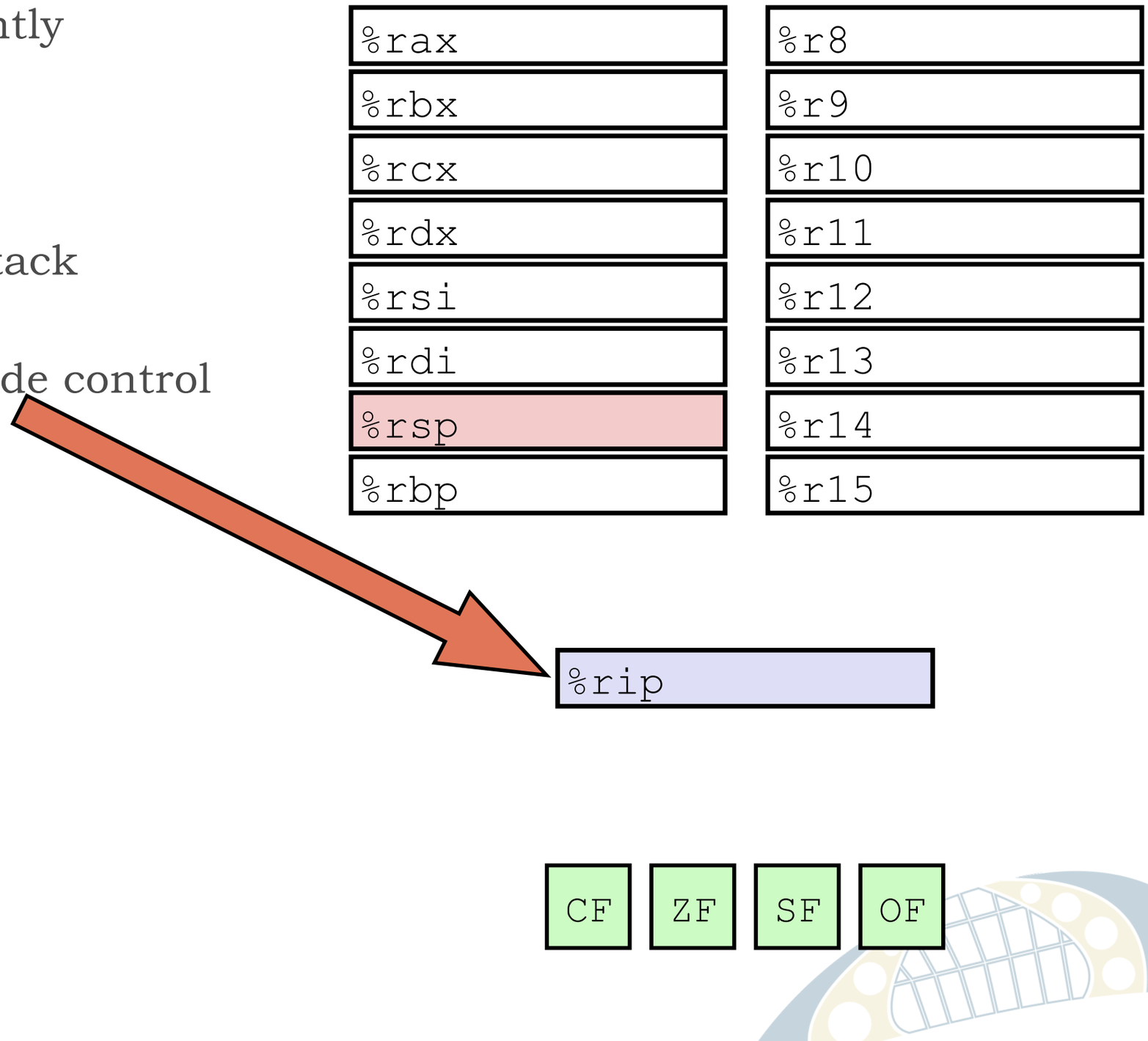
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
  - Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

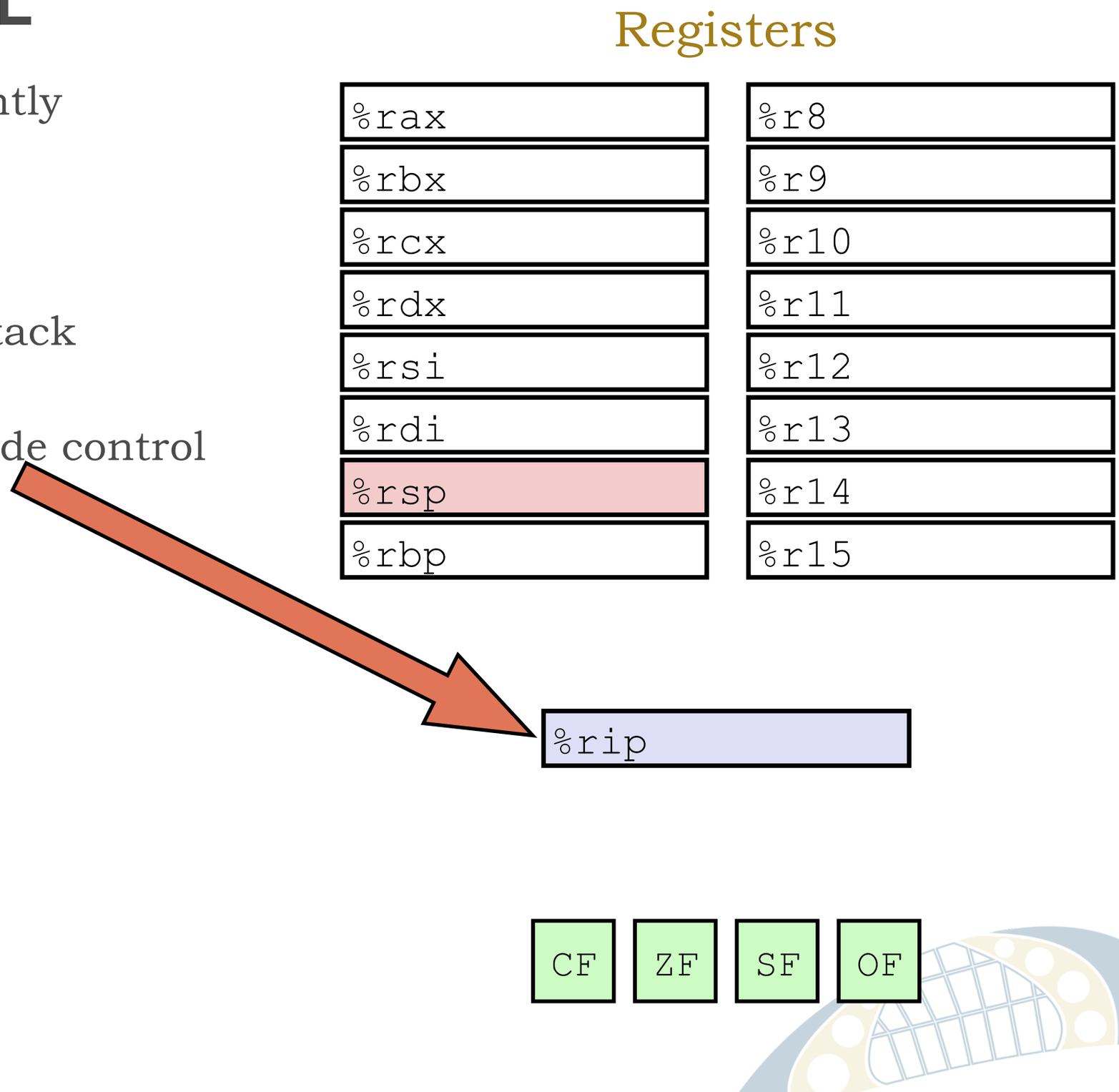
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

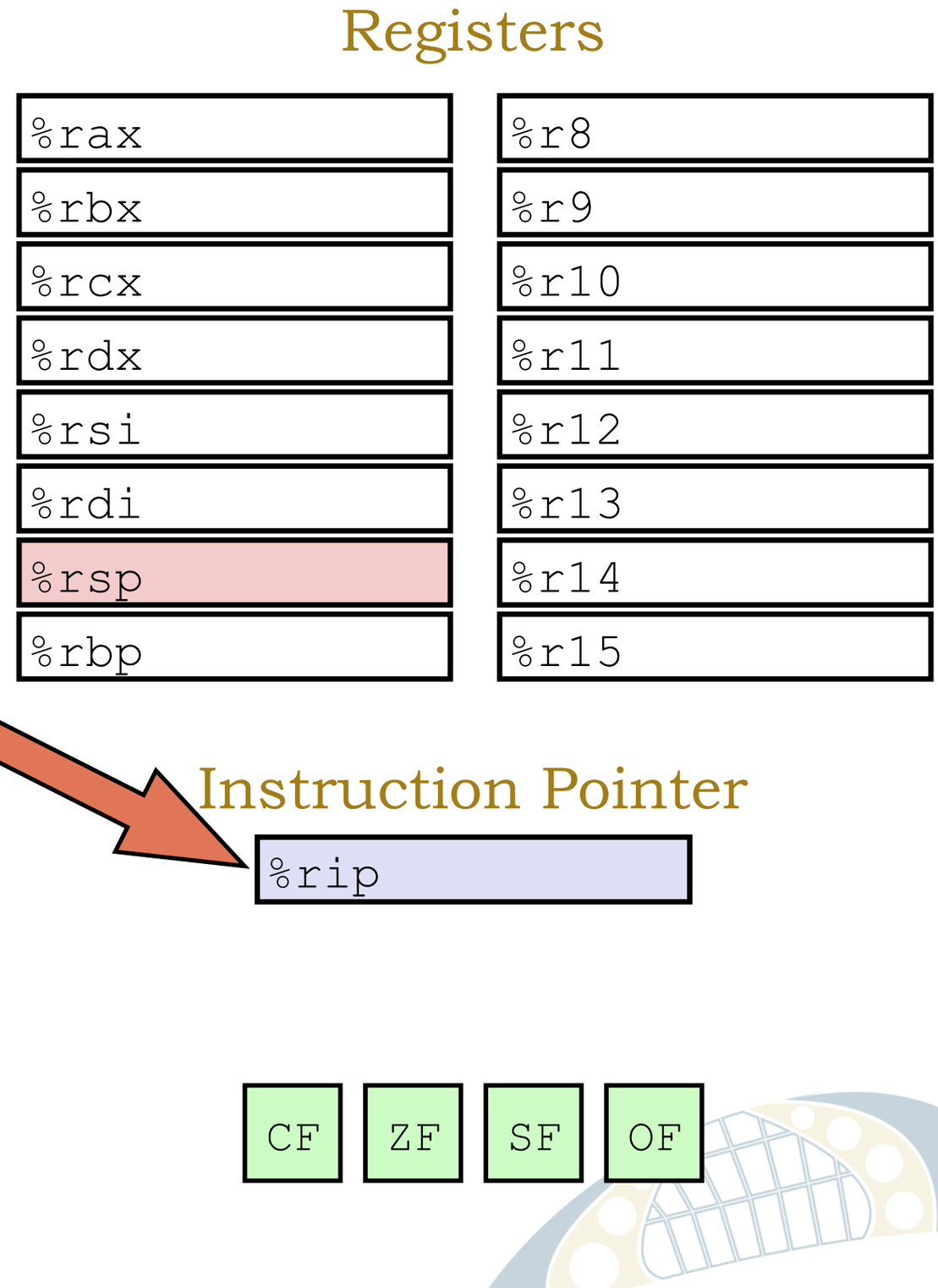
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

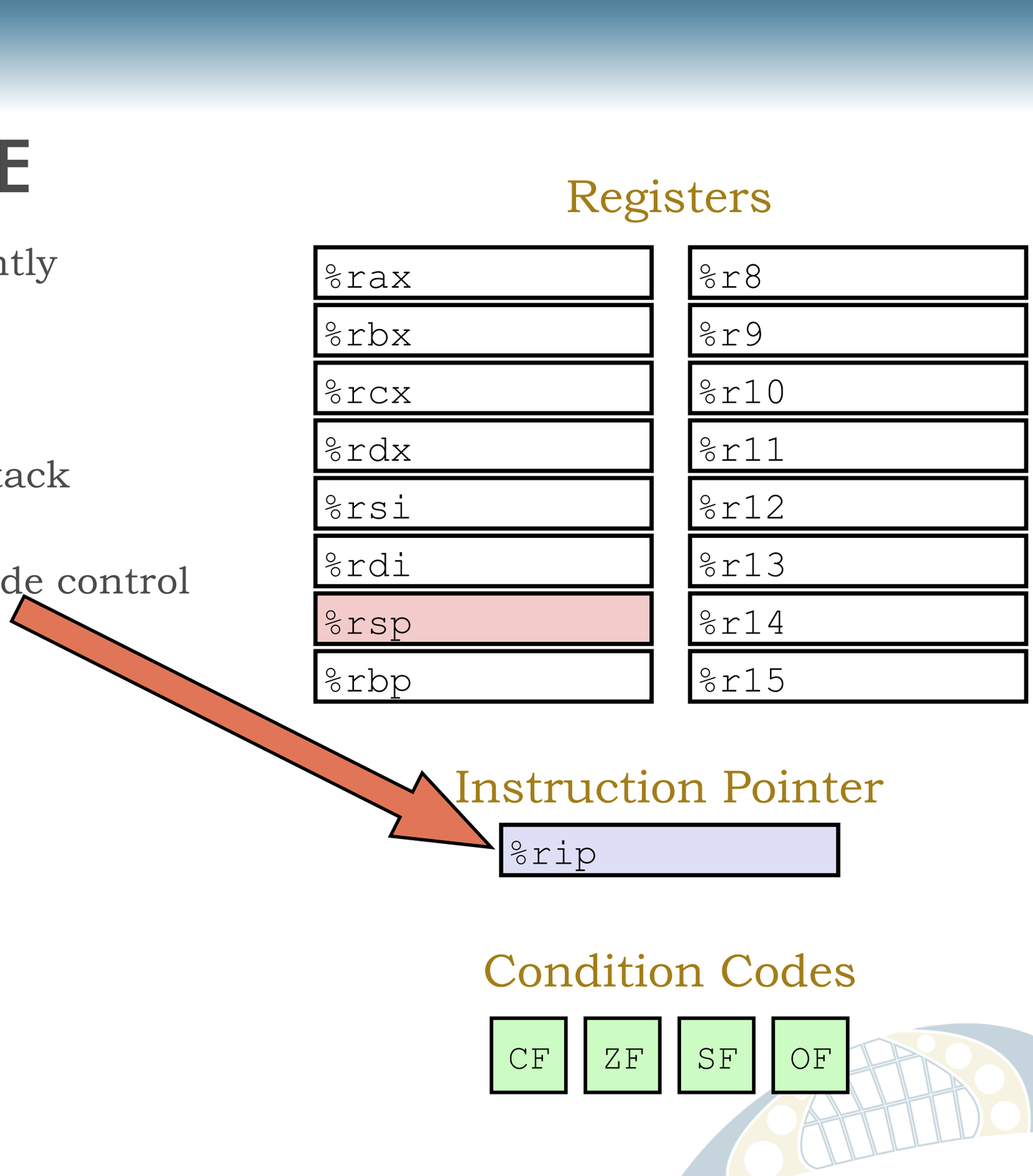
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)



