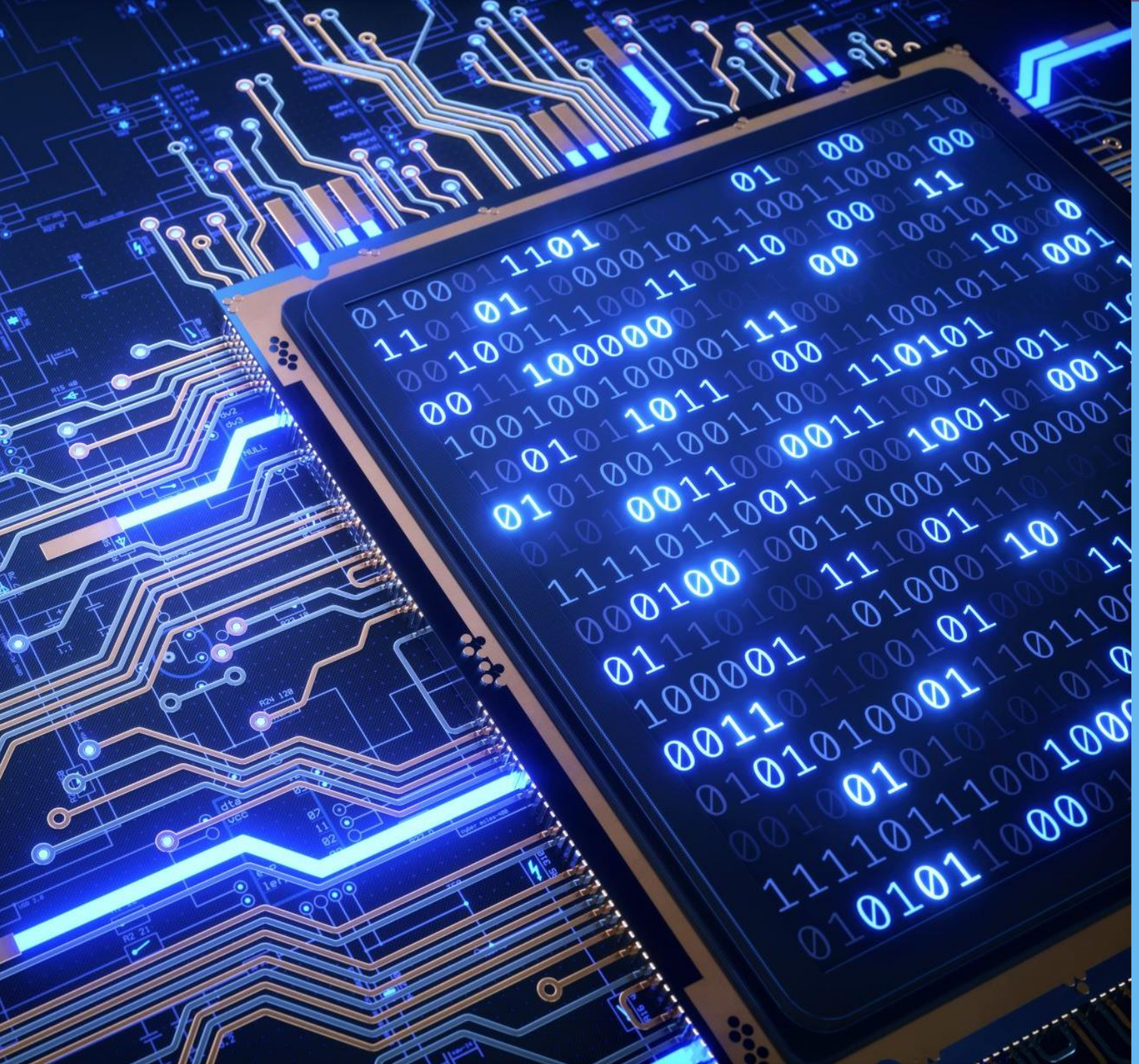


Computer Systems

CS51 Machine

CS51 – Fall 2026

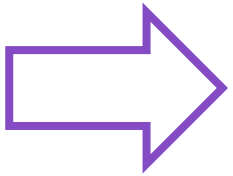
How does a program run on the CPU?

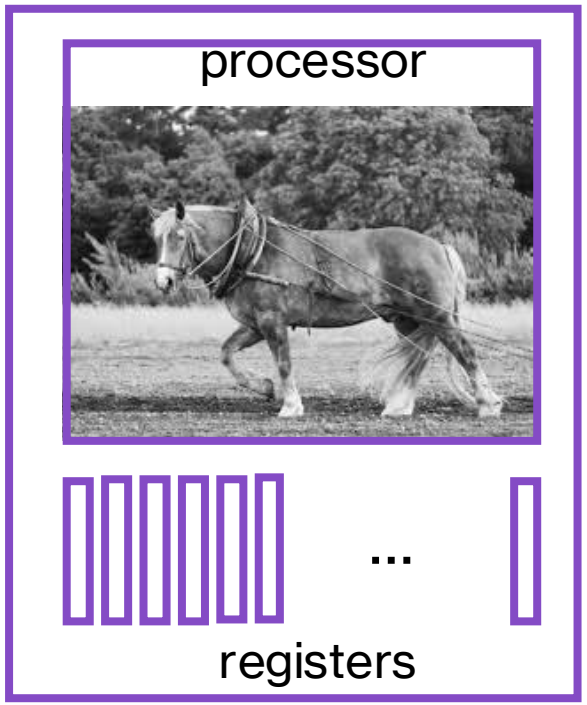
```
def add(num1, num2):
        return num1 + num2

def double(num):
        return 2 * num

def add_then_double(a, b):
        added = add(a, b)
        return double(added)

>> add_then_double(-10, 4)
```





How do programs run/execute on a computer?

Assembly code

- Python is a “high-level” programming language
- high-level programming languages allow you to write code:
  - without worrying about hardware-specific details of the computer (memory, registers, CPU specifics...)
  - higher-level abstraction, e.g., 2**6 or print()
- What actually runs on the processor is **assembly** code

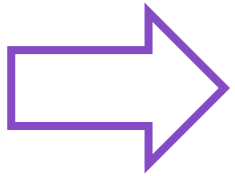
Assembly code

```
def add(num1, num2):
        return num1 + num2

def double(num):
        return 2 * num

def add_then_double(a, b):
        added = add(a, b)
        return double(added)

>> add_then_double(-10, 4)
```



```
add
    psh r2          ; save the return address
    loa r2 r1 4     ; get the second argument
    add r3 r3 r2    ; add r3 = arg1 + arg2
    pop r2          ; get the return address
    jmp r2          ; return

double
    psh r2          ; save the return address
    add r3 r3 r3    ; double the input2
    pop r2          ; get the return address
    jmp r2          ; return

add_then_double
    psh r2          ; save the return address
    loa r2 r1 4     ; get the second argument

    psh r2          ; setup call for add
    lcw r2 add      ; r3 has input one, push input2 on stack
    cal r2 r2       ; call add
    pop r0          ; remove input2 from stack

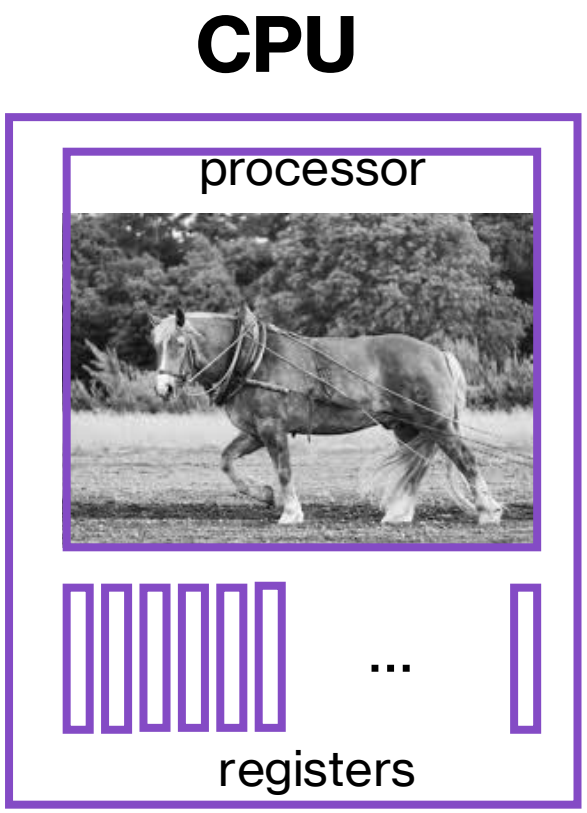
    lcw r2 double   ; result of add is in r3
    cal r2 r2       ; call double

    pop r2          ; get the return address
    jmp r2          ; return
```