CONTROL

PROCESSOR STATE

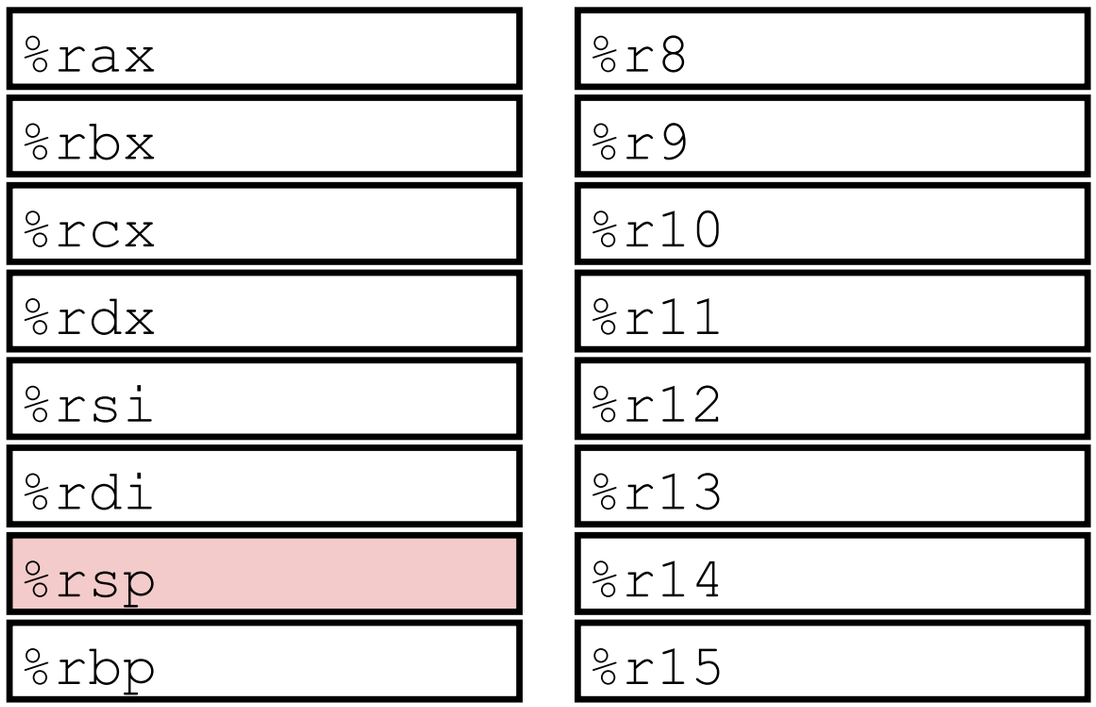
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)

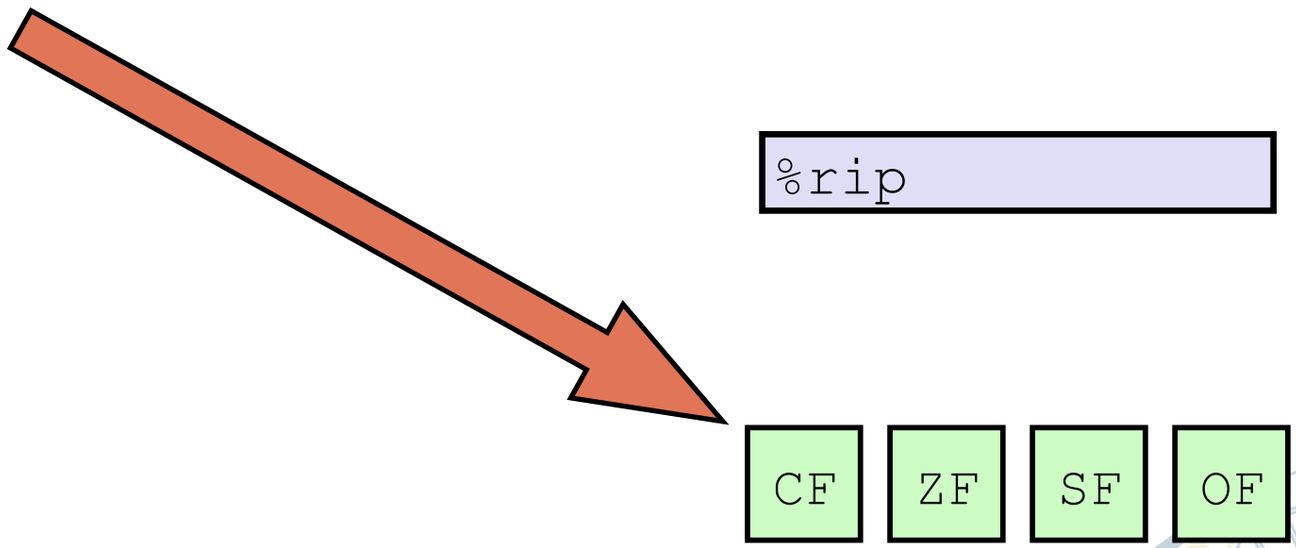


CONTROL

PROCESSOR STATE

- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)

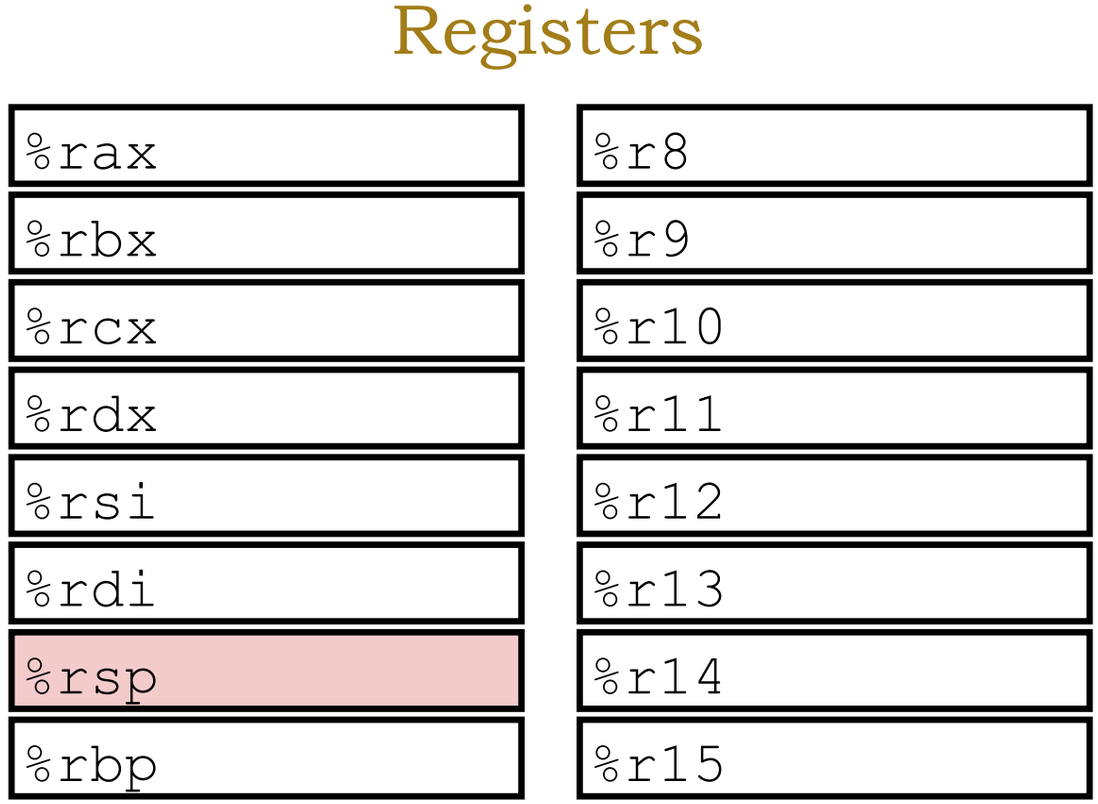


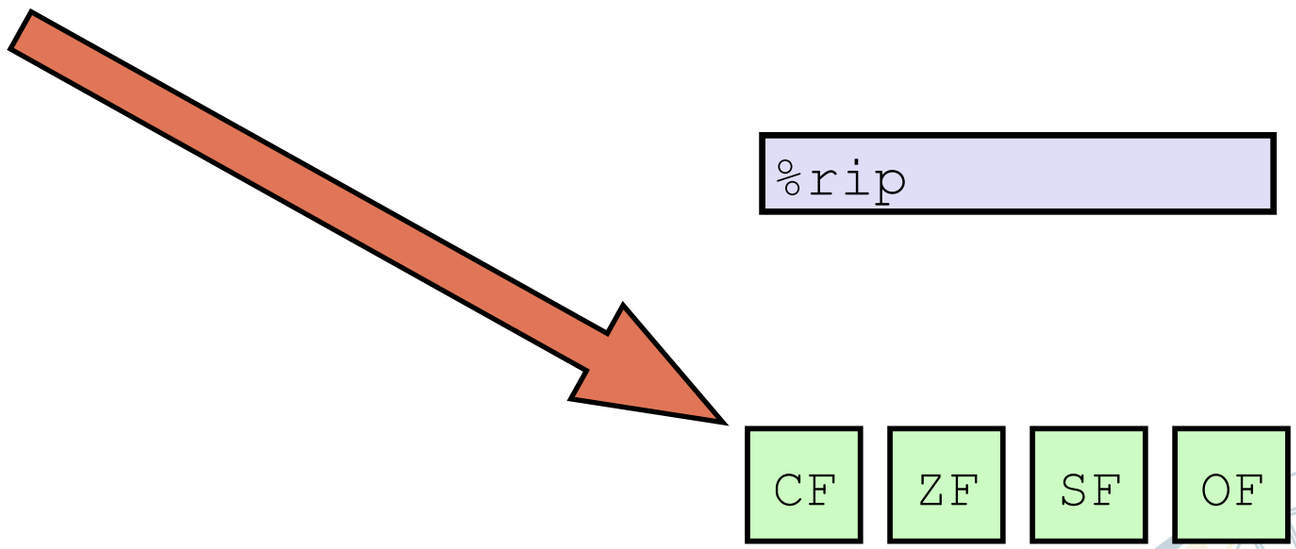


CONTROL

PROCESSOR STATE

- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)

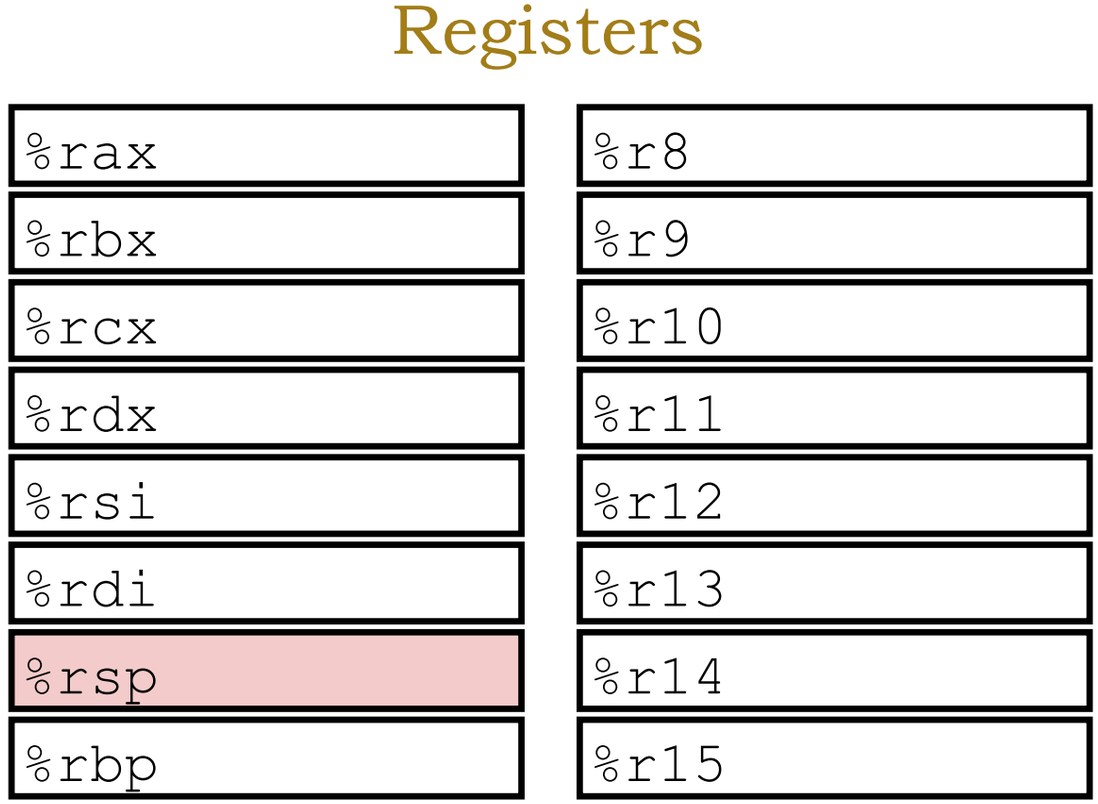


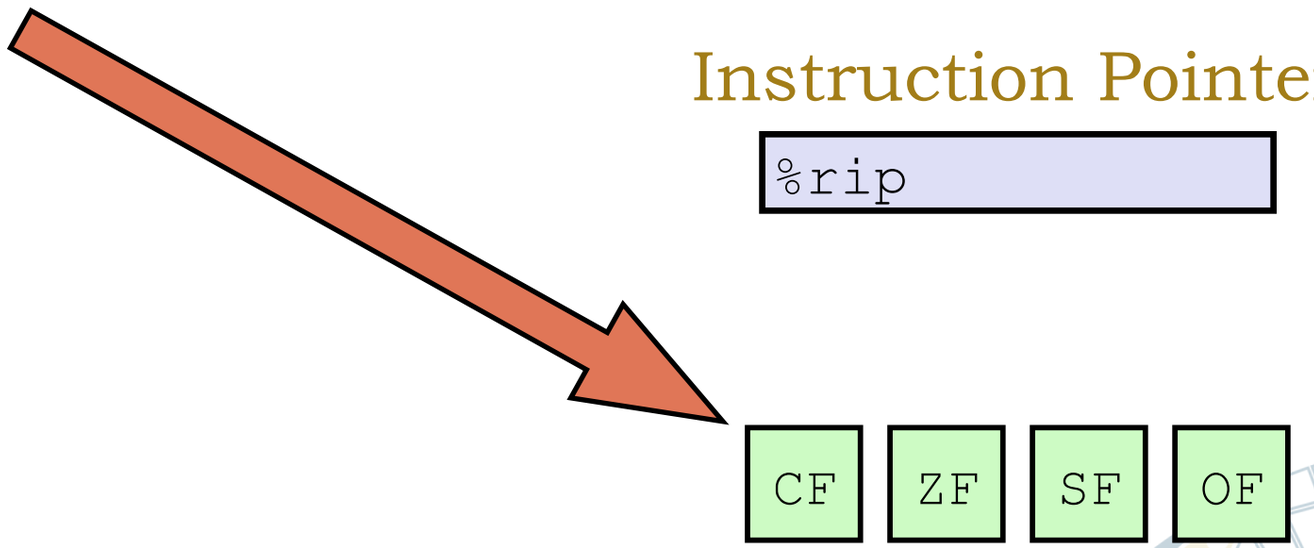


CONTROL

PROCESSOR STATE

- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)

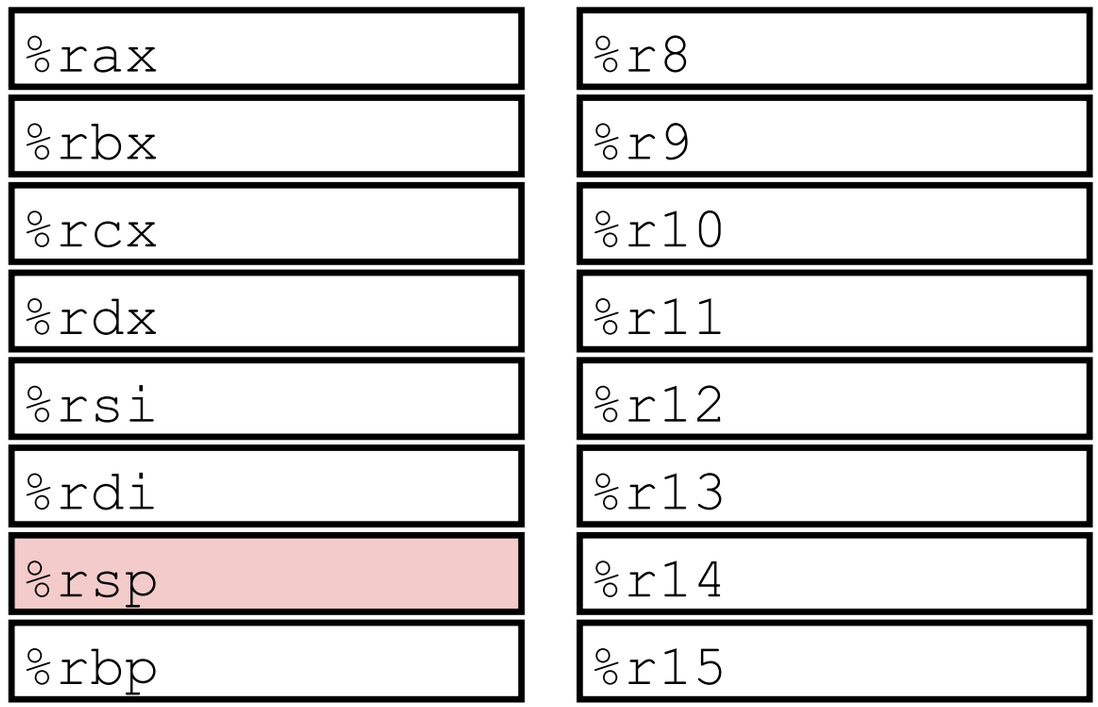


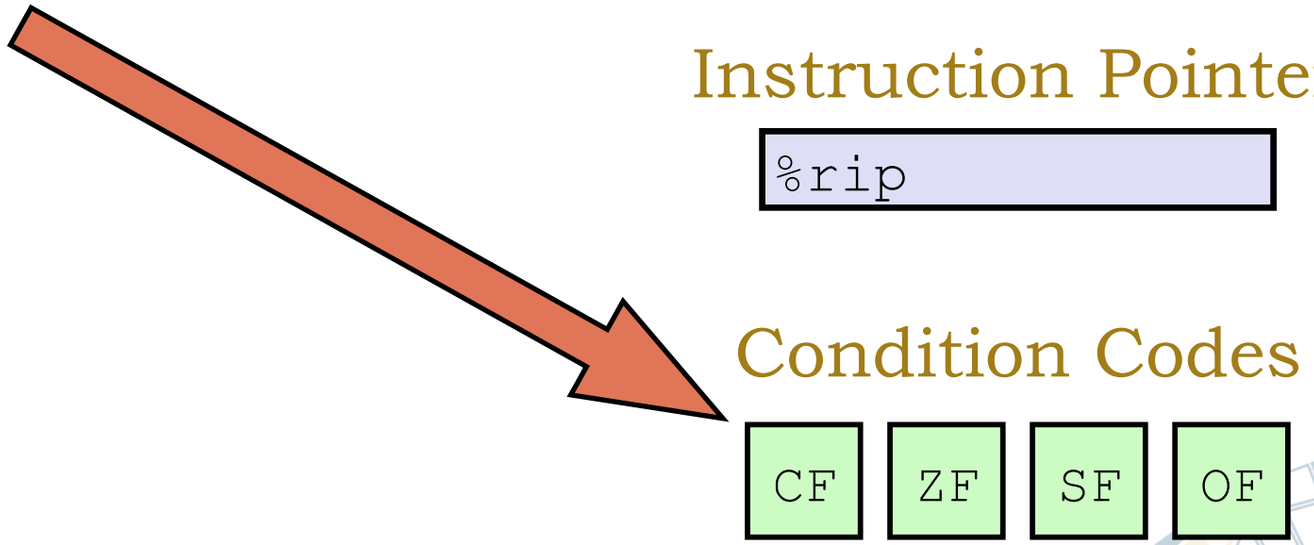


CONTROL

PROCESSOR STATE

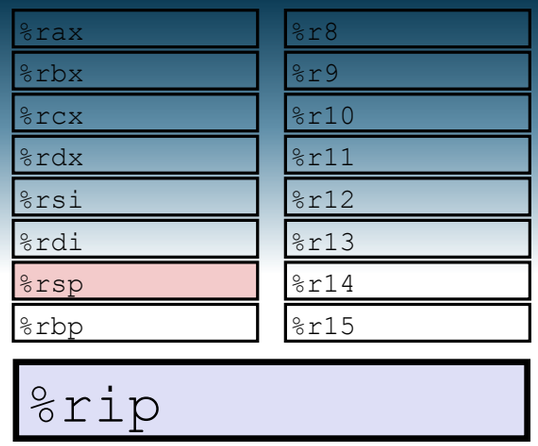
- Information about currently executing program
  - Temporary data (%rax, …)
  - Location of runtime stack (%rsp)
- Location of current code control point (%rip, …)
- Status of recent tests (CF, ZF, SF, OF)

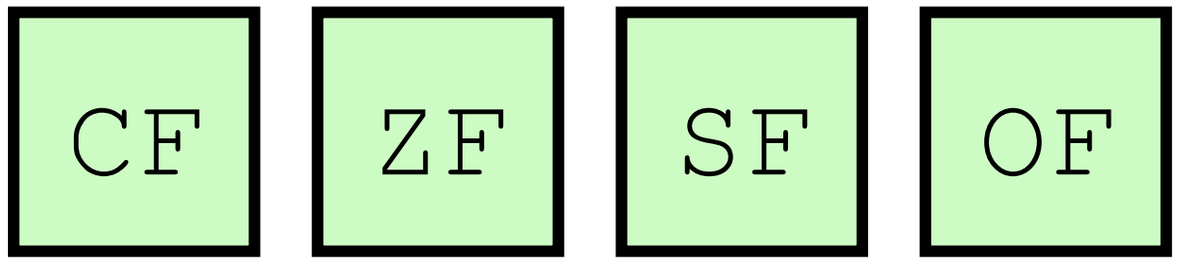




CONTROL

CONDITION CODES

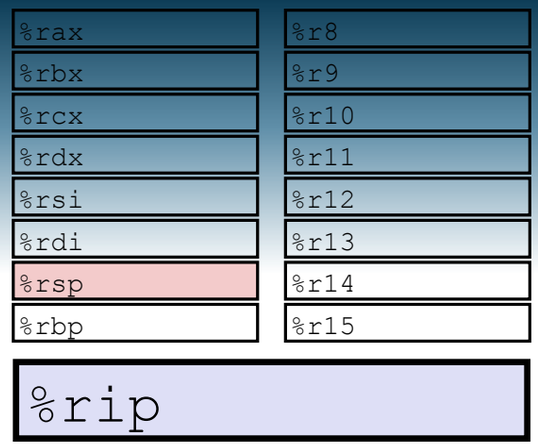


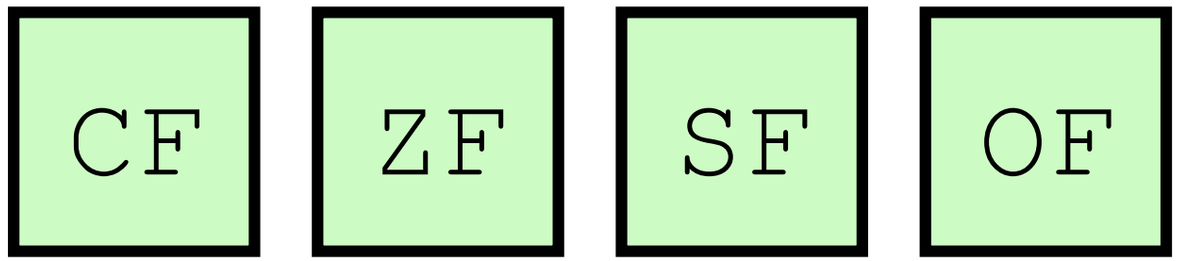


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

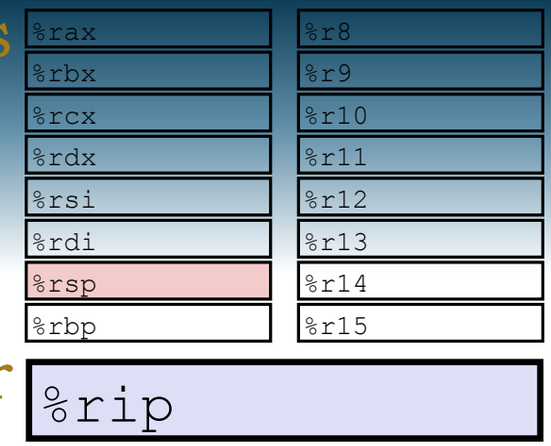


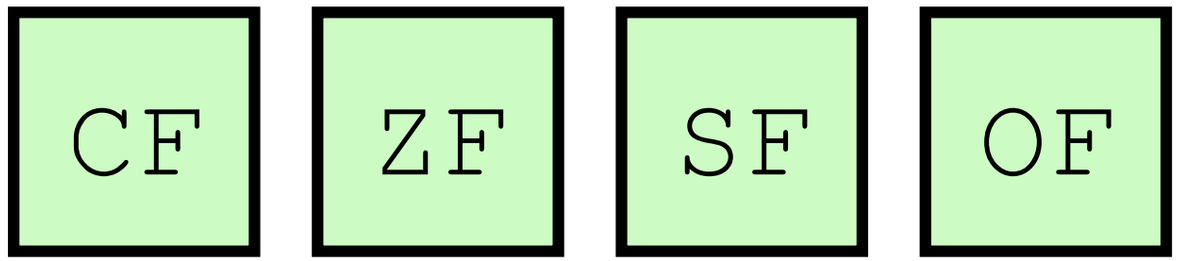


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

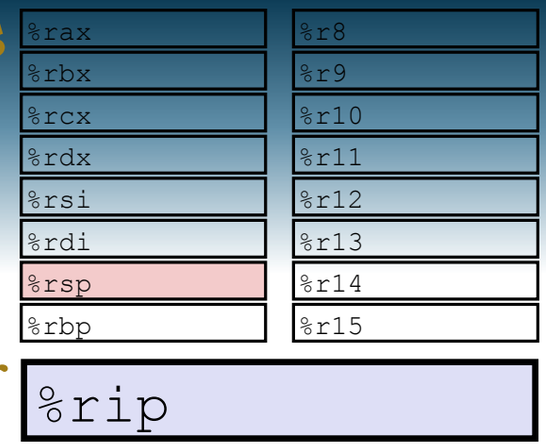




- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



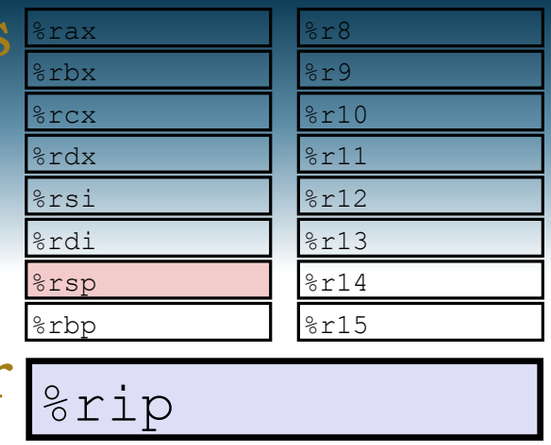
Condition Codes

CF ZF SF OF

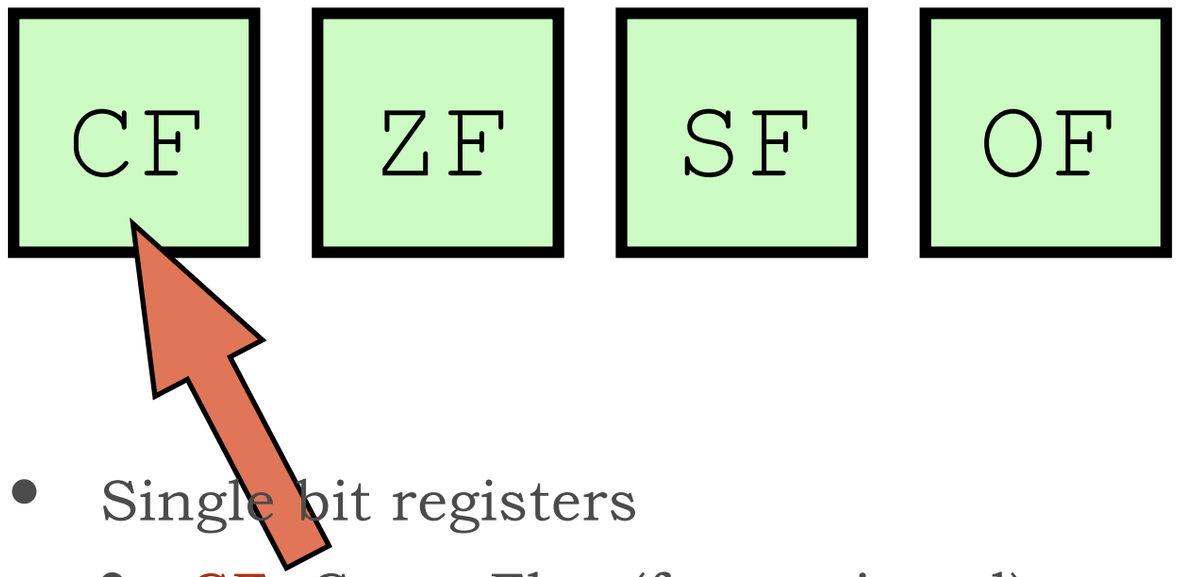
- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