This week

- Introduce the CS51 machine
- This is a simplified version of an assembly language
- It is a “simulator” that assumes a very simple CPU and memory setup

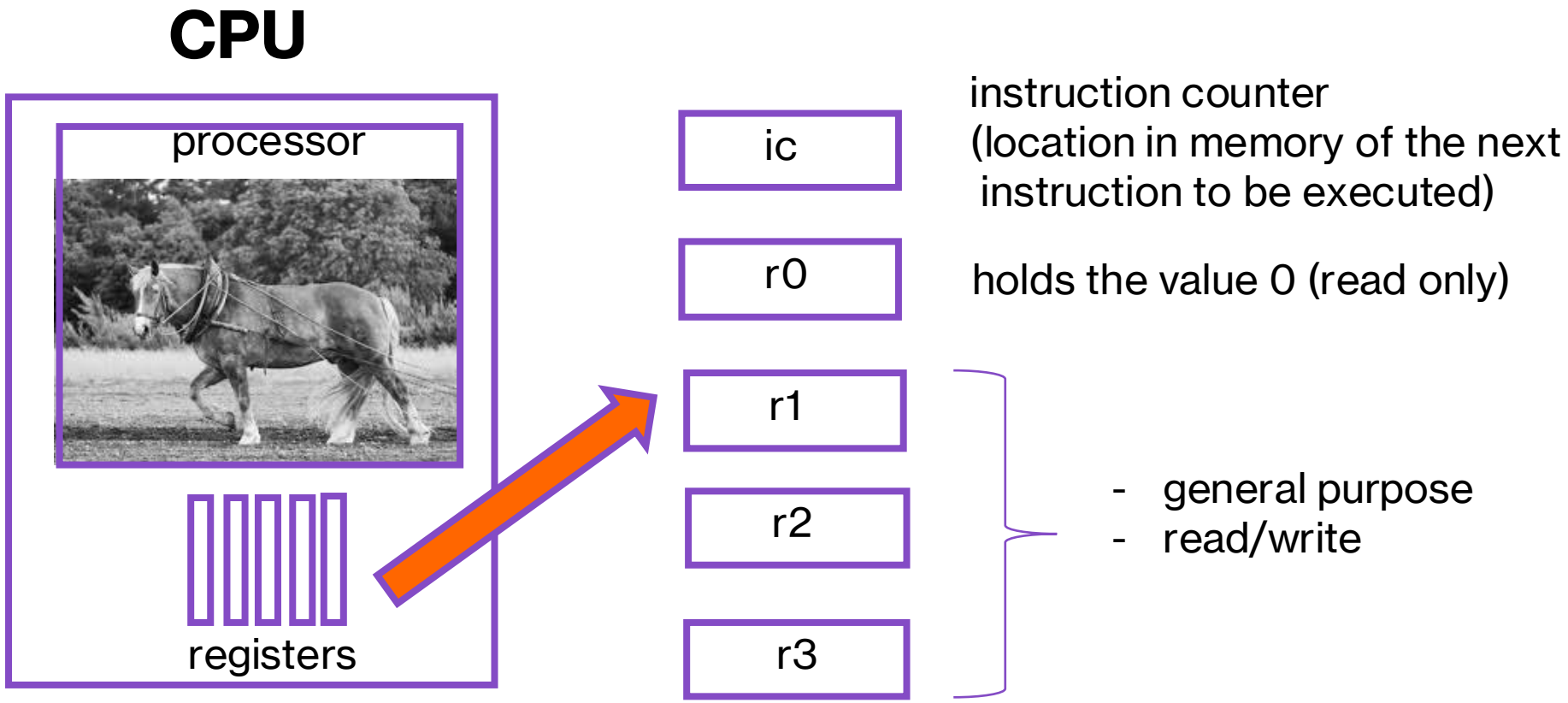
Inside the CPU



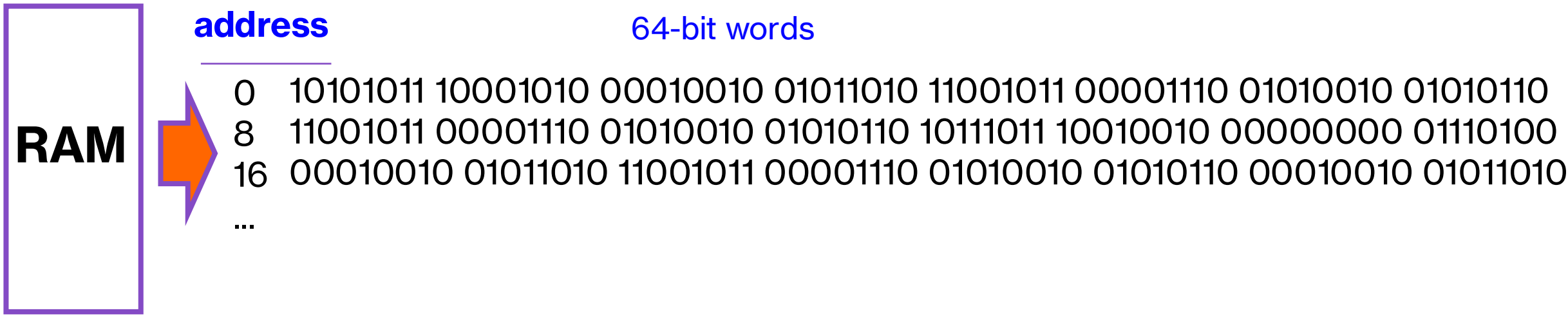
processor: does the work

registers: local, fast memory slots

CS51 machine (processor)

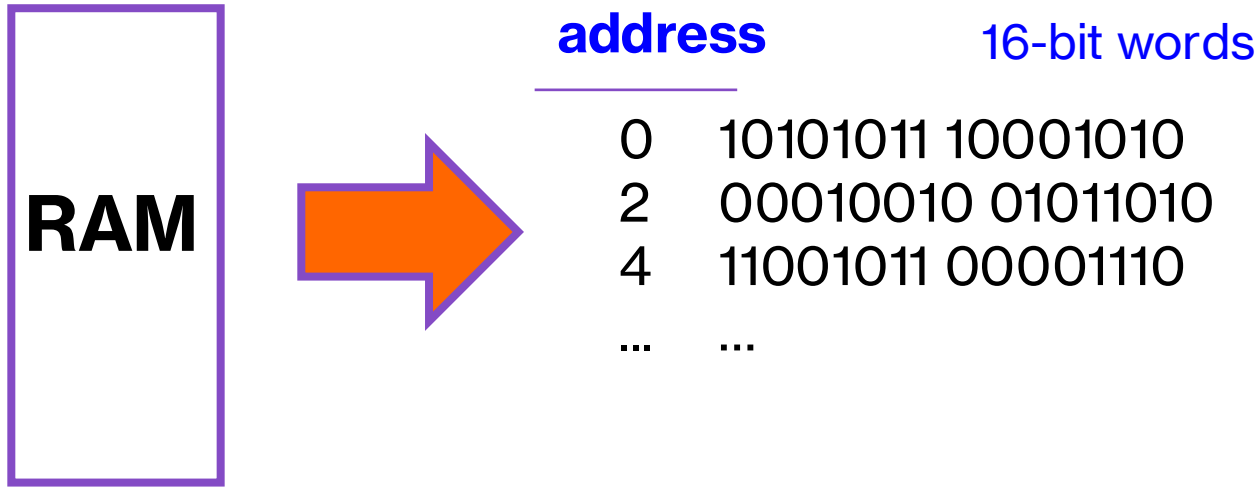


Memory



Most modern computers use 64-bit (8 byte) words

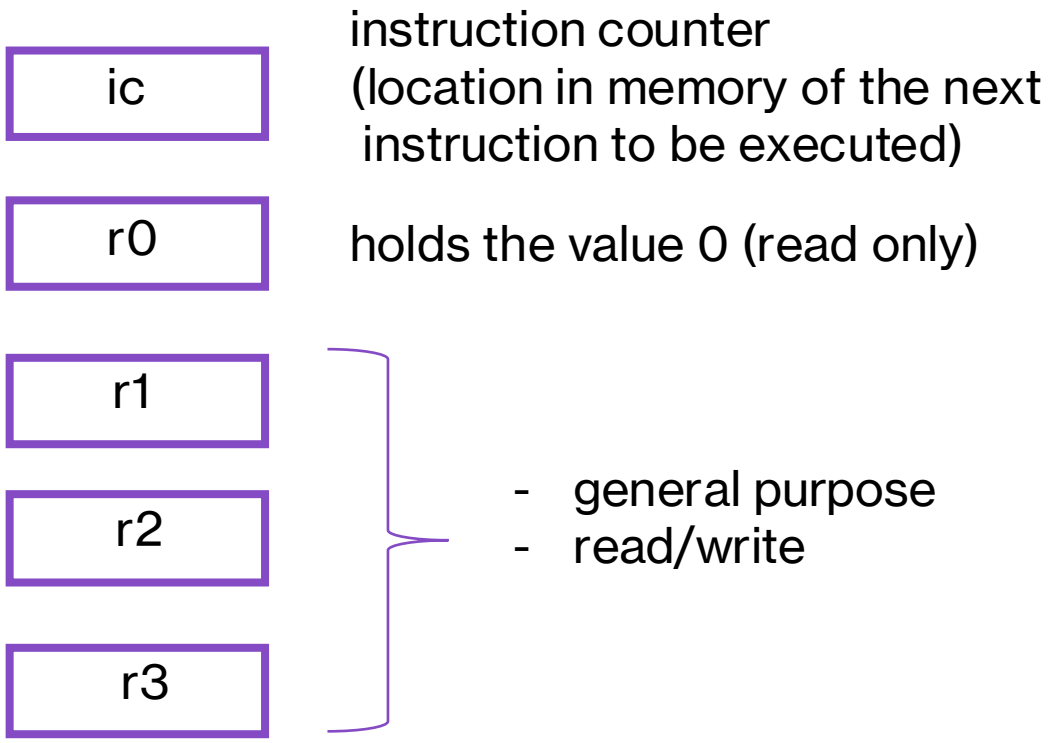
Memory in the CS51 Machine



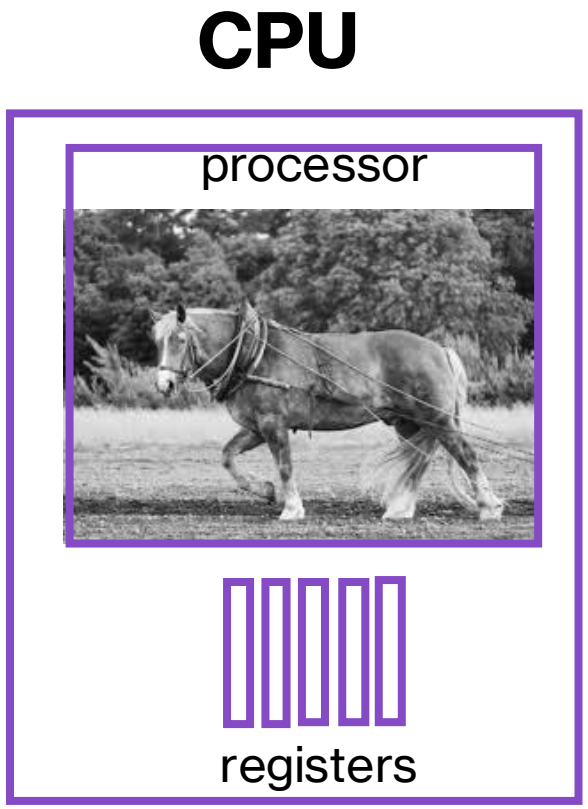
We'll use 16-bit (2-byte) words for our model (the CS51 machine)

Program execution in the CS51 Machine

- When executing a program, the CS51 machine loops over the follow:
  - Fetch the value from $\mathrm{mem}[\mathrm{ic}]$ for use as an instruction
  - Increment ic by 2
  - Decode the instruction and then execute it



Think



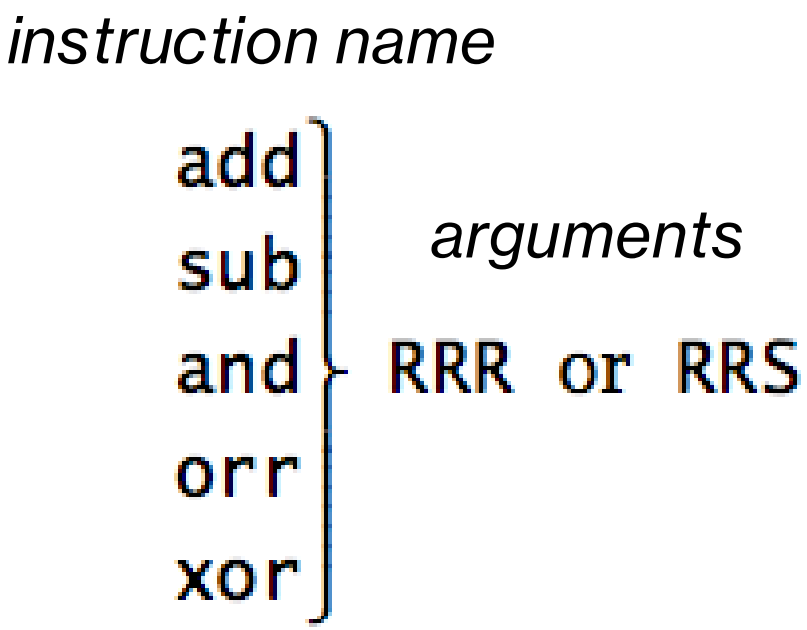
What types of operations might we want to do (think really basic)?