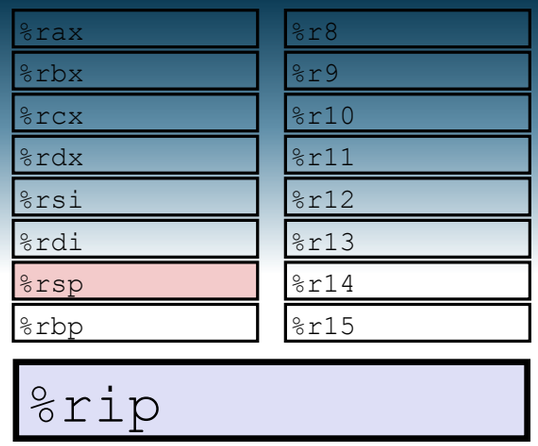
Condition Codes

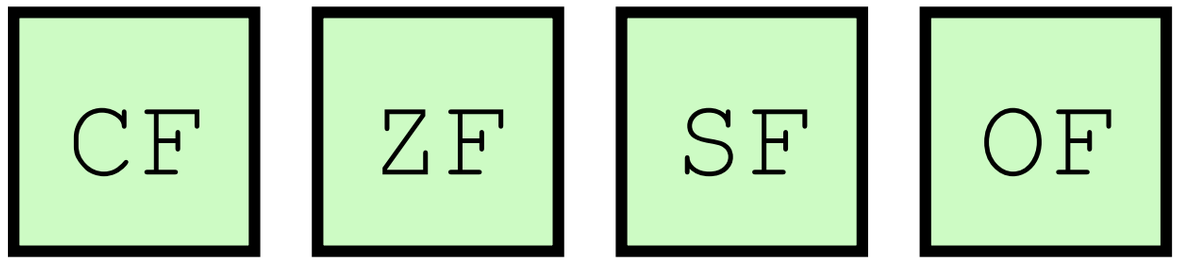


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

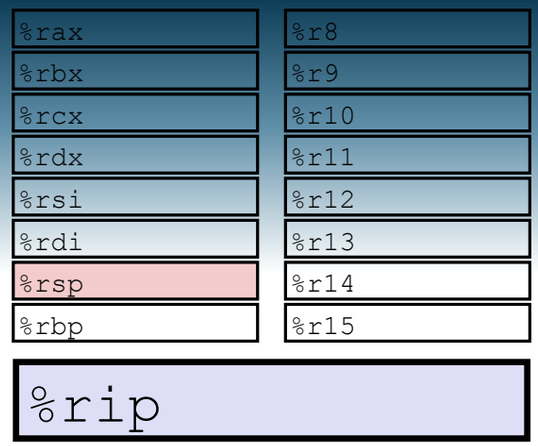


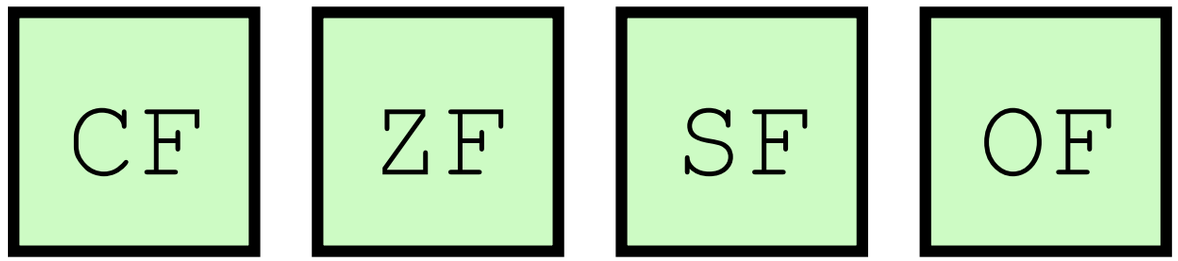


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

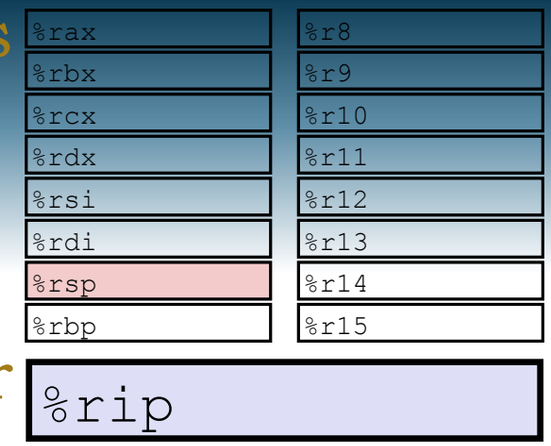


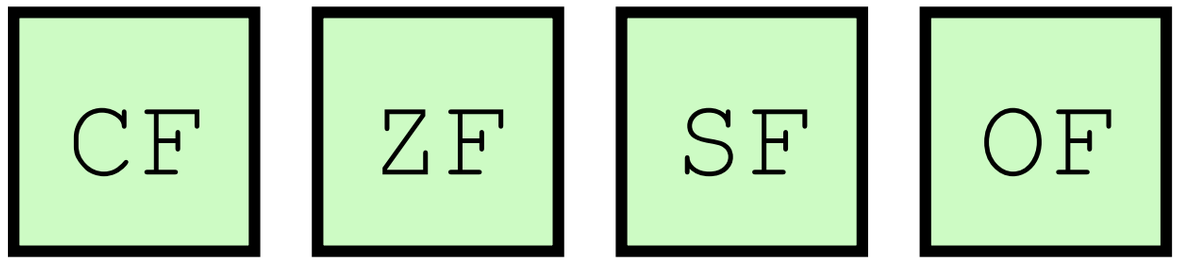


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

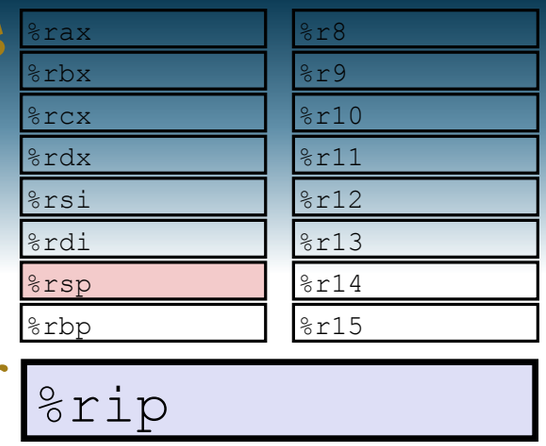




- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



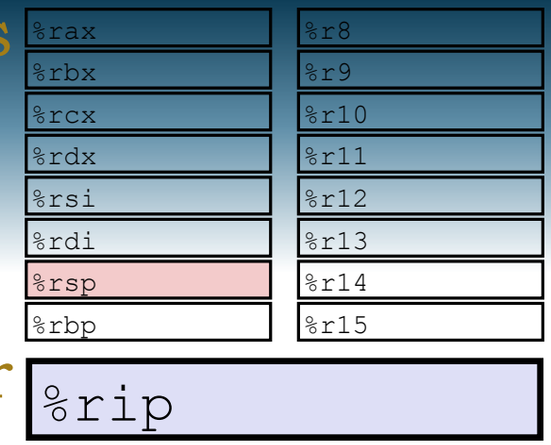
Condition Codes

CF ZF SF OF

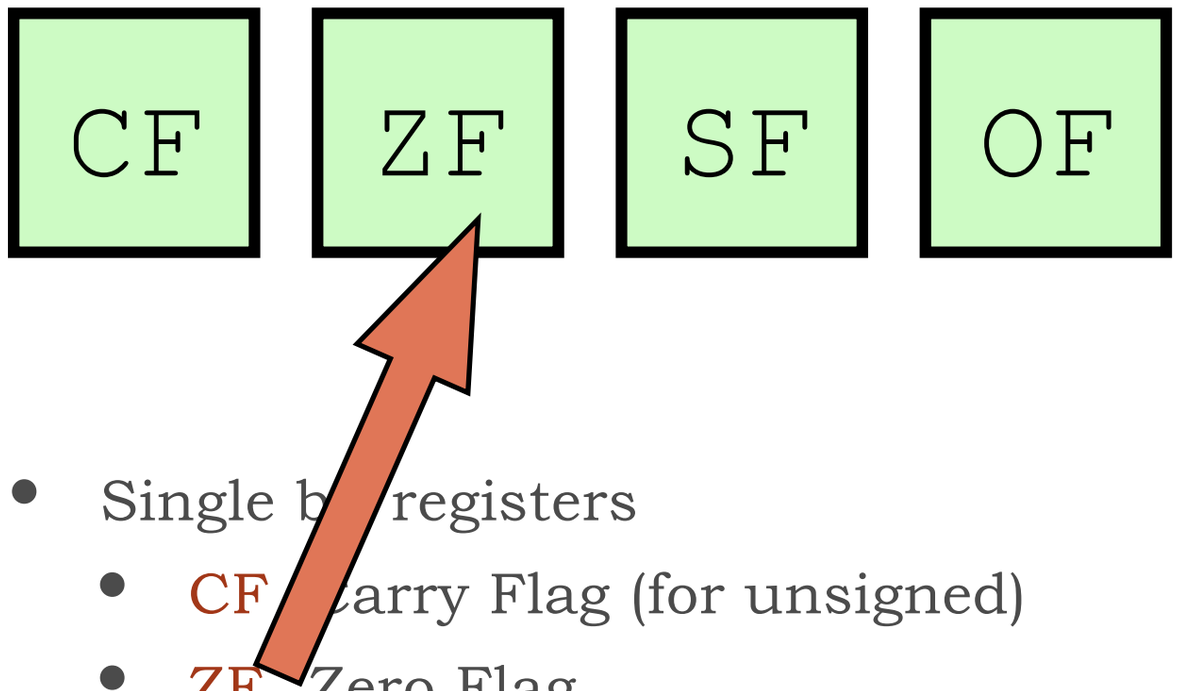
- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