CS51 machine instructions

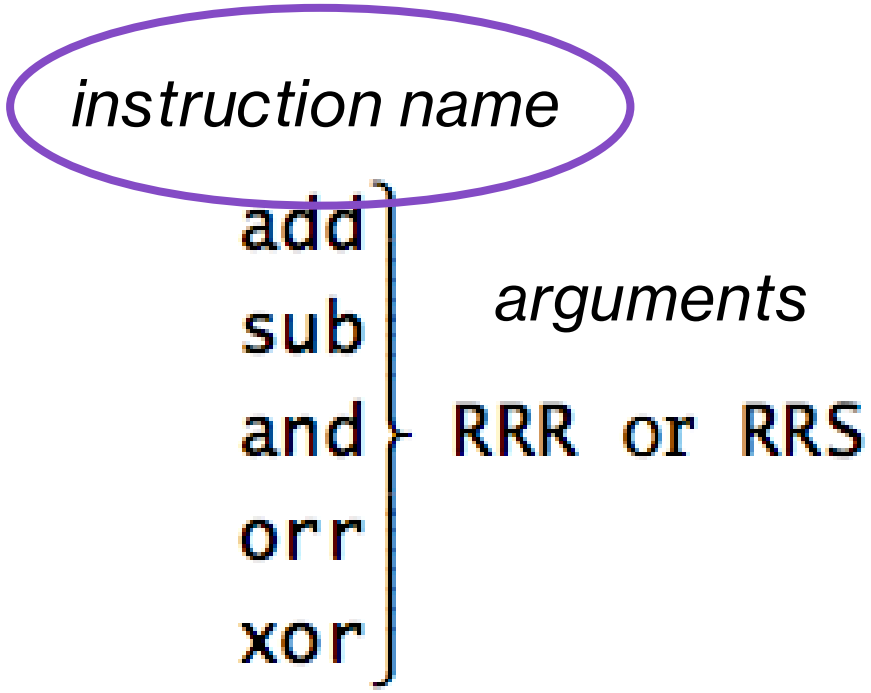
Four main types of instructions

1. math/logical
2. branch (conditionals, loops)
3. memory
4. control the machine (e.g., stop it)

Arithmetic and bitwise logical operations

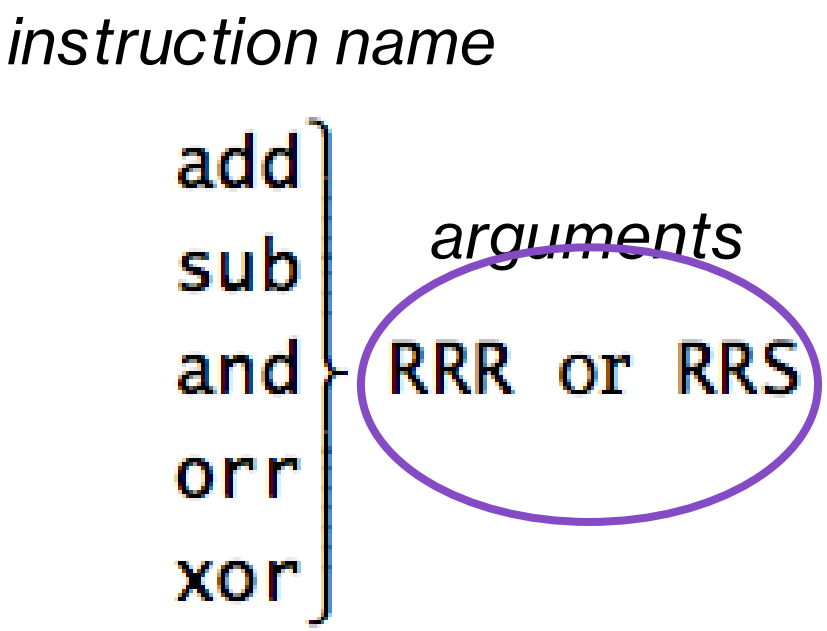


Arithmetic and bitwise logical operations



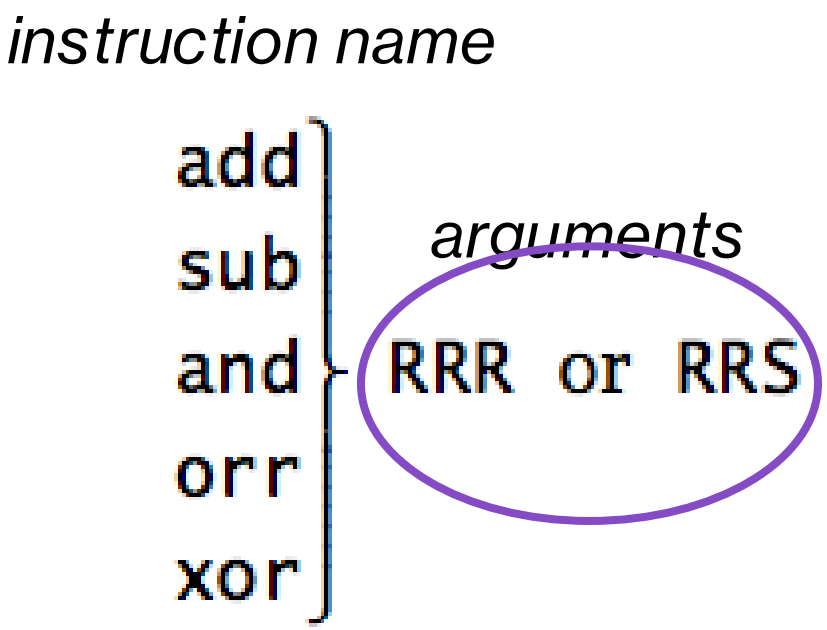
instruction/operation name is always three characters

Arithmetic and bitwise logical operations



R = register (e.g. r0)
S = signed number (byte)

Arithmetic and bitwise logical operations



1st R:	register where the answer will go
2nd R:	register of first operand
3rd S/R:	register/value of second operand

Think

```
add r1 r2 r3
```

What does this do?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Arithmetic and bitwise logical operations

```
add r1 r2 r3
```

r1 = r2 + r3

Add contents of registers r2 and r3 and store the result in r1

1st R:	register where the answer will go
2nd R:	register of first operand
3rd S/R:	register/value of second operand

Think

```
add r2 r1 10
```

What does this do?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Arithmetic and bitwise logical operations

```
add r2 r1 10
```

r2 = r1 + 10

Add 10 to the contents of register r1 and store in r2

1st R:	register where the answer will go
2nd R:	register of first operand
3rd S/R:	register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 8
sub r2 r0 r1
sub r2 r1 r2
```

What number is in r2?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 8
sub r2 r0 r1
sub r2 r1 r2
```

r1	8

What number is in r2?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 8
sub r2 r0 r1
sub r2 r1 r2
```

r1	8
r2	-8

What number is in r2?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 8
sub r2 r0 r1
sub r2 r1 r2
```

r1	8
r2	16

What number is in r2?

16

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 6
and r2 r1 10
add r3 r1 r2
```

What number is in r3?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 6
and r2 r1 10
add r3 r1 r2
```

r1	6

What number is in r3?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 6
and r2 r1 10
add r3 r1 r2
```

r1	6

$6_{10} = 0110$
& $10_{10} = 1010$

What number is in r3?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 6
and r2 r1 10
add r3 r1 r2
```

r1	6
r2	2

$6_{10} = 0110$
& $10_{10} = 1010$

$2_{10} = 0010$

What number is in r3?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Think

Hint: r0 is always 0

```
add r1 r0 6
and r2 r1 10
add r3 r1 r2
```

r3 = r1 + r2

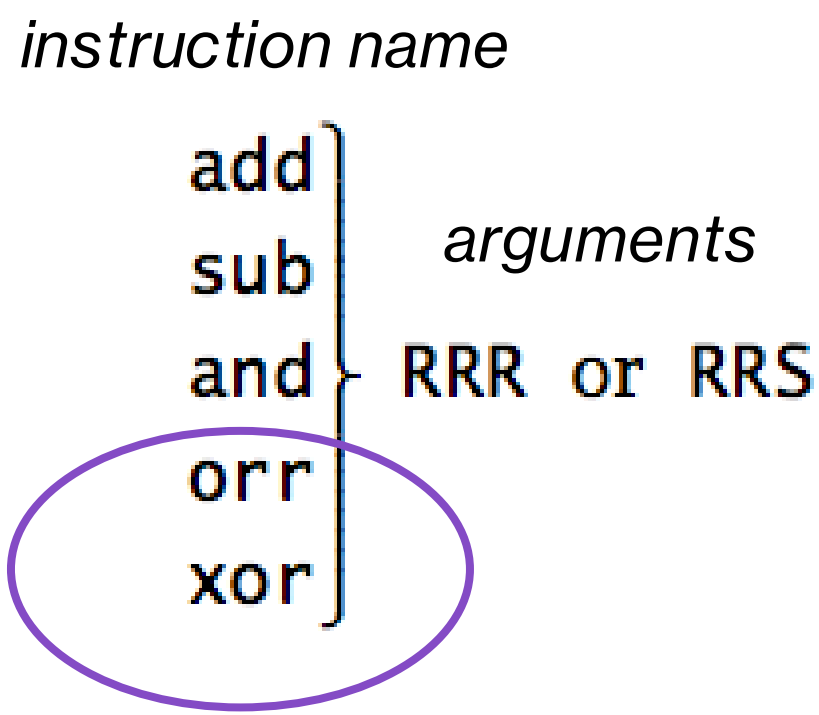
r1	6
r2	2
r3	8

What number is in r3?

8

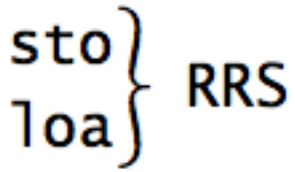
1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Arithmetic and bitwise logical operations



R = register (e.g. r0)
S = signed number (byte)

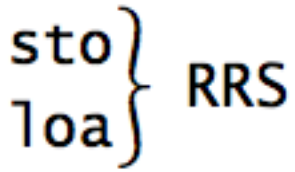
Memory and data-moving instructions



`sto` = save data in register TO memory
`loa` = put data FROM memory into a register

```
sto r1 r2   ; store the contents of r1 to mem[r2]
loa r1 r2   ; get data from mem[r2] and put into r1
```

Special cases for memory access



`sto` = save data in register TO memory
`loa` = put data FROM memory into a register

- saving TO (sto) address 0 (r0) prints, e.g., `sto r1 r0`
- loading from (loa) address 0 (r0) gets input from user, e.g., `loa r1 r0` reads number from user and stores it in r1

Basic structure of CS51 program

```
; great comments at the top!
;
        instruction1            ; comment
        instruction2            ; comment
        ...
label1
        instruction3            ; comment
        instruction4            ; comment
label2
        ...
        hlt
```



- whitespace before operations/instructions
- labels go here

subtract.a51

```
; A simple CS51 Machine program that subtracts
; two numbers.

        loa r2 r0           ; get first value
        loa r3 r0           ; get second value
        sub r2 r2 r3        ; subtract them
        sto r2 r0           ; print result
        hlt                 ; quit
```

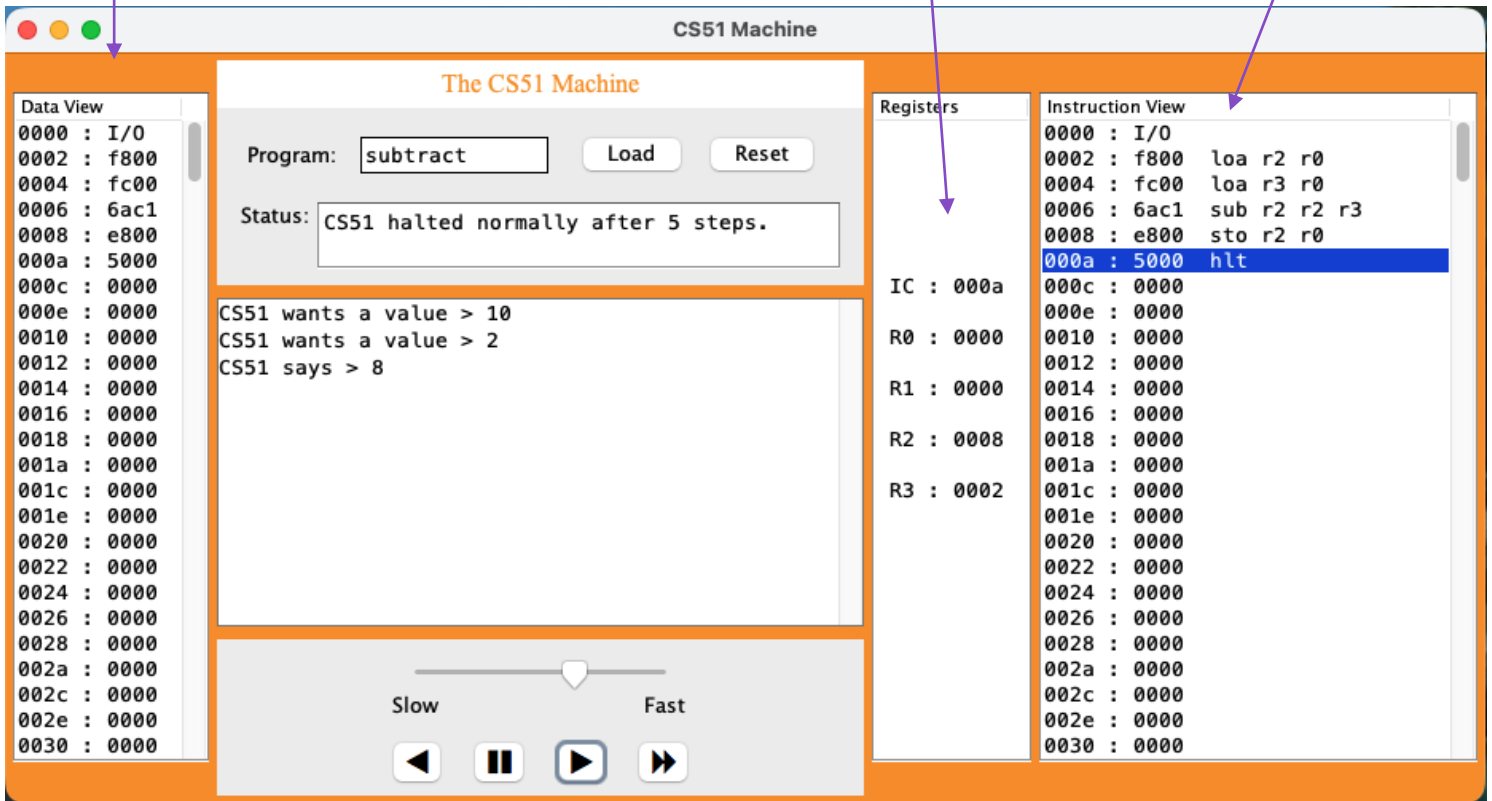
CS51 simulator

memory

instruction execution

registers

I/O and running program



Control instructions

branch (always)	`brs`	`B`
branch if ==	`beq`	`RRB`
branch if !=	`bne`	
branch if <	`blt`	
branch if >=	`bge`	
branch if >	`bgt`	
branch if <=	`ble`	

1st R: first register for comparison

2nd R: second register in comparison

3rd B: label

Think

```
beq r3 r0 done
```

What does this do?

1st R:	first register for comparison
2nd R:	second register in comparison
3rd B:	label

Think

```
beq r3 r0 done
```

If r3 == 0, branch to the label “done”
if not (else) ic is incremented as normal to the next instruction

1st R:	first register for comparison
2nd R:	second register in comparison
3rd B:	label

Think

```
ble r2 r3 done
```

What does this do?