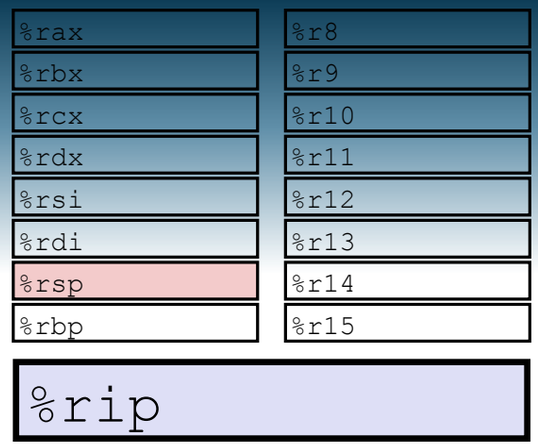
Condition Codes

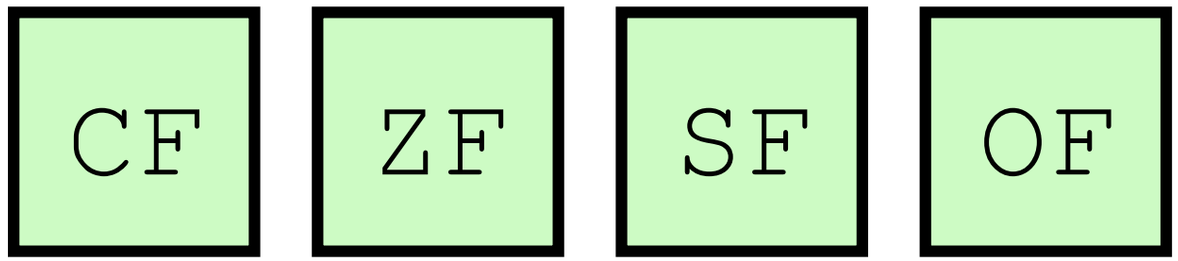


- Single b registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

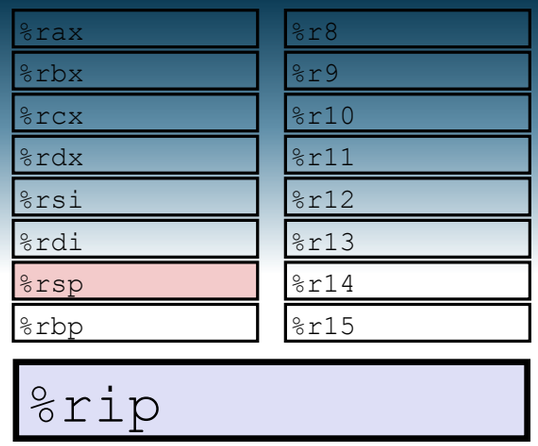


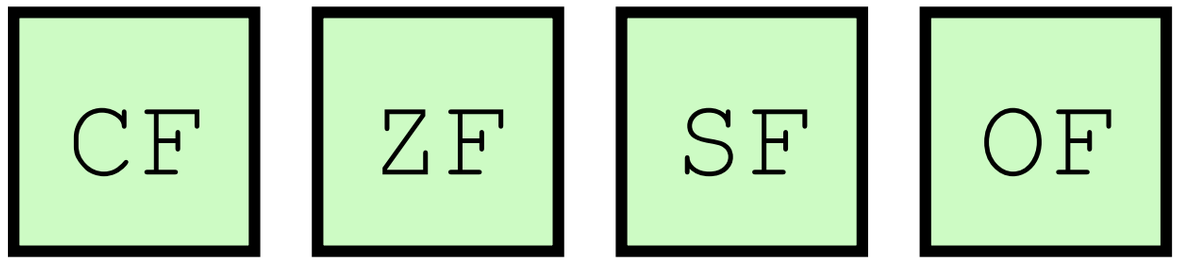


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

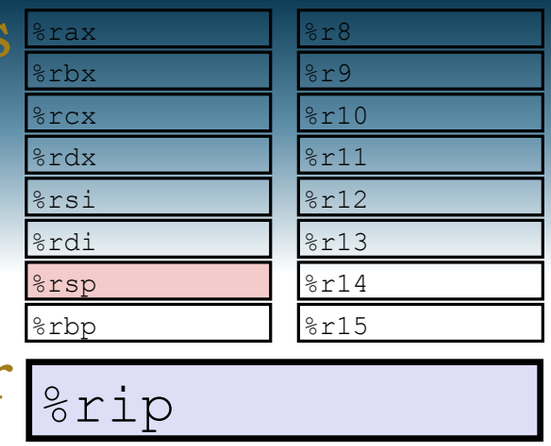


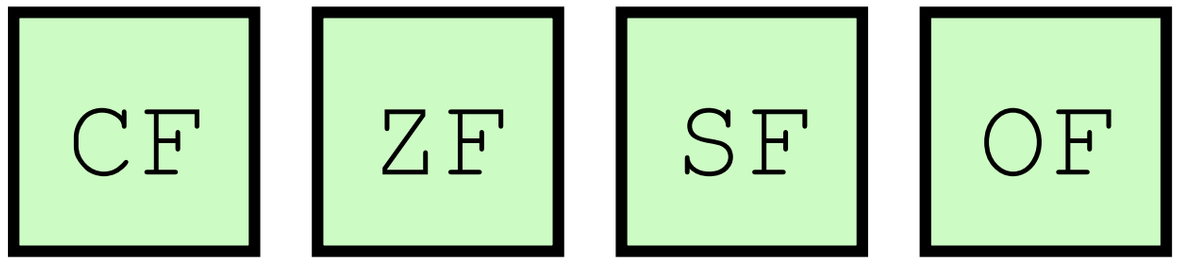


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

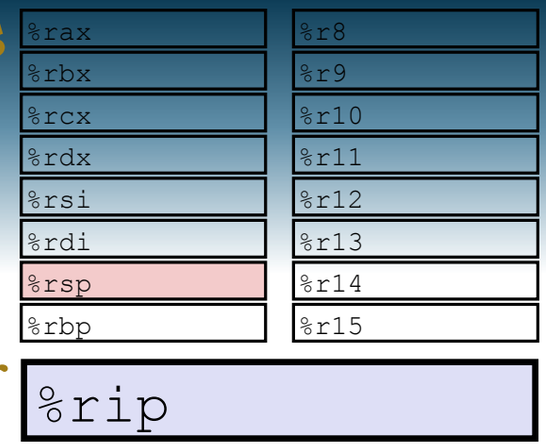




- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



Condition Codes

CF ZF SF OF

- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

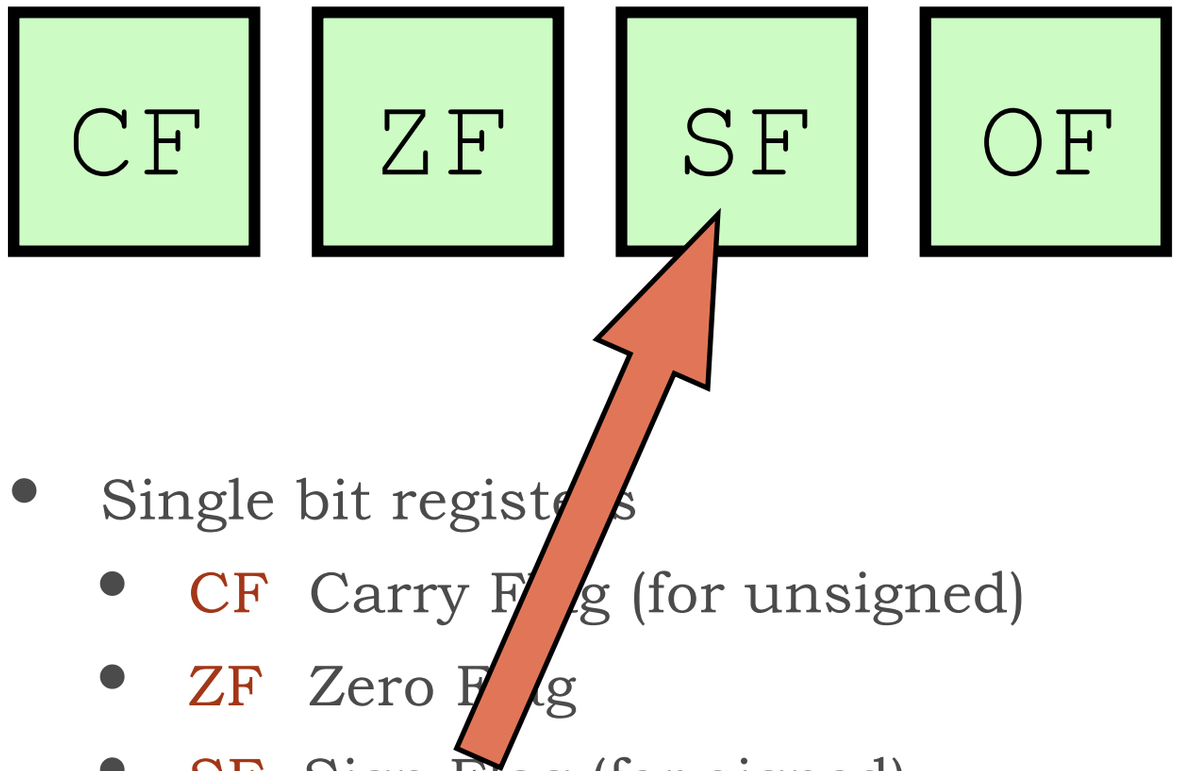
Registers

%rax	%r8
%rbx	%r9
%rcx	%r10
%rdx	%r11
%rsi	%r12
%rdi	%r13
%rsp	%r14
%rbp	%r15

Instruction Pointer

%rip

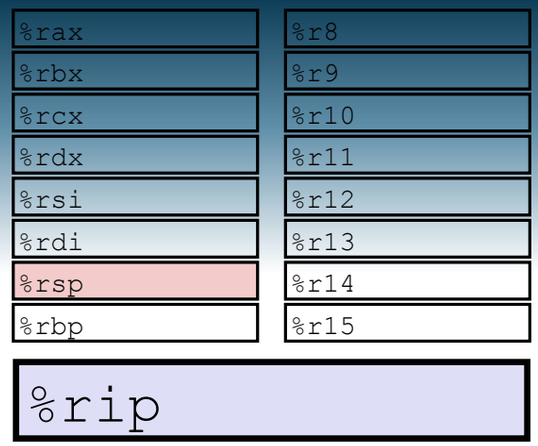
Condition Codes

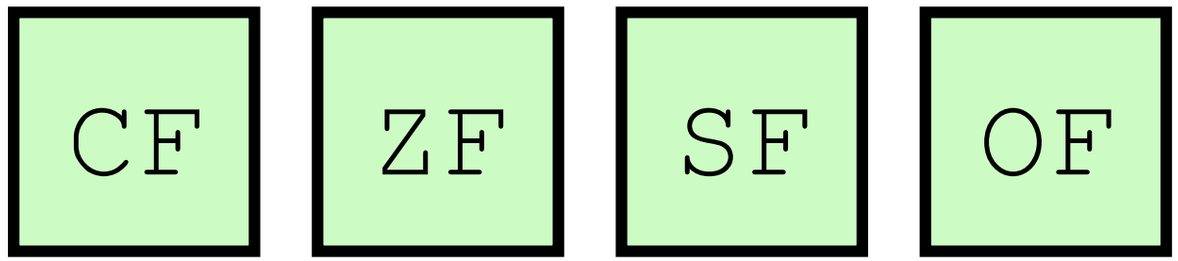


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

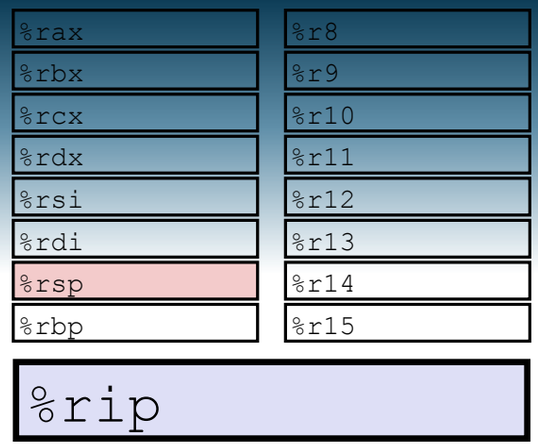


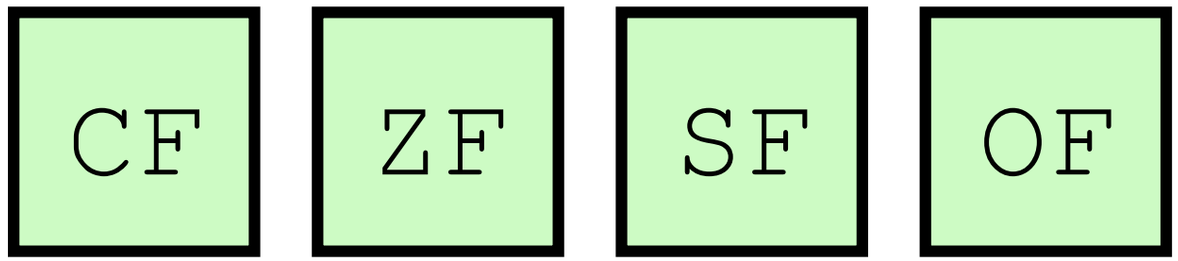


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

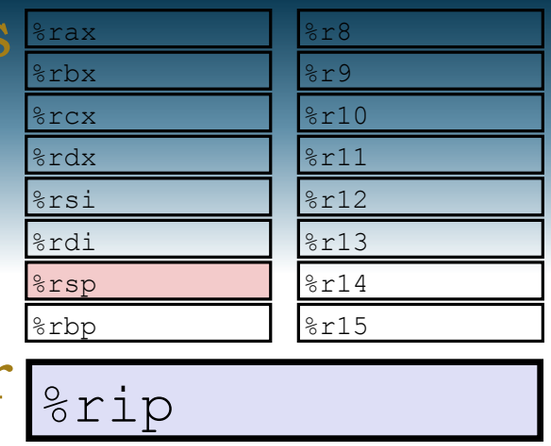


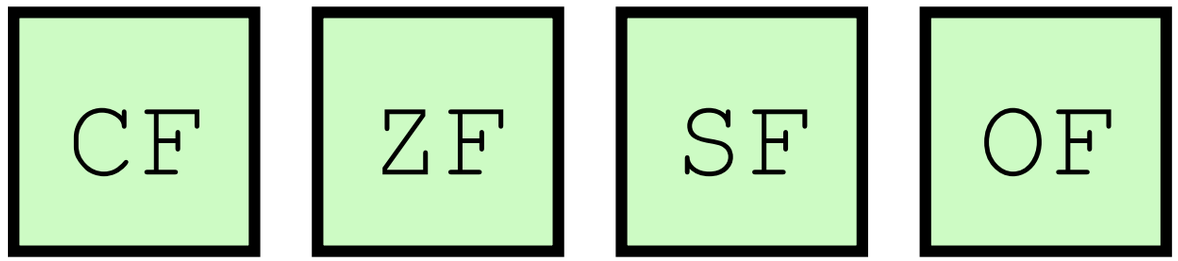


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES

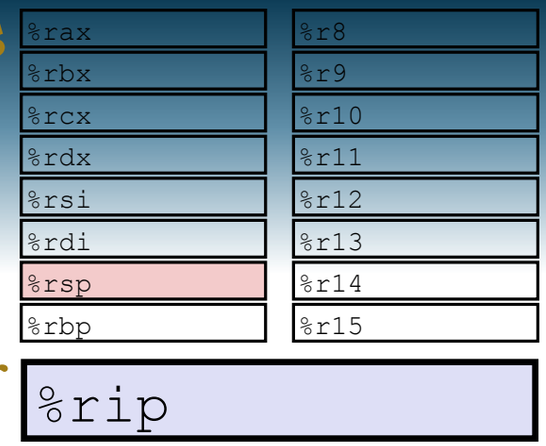




- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



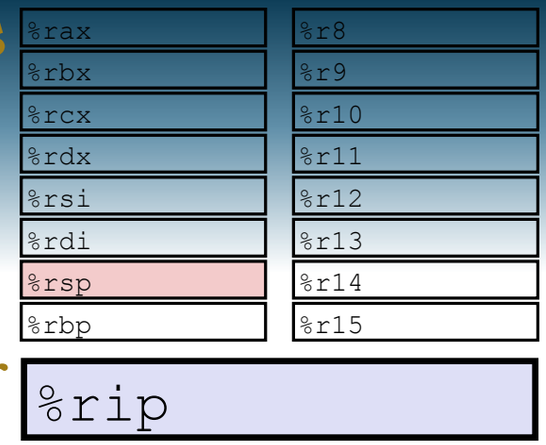
Condition Codes

CF ZF SF OF

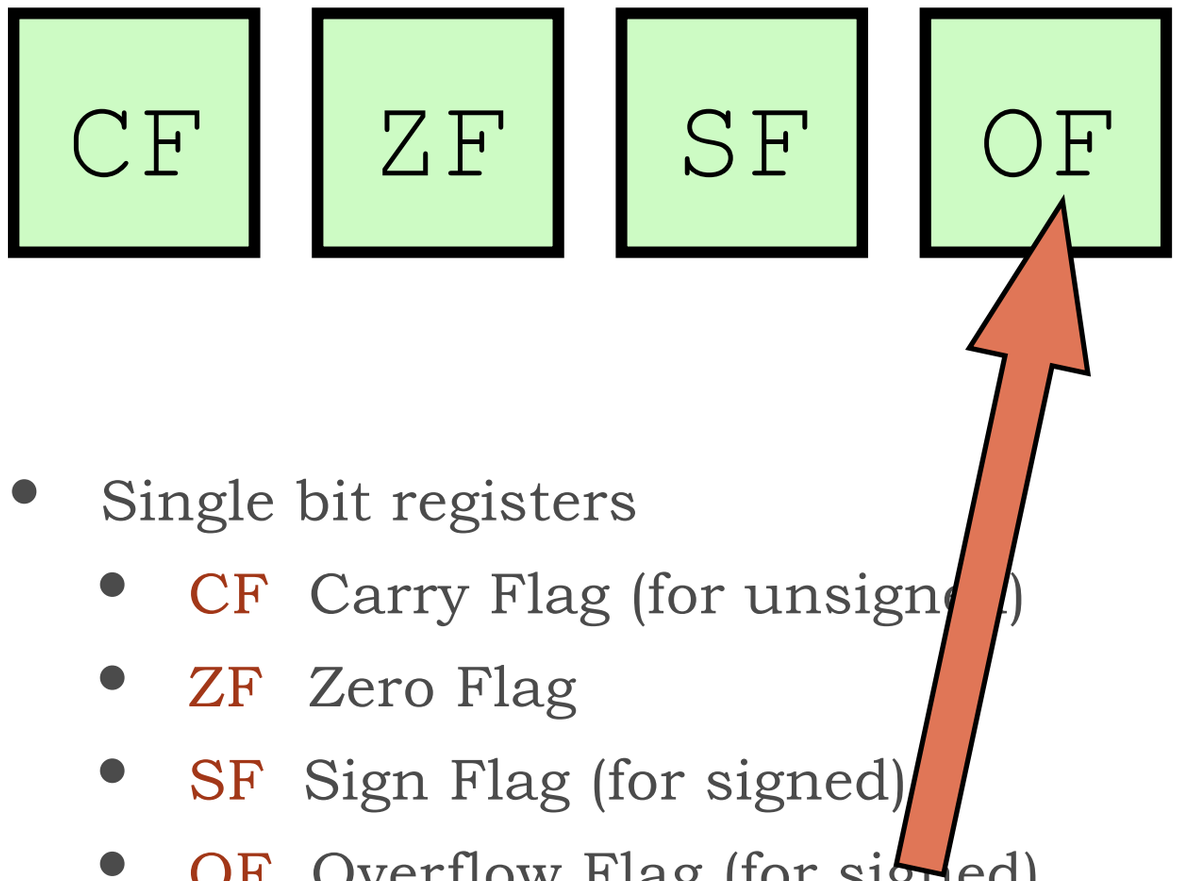
- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



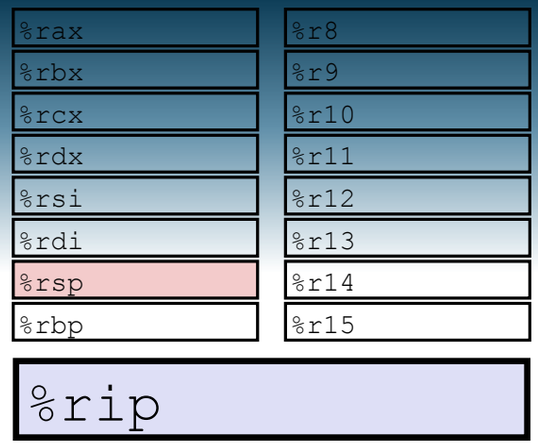
Condition Codes

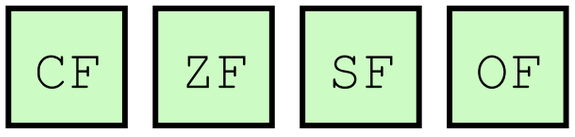


- Single bit registers
  - CF Carry Flag (for unsigned)
  - ZF Zero Flag
  - SF Sign Flag (for signed)
  - OF Overflow Flag (for signed)

CONTROL

CONDITION CODES



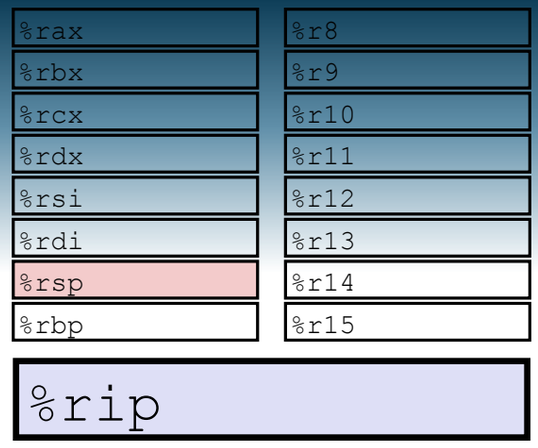


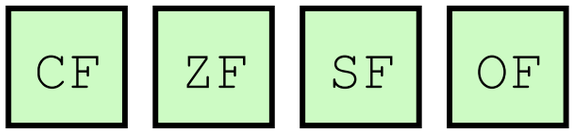
- **Implicitly set**
  - (think of it as side effect) by arithmetic operations
    - Example: `addq src, dest` ↔ t = a+b
      - CF set if carry out from most significant bit
        - (unsigned overflow)
      - ZF set if t == 0
      - SF set if t < 0
        - (as signed)
      - OF set if two's-complement (signed) overflow
        (a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

- **Not set by leaq instruction**

CONTROL

CONDITION CODES



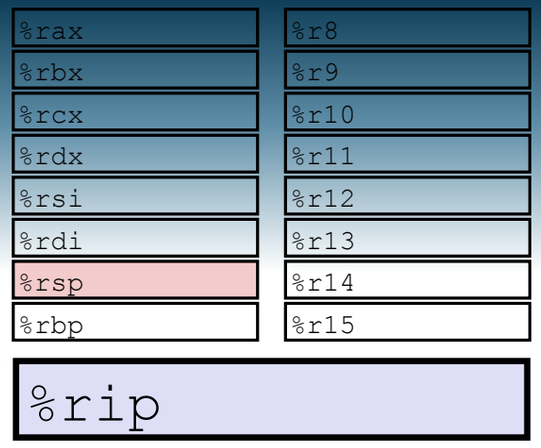


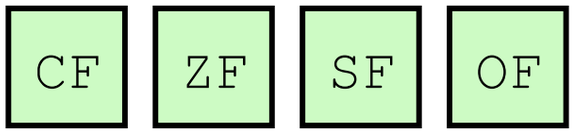
- **Implicitly set**
  - (think of it as side effect) by arithmetic operations
    - Example: `addq src, dest` ↔ t = a+b
      - CF set if carry out from most significant bit
        - (unsigned overflow)
      - ZF set if t == 0
      - SF set if t < 0
        - (as signed)
      - OF set if two's-complement (signed) overflow
        (a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

- **Not set by leaq instruction**

CONTROL

CONDITION CODES



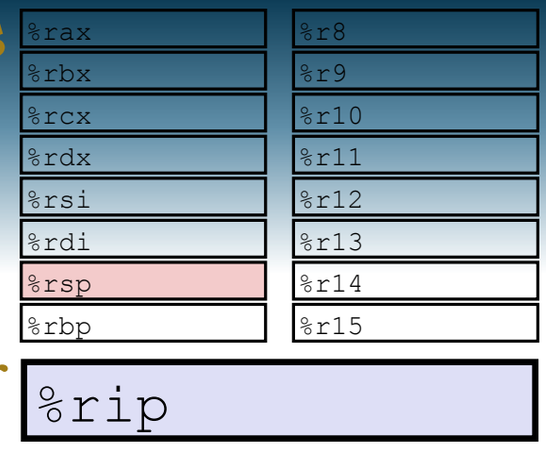


- **Implicitly set**
  - (think of it as side effect) by arithmetic operations
    - Example: `addq src, dest` ↔ t = a+b
      - CF set if carry out from most significant bit
        - (unsigned overflow)
      - ZF set if t == 0
      - SF set if t < 0
        - (as signed)
      - OF set if two's-complement (signed) overflow
        (a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

- **Not set by leaq instruction**

CONTROL

CONDITION CODES

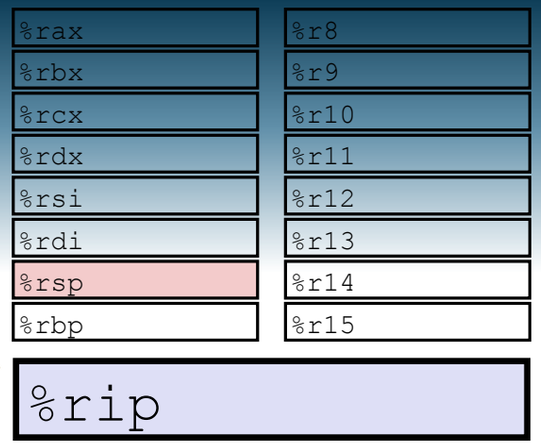


- **Implicitly set**
  - (think of it as side effect) by arithmetic operations
    - Example: `addq src, dest` ↔ t = a+b
      - CF set if carry out from most significant bit
        - (unsigned overflow)
      - ZF set if t == 0
      - SF set if t < 0
        - (as signed)
      - OF set if two's-complement (signed) overflow
        (a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

- **Not set by leaq instruction**

CONTROL

CONDITION CODES

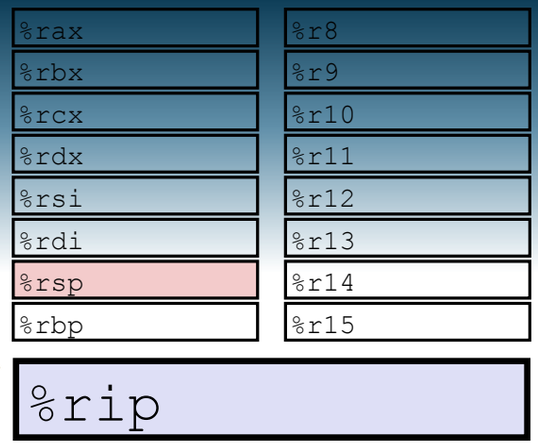


- **Implicitly set**
  - (think of it as side effect) by arithmetic operations
    - Example: `addq src, dest` → t = a+b
      - CF set if carry out from most significant bit
        - (unsigned overflow)
      - ZF set if t == 0
      - SF set if t < 0
        - (as signed)
      - OF set if two's-complement (signed) overflow
        (a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

- **Not set by leaq instruction**

CONTROL

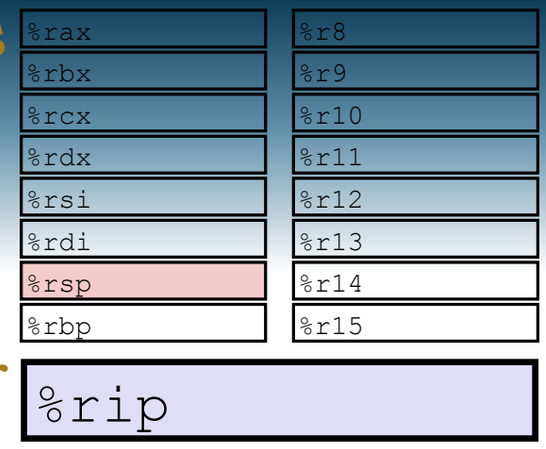
CONDITION CODES



- **Explicitly Setting** by Compare Instruction
  - `cmpq src2, src1`
  - `cmpq b,a`
    - like computing a-b without setting destination
    - CF set if carry out from most significant bit (used for unsigned comparisons)
    - ZF set if a == b
    - SF set if (a-b) < 0 (as signed)
    - OF set if two's-complement (signed) overflow
      (a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0)

CONTROL

CONDITION CODES



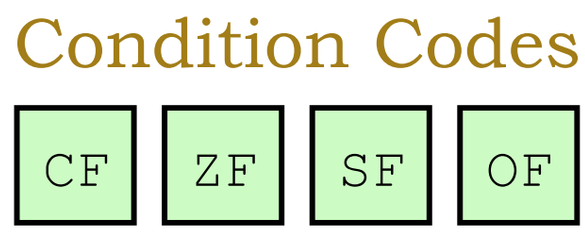
- **Explicitly Setting** by Test instruction
  - `testq src2, src1`
  - `testq b,a`
    - like computing a&b without setting destination
- Sets condition codes based on value of src1 & src2
  - Useful to have one of the operands be a mask
- ZF set when a&b == 0
- SF set when a&b < 0

CONTROL

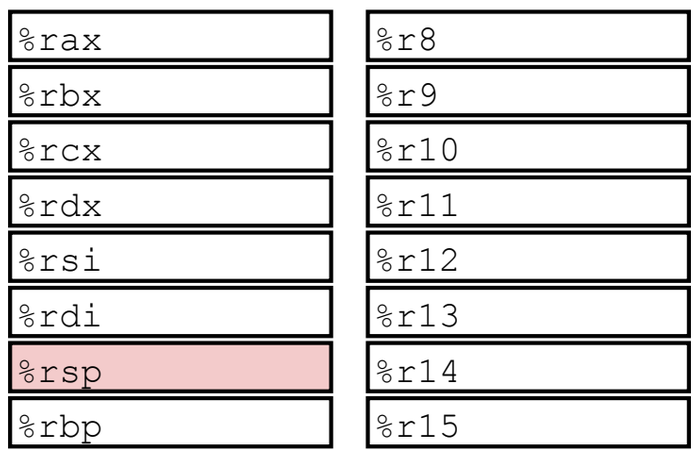
READING CONDITION CODES

- SetX Instructions
  - Set low-order byte of destination to 0 or 1 based on combinations of condition codes
  - Does not alter remaining 7 bytes

SetX	Condition	Description
`sete`	`ZF`	Equal / Zero
`setne`	`~ZF`	Not Equal / Not Zero
`sets`	`SF`	Negative
`setns`	`~SF`	Nonnegative
`setg`	`~(SF^OF)&~ZF`	Greater (Signed)
`setge`	`~(SF^OF)`	Greater or Equal (Signed)
`setl`	`(SF^OF)`	Less (Signed)
`setle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`seta`	`~CF&~ZF`	Above (unsigned)
`setb`	`CF`	Below (unsigned)



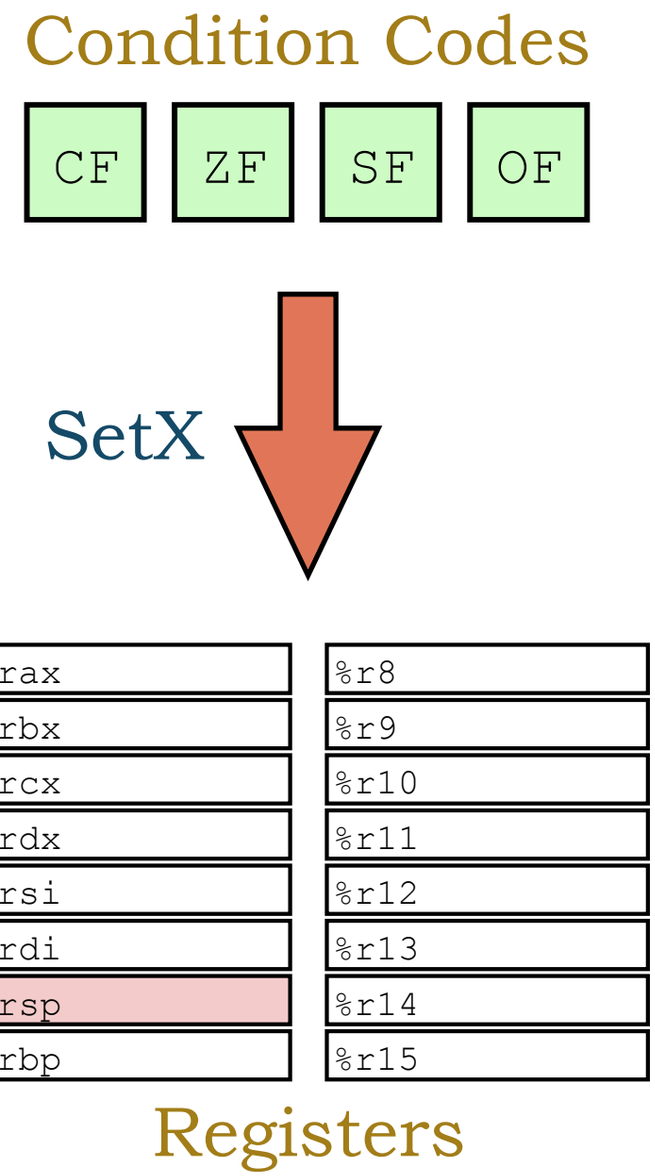
SetX



READING CONDITION CODES

- SetX Instructions
  - Set low-order byte of destination to 0 or 1 based on combinations of condition codes
  - Does not alter remaining 7 bytes

SetX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF)&~ZF	Greater (Signed)
setge	~(SF^OF)	Greater or Equal (Signed)
setl	(SF^OF)	Less (Signed)
setle	(SF^OF)|ZF	Less or Equal (Signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

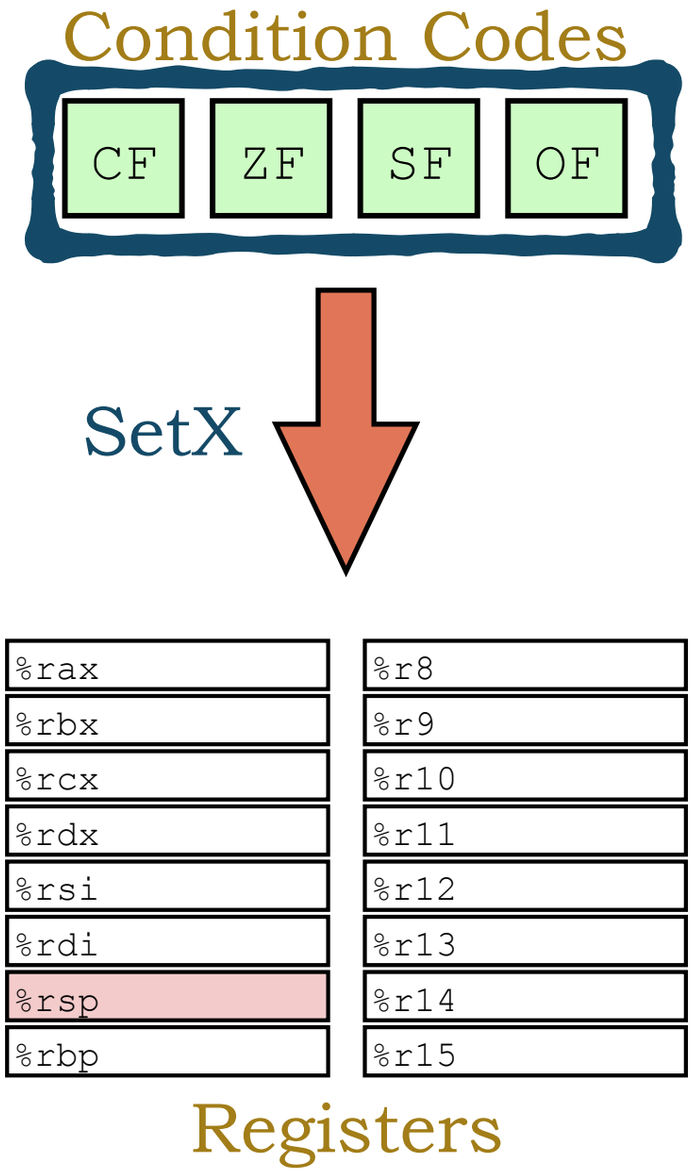


CONTROL

READING CONDITION CODES

- SetX Instructions
  - Set low-order byte of destination to 0 or 1 based on combinations of condition codes
  - Does not alter remaining 7 bytes

SetX	Condition	Description
`sete`	`ZF`	Equal / Zero
`setne`	`~ZF`	Not Equal / Not Zero
`sets`	`SF`	Negative
`setns`	`~SF`	Nonnegative
`setg`	`~(SF^OF)&~ZF`	Greater (Signed)
`setge`	`~(SF^OF)`	Greater or Equal (Signed)
`setl`	`(SF^OF)`	Less (Signed)
`setle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`seta`	`~CF&~ZF`	Above (unsigned)
`setb`	`CF`	Below (unsigned)