1st R:	first register for comparison
2nd R:	second register in comparison
3rd B:	label

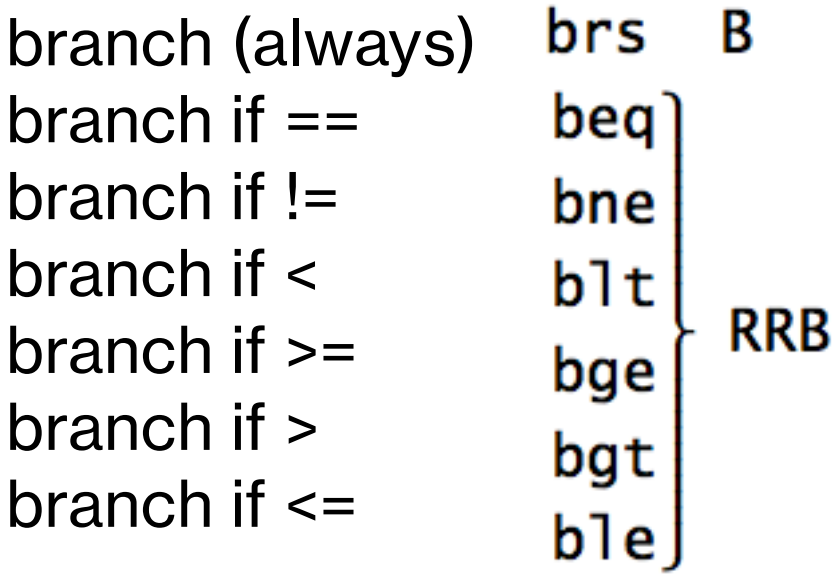
Think

```
ble r2 r3 done
```

If r2 <= r3, branch to the label “done”
if not (else) ic is incremented as normal to the next instruction

1st R:	first register for comparison
2nd R:	second register in comparison
3rd B:	label

Control instructions



- conditionals
- loops
- change the order that instructions are executed

More CS51 machine examples

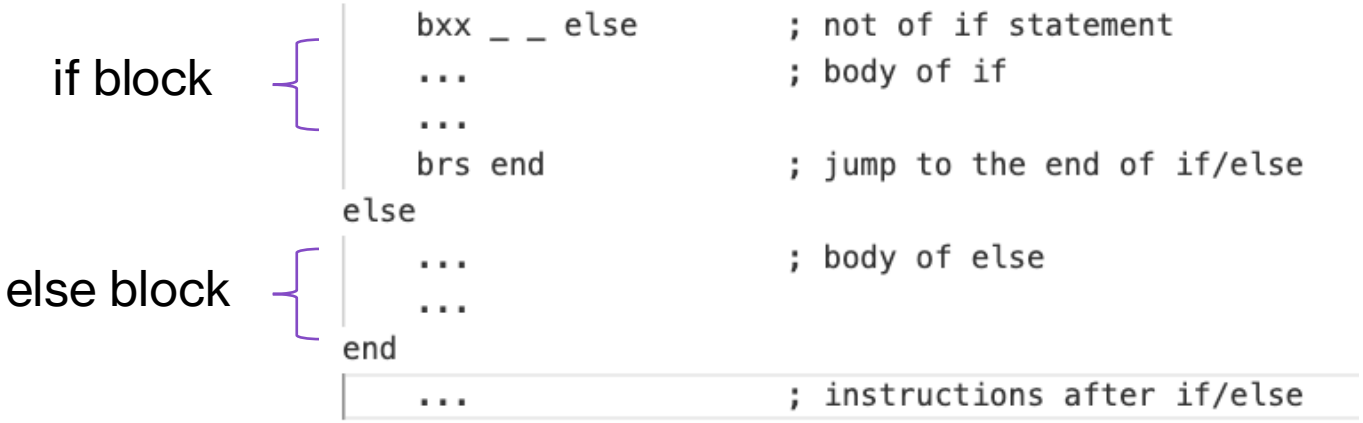
- Look at max_simple.a51
  - Gets two values from the user
  - Compares them
  - Uses a branch to distinguish between the two cases
  - Goal is to get largest value in r3
  - print largest value

simple_max.a51

```
;
; simple program to compute the max of
; two numbers
;
        loa r2 r0       ; get the first value and put it in r2
        loa r3 r0       ; get the second value and put it in r3

        bge r3 r2 done  ; check if r3 >= r2, if so jump to done
        add r3 r2 0     ; r3 = r2, (r2 is larger so copy it)
done
        sto r3 r0
        hlt
```

if/else



- check the opposite of the if statement
  - if it is true, we'll jump down to else
  - if it is not true, we'll continue into the body of the if part
- At the end of the if block, need to jump to the end, otherwise, we'd continue onto else

Think

```
    loa r3 r0

    and r2 r3 1
    beq r2 r0 else
    add r3 r0 47
    brs end
else
    add r3 r0 -47
end

    sto r3 r0
    hlt
```

What does this code do?

Think

```
    loa r3 r0

    and r2 r3 1
    beq r2 r0 else
    add r3 r0 47
    brs end
else
    add r3 r0 -47
end

    sto r3 r0
    hlt
```

if block

else block

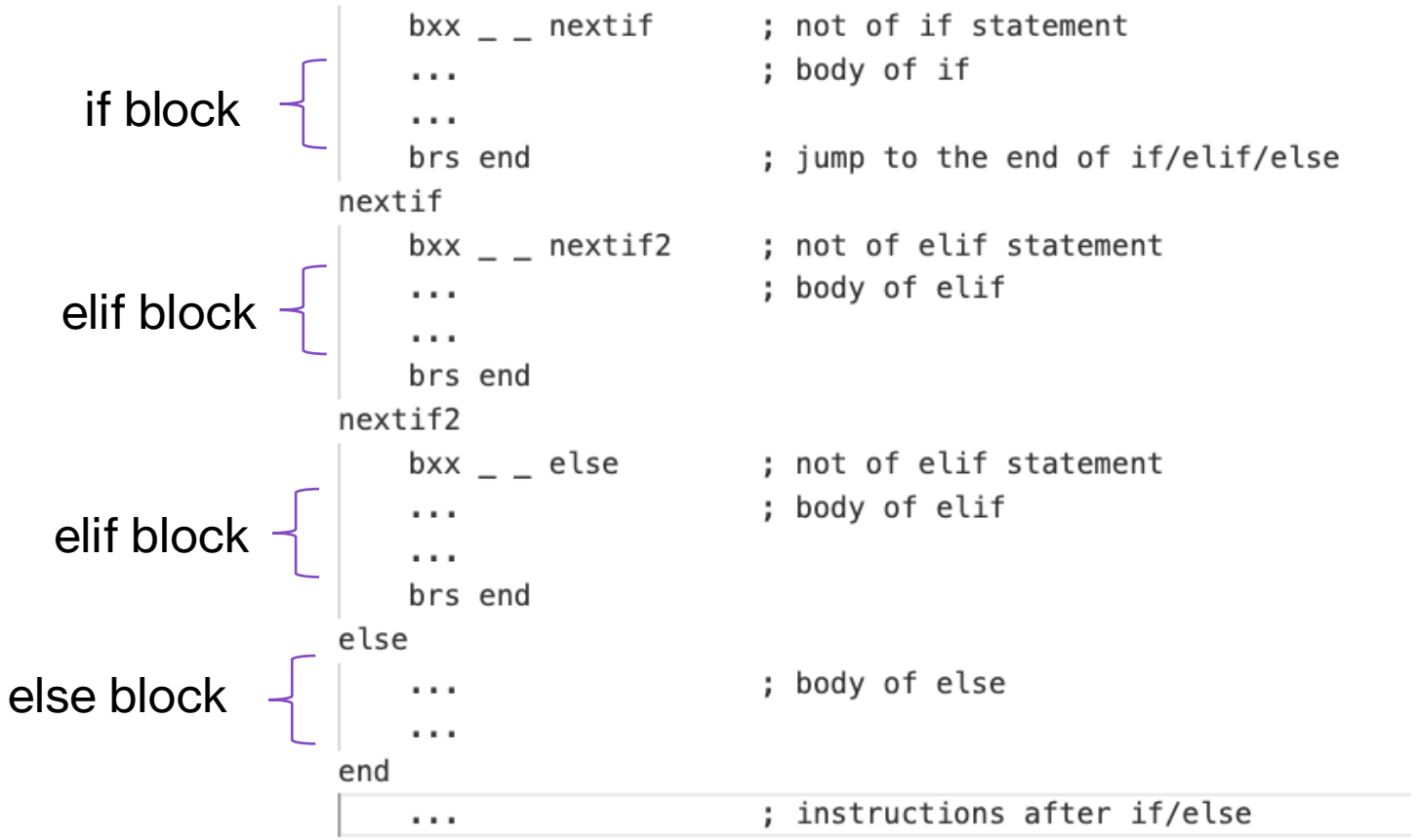
if/else (even_commented.a51)

```
    loa r3 r0           ; get a value from the user

    and r2 r3 1         ; get the low-order bit into r2
    beq r2 r0 else      ; branch to else if even
    add r3 r0 47        ; put 47 in r3
    brs end             ; go to the end of the if/else
else
    add r3 r0 -47       ; put -47 in r3
end

    sto r3 r0           ; print out r3
    hlt
```