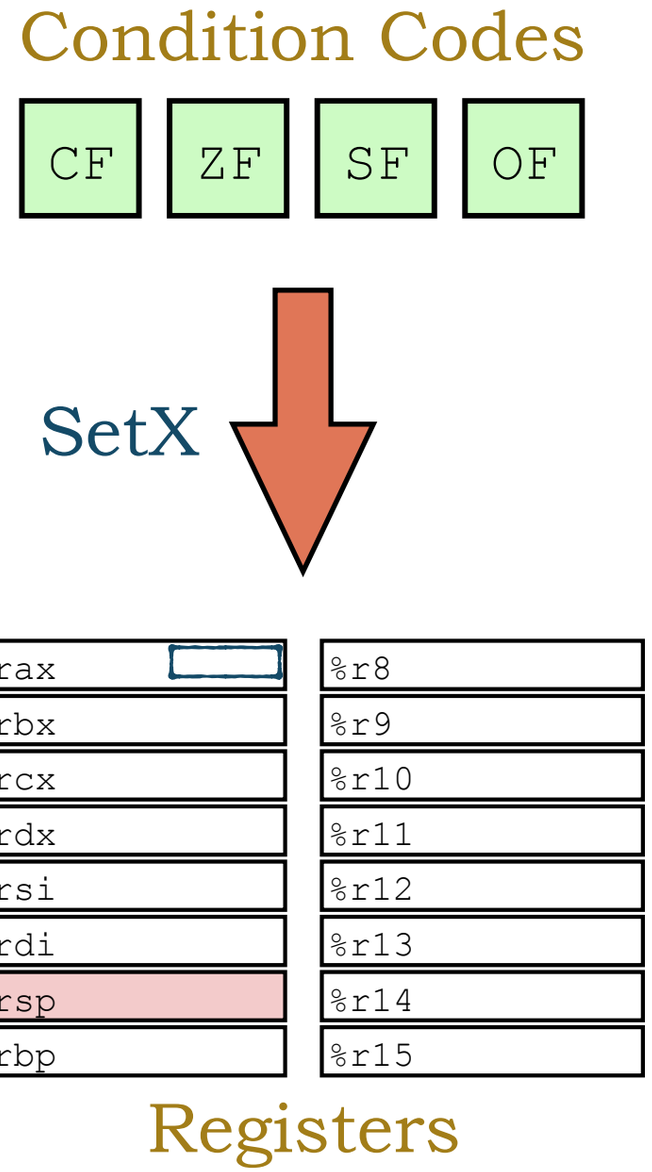


CONTROL

READING CONDITION CODES

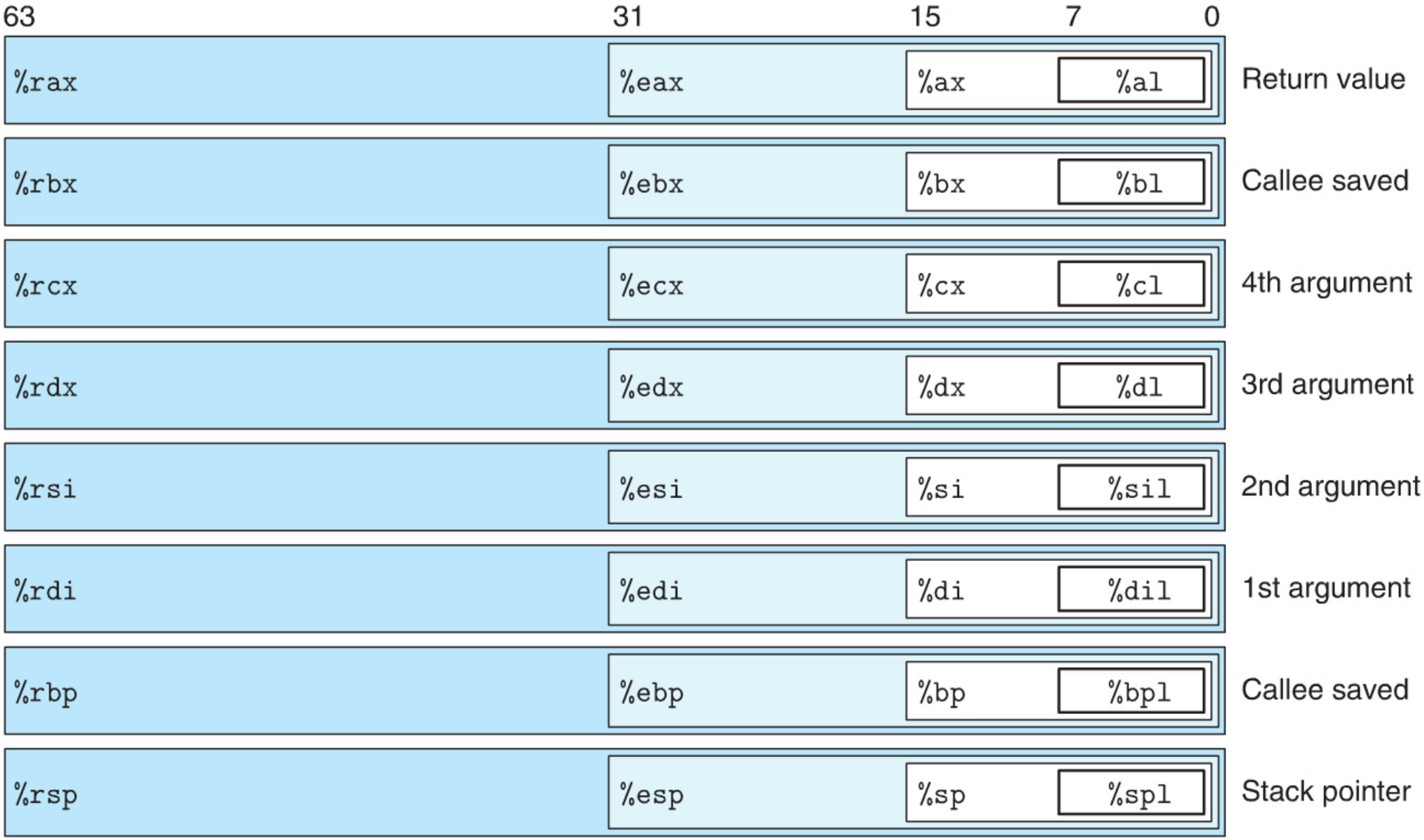
- SetX Instructions
  - Set low-order byte of destination to 0 or 1 based on combinations of condition codes
  - Does not alter remaining 7 bytes

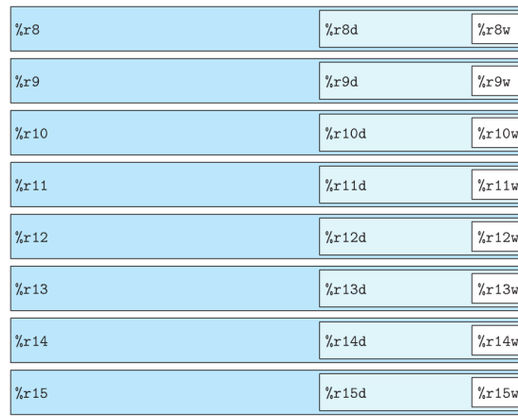
SetX	Condition	Description
`sete`	`ZF`	Equal / Zero
`setne`	`~ZF`	Not Equal / Not Zero
`sets`	`SF`	Negative
`setns`	`~SF`	Nonnegative
`setg`	`~(SF^OF)&~ZF`	Greater (Signed)
`setge`	`~(SF^OF)`	Greater or Equal (Signed)
`setl`	`(SF^OF)`	Less (Signed)
`setle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`seta`	`~CF&~ZF`	Above (unsigned)
`setb`	`CF`	Below (unsigned)



CONTROL

READING CONDITION CODES

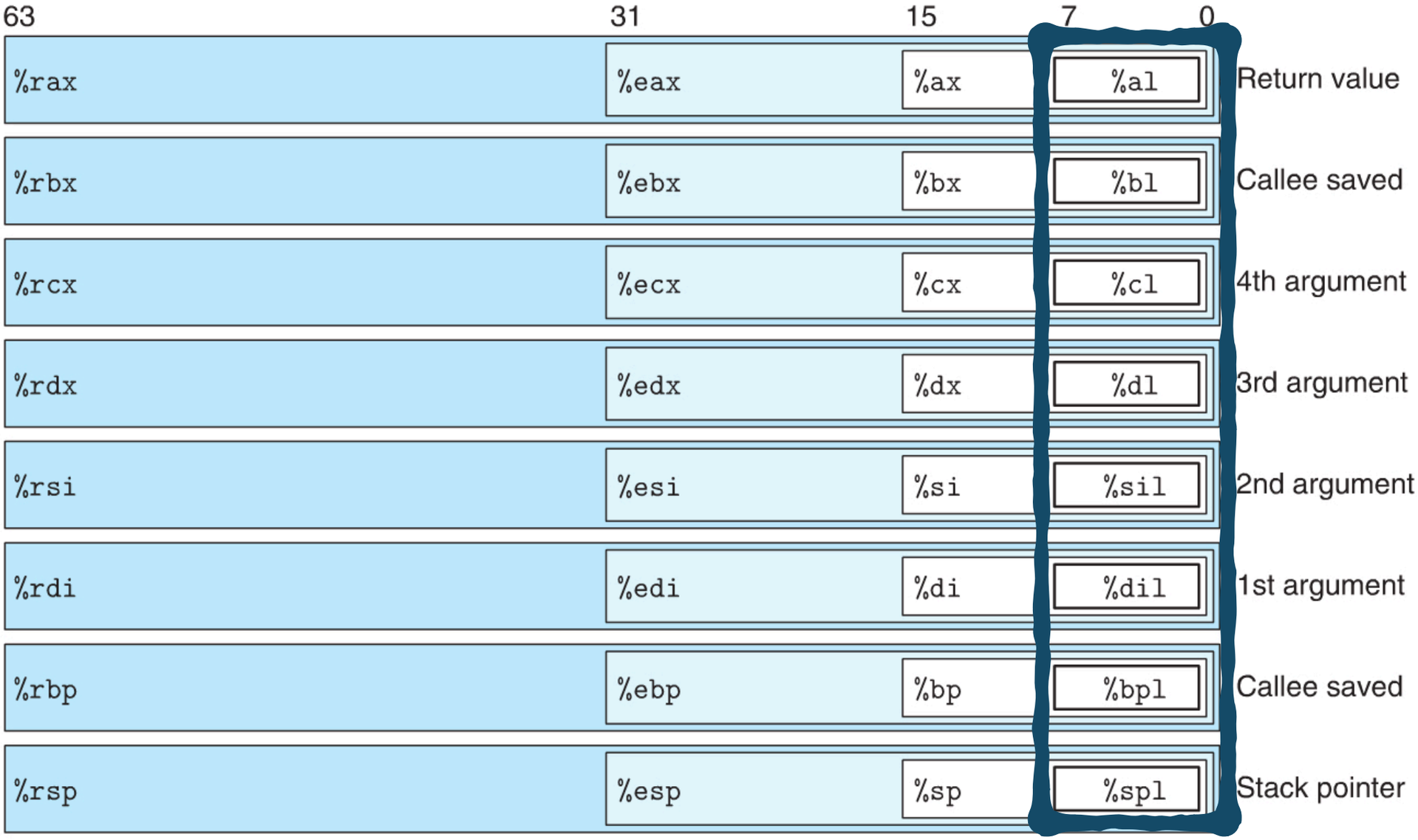


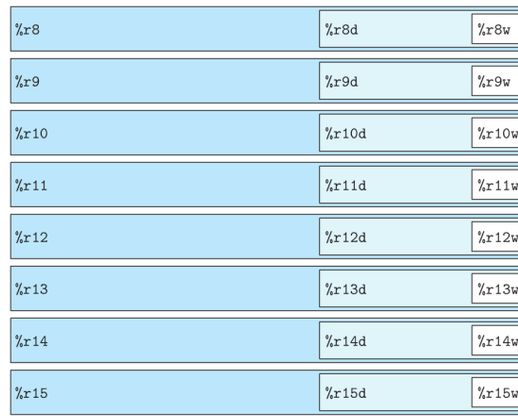


- **Can reference low-order bytes**

CONTROL

READING CONDITION CODES

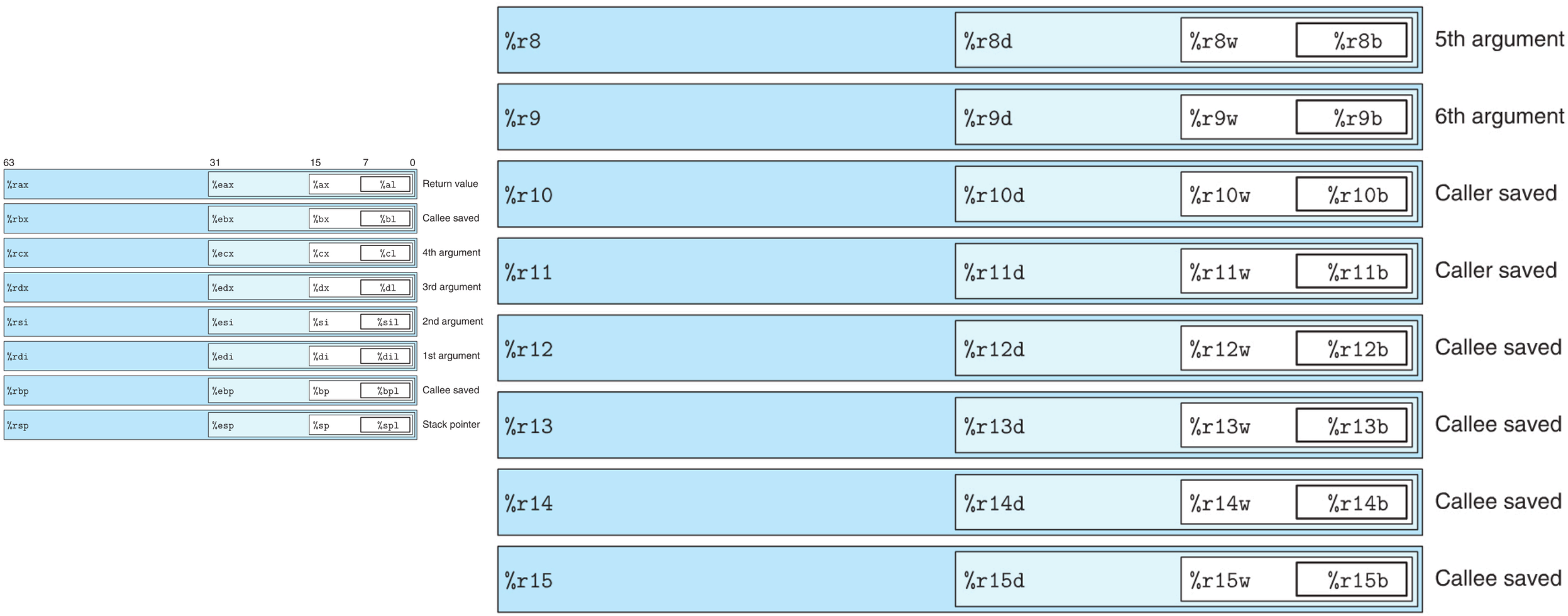




- **Can reference low-order bytes**

CONTROL

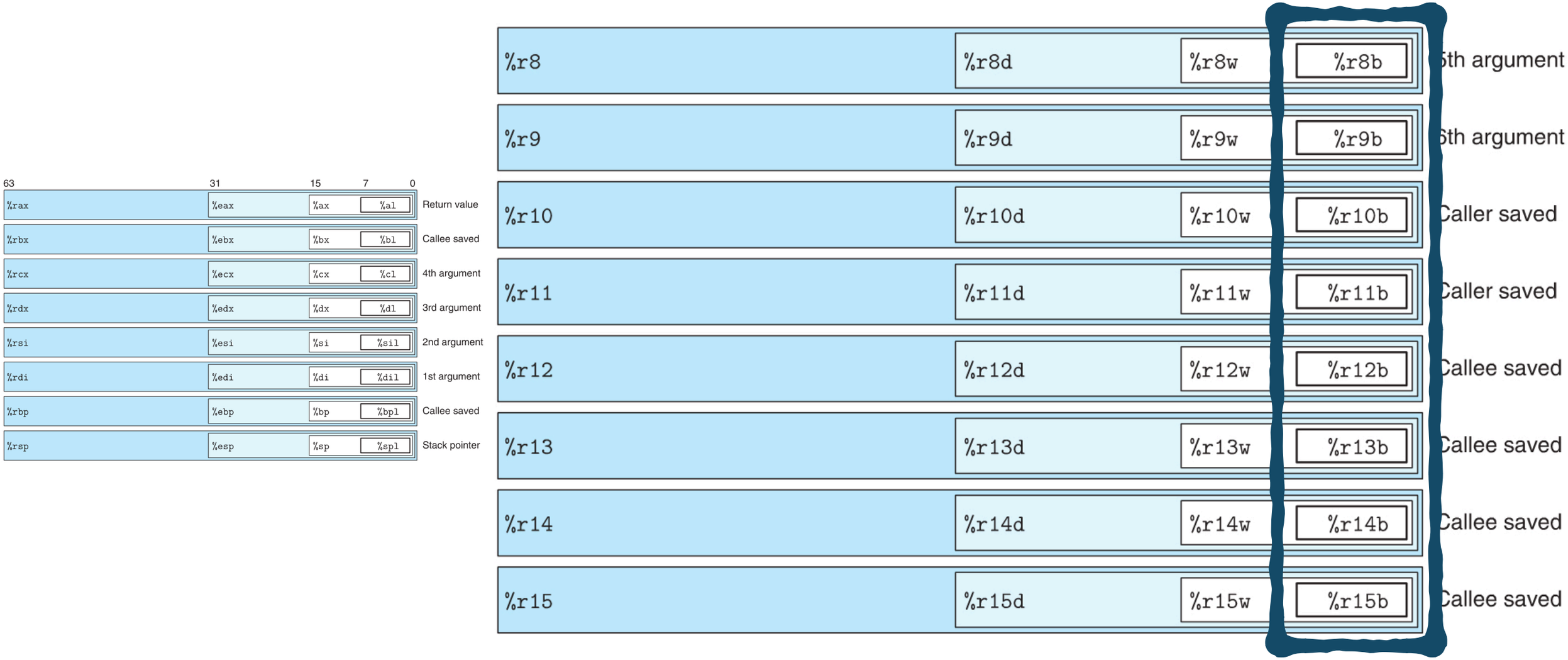
READING CONDITION CODES



- **Can reference low-order bytes**

CONTROL

READING CONDITION CODES



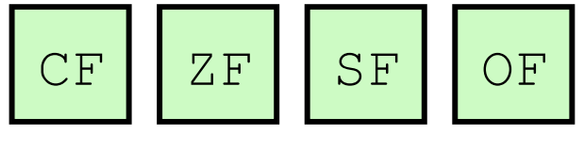
- **Can reference low-order bytes**

CONTROL

READING CONDITION CODES

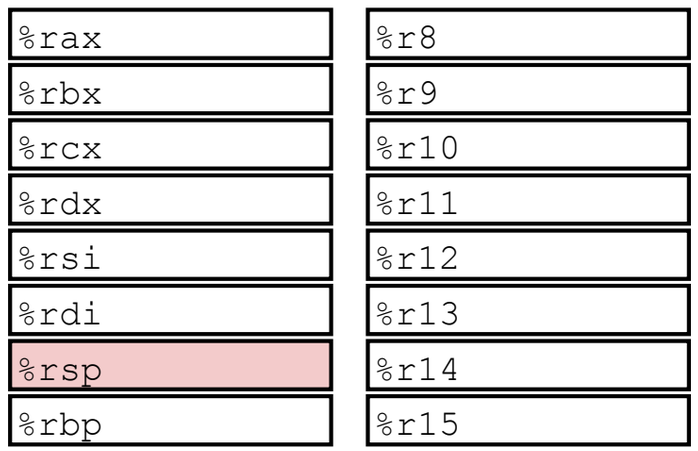
- SetX Instructions
  - Set single byte based on combination of condition codes
- One of addressable byte registers
  - Does not alter remaining bytes
- Typically use `movzbl` to finish job
  - 32-bit instructions also set upper 32 bits to 0

Condition Codes



SetX

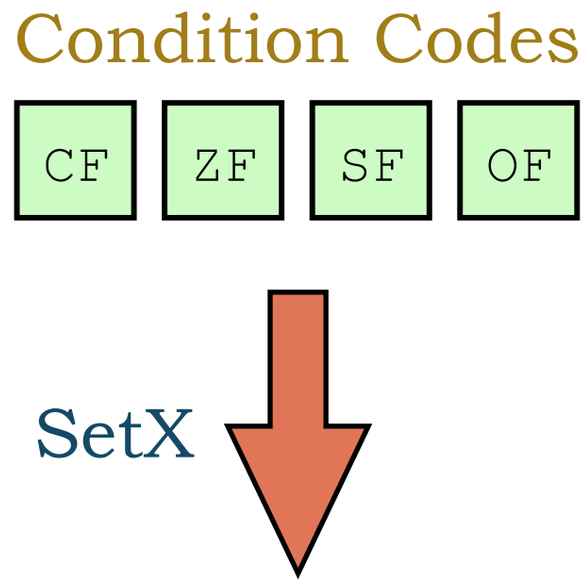
Greater Than Example



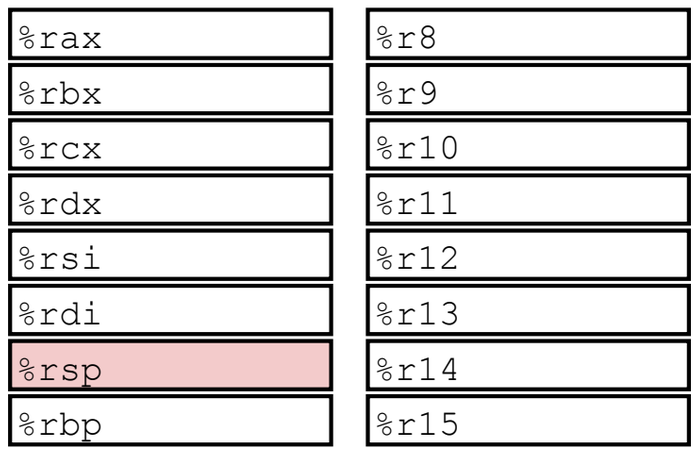
Registers

READING CONDITION CODES

- SetX Instructions
  - Set single byte based on combination of condition codes
- One of addressable byte registers
  - Does not alter remaining bytes
  - Typically use `movzbl` to finish job
  - 32-bit instructions also set upper 32 bits to 0



Greater Than Example



KEY DISASSEMBLY COMMANDS

`wcpkneel: objdump -d file`

disassemble the executable parts of file

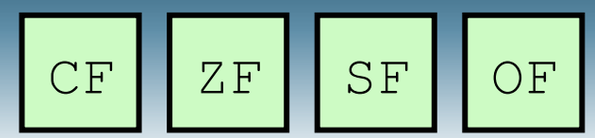
`wcpkneel: objdump -x file`

disassemble all parts of file and show sections

`wcpkneel: objdump -j.rodata file`

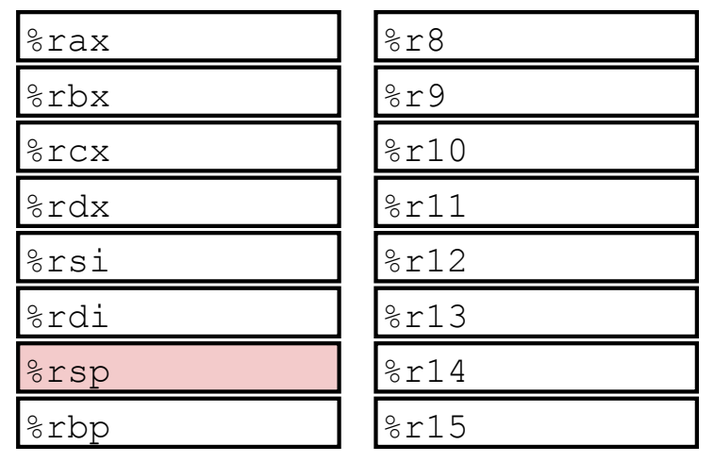
disassemble just the .rodata section of file

CONTROL



READING CONDITION CODES

- SetX Instructions
  - Set single byte based on combination of condition codes
- One of addressable byte registers
  - Does not alter remaining bytes
  - Typically use `movzbl` to finish job
    - 32-bit instructions also set upper 32 bits to 0

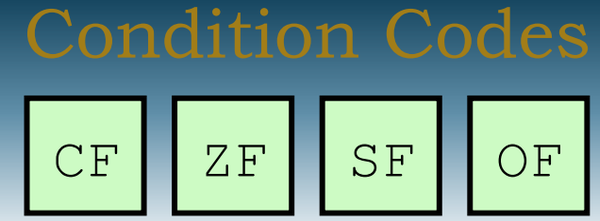



```
int gt (long x, long y)
{
  return x > y;
}
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

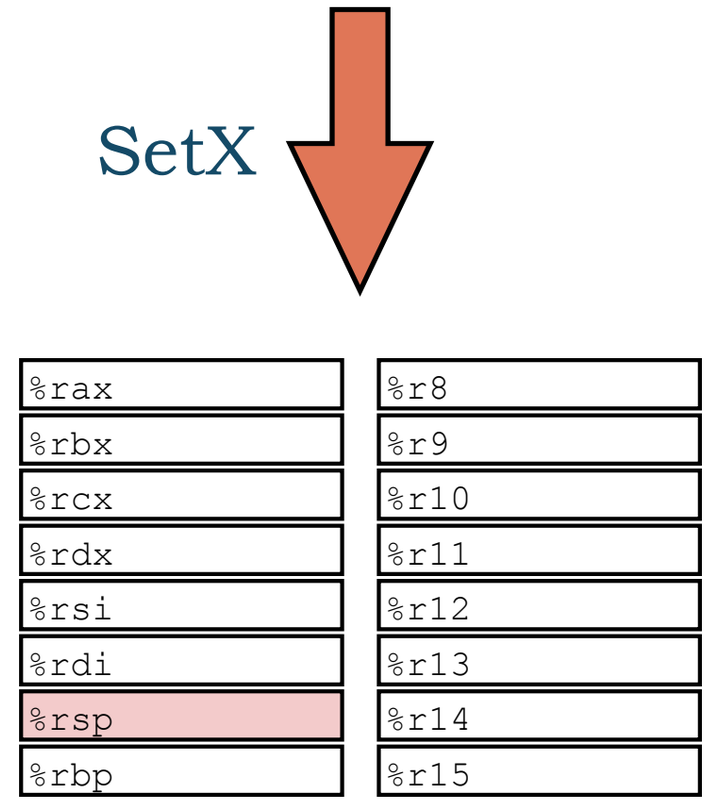
```
    cmpq    %rsi, %rdi    # Compare x:y
    setg    %al           # Set when >
    movzbl  %al, %eax     # Zero rest of %rax
    ret
```

CONTROL



READING CONDITION CODES

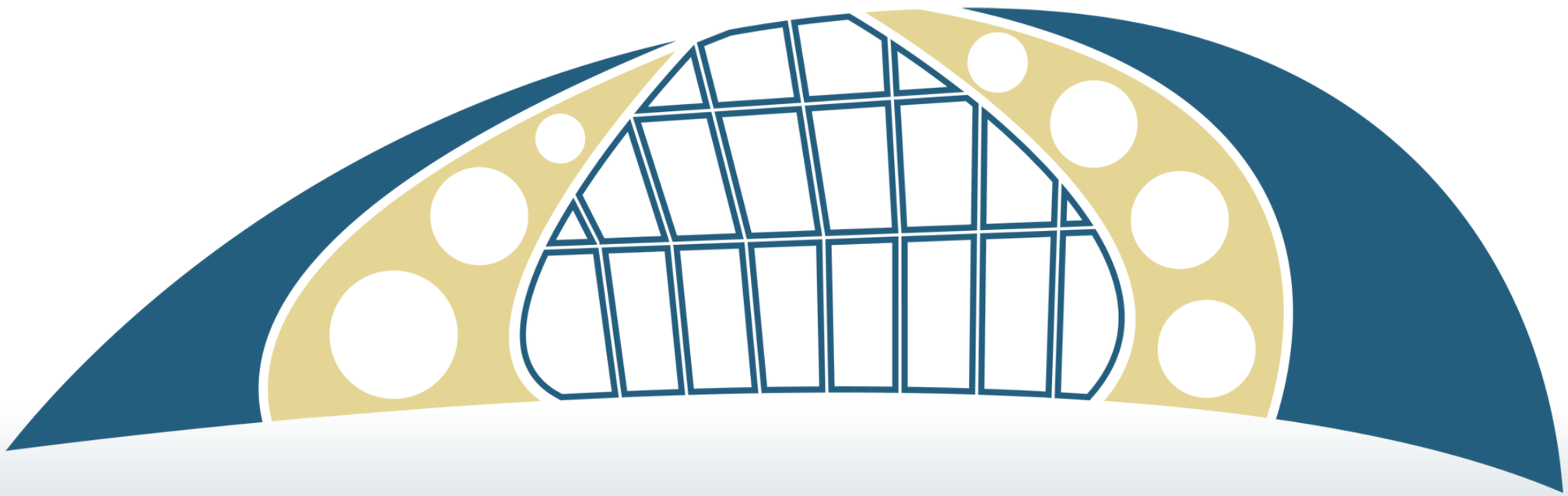
- SetX Instructions
  - Set single byte based on combination of condition codes
- One of addressable byte registers
  - Does not alter remaining bytes
  - Typically use `movzbl` to finish job
    - 32-bit instructions also set upper 32 bits to 0




```
int gt (long x, long y)
{
  return x > y;
}
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

```
    cmpq    %rsi, %rdi    # Compare x:y
    setg    %al           # Set when >
    movzbl  %al, %eax     # Zero rest of %rax
    ret
```



WESTMONT INSPIRED

COMPUTING LAB

```c
#include <stdio.h>

int greater_than(long x, long y)
{
    return x > y;
}


int main(){
    int result = greater_than(1,0);

    if(result){
        printf("1 is greater than 0\n");
    }
    else{
        printf("1 is not greater than 0\n");
    }
}
```

```
_greater_than:
100000ef0:  55  pushq   %rbp
100000ef1:  48 89 e5    movq    %rsp, %rbp
100000ef4:  48 39 f7    cmpq    %rsi, %rdi
100000ef7:  0f 9f c0    setg    %al
100000efa:  0f b6 c0    movzbl  %al, %eax
100000efd:  5d  popq    %rbp
100000efe:  c3  retq
100000eff:  90  nop

_main:
100000f00:  55  pushq   %rbp
100000f01:  48 89 e5    movq    %rsp, %rbp
100000f04:  bf 01 00 00 00  movl    $1, %edi
100000f09:  31 f6   xorl    %esi, %esi
100000f0b:  e8 e0 ff ff ff  callq   _greater_than
100000f10:  85 c0   testl   %eax, %eax
100000f12:  74 09   je  0x100000f1d
100000f14:  48 8d 3d 85 00 00 00    leaq    133(%rip), %rdi ## li
teral pool for: "1 is greater than 0"
100000f1b:  eb 07   jmp 0x100000f24
100000f1d:  48 8d 3d 5c 00 00 00    leaq    92(%rip), %rdi ## lit
eral pool for: "1 is not greater than 0"
100000f24:  e8 05 00 00 00  callq   0x100000f2e ## symbol stub fo
r: _puts
100000f29:  31 c0   xorl    %eax, %eax
100000f2b:  5d  popq    %rbp
100000f2c:  c3  retq

Contents of (__TEXT,__cstring) section
0000000100000f50  1 is greater than 0\n
0000000100000f65  1 is not greater than 0\n
0000000100000f7e
0000000100000f7f
0000000100000f80  1 is not greater than 0
0000000100000f98
0000000100000f99
0000000100000f9a
0000000100000f9b
0000000100000f9c
0000000100000f9d
0000000100000f9e
0000000100000f9f
0000000100000fa0  1 is greater than 0
```