If/elif/else


if block
elif block
elif block
else block
bxx _ _ nextif ; not of if statement
... ; body of if
...
brs end ; jump to the end of if/elif/else
nextif
bxx _ _ nextif2 ; not of elif statement
... ; body of elif
...
brs end
nextif2
bxx _ _ else ; not of elif statement
... ; body of elif
...
brs end
else
... ; body of else
...
end
... ; instructions after if/else

Think

```
    loa r3 r0

    bge r3 r0 nextif
    add r3 r0 -1
    brs end
nextif
    bgt r3 r0 else
    add r3 r0 0
    brs end
else
    add r3 r0 1
end

    sto r3 r0
    hlt
```

What does this code do?

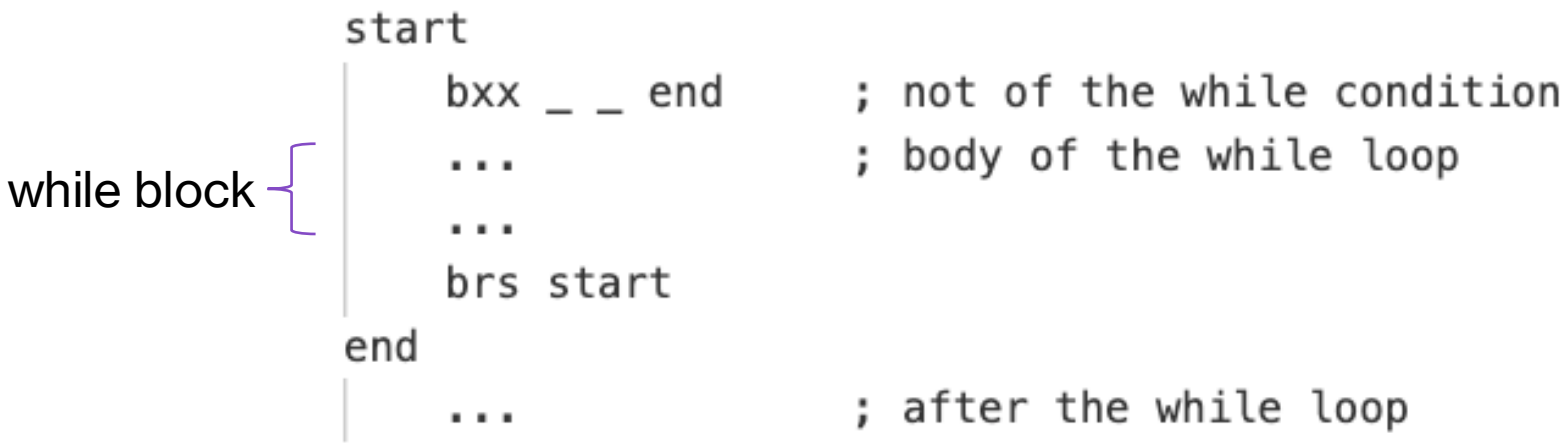
if/elif/else (sign_commented.a51)

```
    loa r3 r0                   ; get a number from the user

    bge r3 r0 nextif            ; if r3 < 0
    add r3 r0 -1                ; r3 = -1
    brs end
nextif
    bgt r3 r0 else              ; if r3 <= 0
    add r3 r0 0                 ; r3 = 0
    brs end
else
    add r3 r0 1                 ; r3 is positive: r3 = 1
end

    sto r3 r0                   ; print out r3
    hlt
```

while loop


while block
start
bxx _ _ end ; not of the while condition
... ; body of the while loop
...
brs start
end
... ; after the while loop

Think

```
    loa r3 r0

    add r2 r0 0
start
    ble r3 r0 end
    add r2 r2 r3
    sub r3 r3 1
    brs start
end

    sto r2 r0
    hlt
```

What does this code do?

while loop (sum_commented.a51)

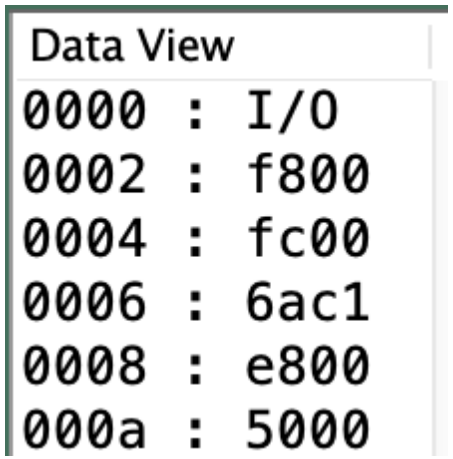
```
    loa r3 r0           ; get a number from the user

    add r2 r0 0         ; r2 = 0
start
    ble r3 r0 end       ; while r3 > 0
    add r2 r2 r3        ; r2 += r3
    sub r3 r3 1         ; r3 -= 1
    brs start
end

    sto r2 r0           ; print out r2
    hlt
```

Instructions to binary

CS51 Machine uses 16-bit words



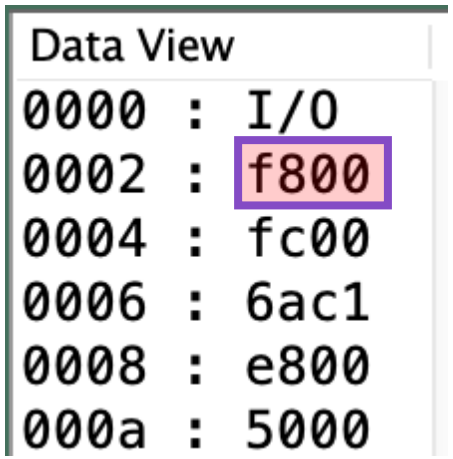
This is my assembly program

16-bit value
(displayed as hexadecimal number)

memory address

Instructions to binary

CS51 Machine uses 16-bit words



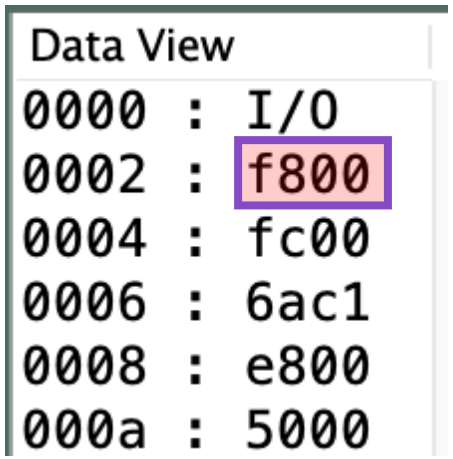
What binary number is this?

16-bit value
(displayed as hexadecimal number)

memory address

Instructions to binary

CS51 Machine uses 16-bit words



What binary number is this?

15 8 0 0

1111 1000 0000 0000

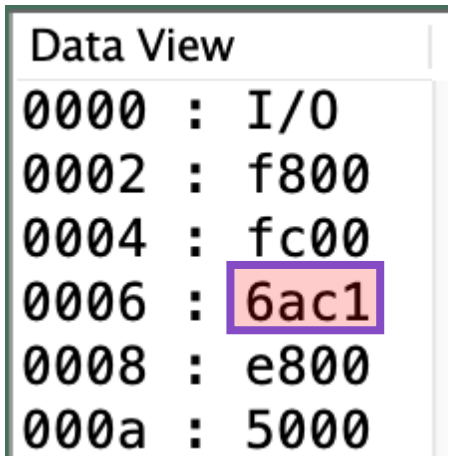
16 bits

16-bit value
(displayed as hexadecimal number)

memory address

Instructions to binary

CS51 Machine uses 16-bit words



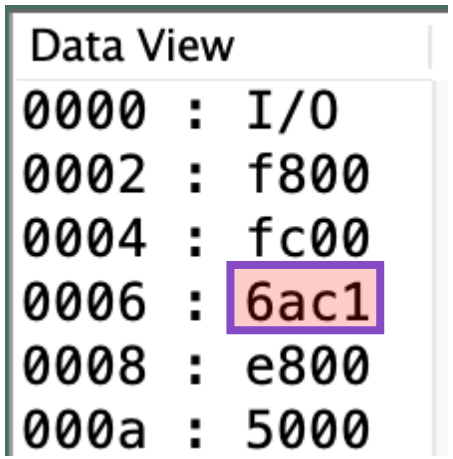
What binary number is this?

16-bit value
(displayed as hexadecimal number)

memory address

Instructions to binary

CS51 Machine uses 16-bit words



What binary number is this?

6 10 12 1

0110 1010 1100 0001

16-bit value
(displayed as hexadecimal number)

memory address

Encoding instructions

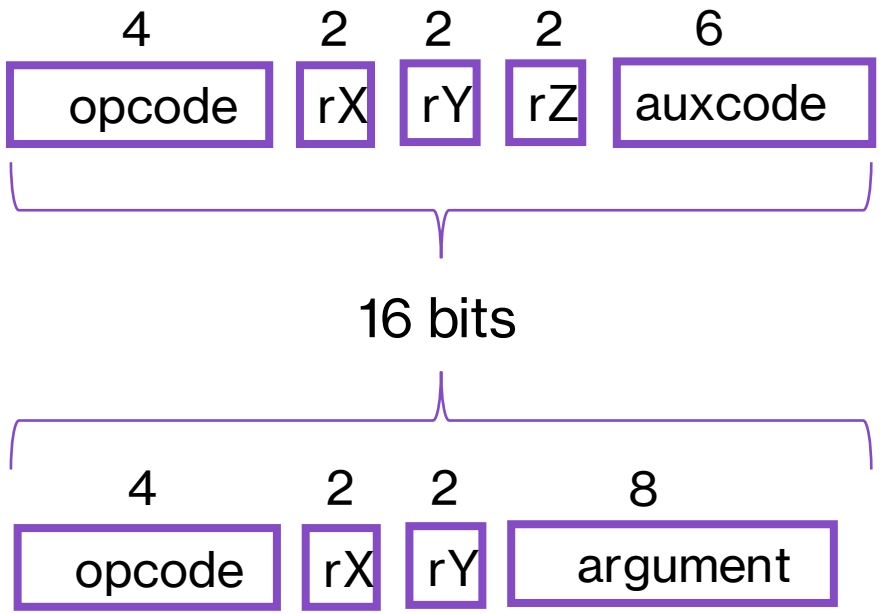
Two formats for instructions

opcode: specifies what operation (or category of operation)

r_: specifies a register

auxcode: specifies additional operations

argument: a number



opcode

opcode	instruction
0x0	beq
0x1	bne
0x2	blt
0x3	bge
0x4	cal
0x5	hlt
0x6	arithmetic/logical instruction
...	
0xe	sto
0xf	loa

Instructions to binary

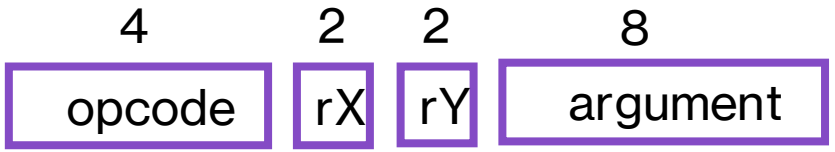
Data View

```
0000 : I/O
0002 : f800
0004 : fc00
0006 : 6ac1
0008 : e800
000a : 5000
```

What binary number is this?

15 8 0 0

1111 1000 0000 0000



Instructions to binary

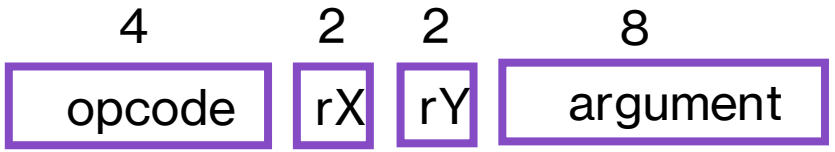
Data View

```
0000 : I/O
0002 : f800
0004 : fc00
0006 : 6ac1
0008 : e800
000a : 5000
```

What is this instruction?

15 8 0 0

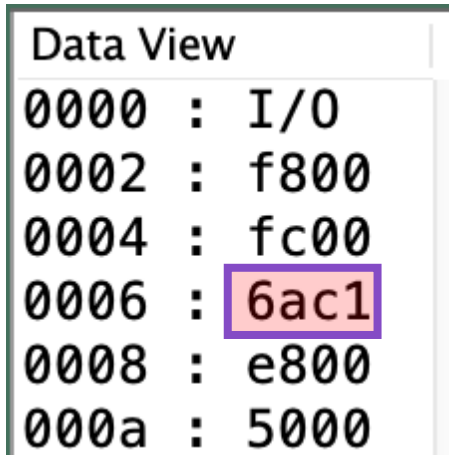
1111 1000 0000 0000



loa r2 r0

opcode	instruction
0x0	beq
0x1	bne
0x2	blt
0x3	bge
0x4	cal
0x5	hlt
0x6	arithmeti/logical instruction
...	
0xe	sto
0xf	loa

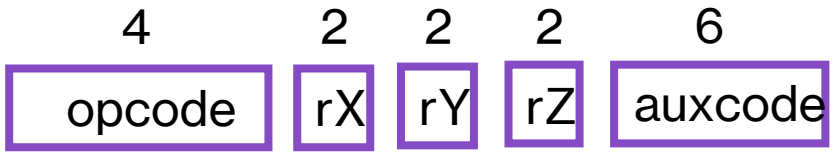
Instructions to binary



What is this instruction?

6 10 12 1

0110 1010 1100 0001



opcode	instruction
0x0	beq
0x1	bne
0x2	blt
0x3	bge
0x4	cal
0x5	hlt
0x6	arithmetic/logical instruction
...	
0xe	sto
0xf	loa

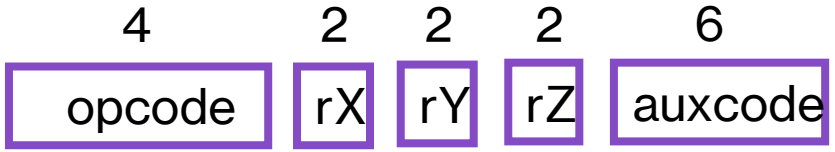
Instructions to binary

```
Data View
0000 : I/O
0002 : f800
0004 : fc00
0006 : 6ac1
0008 : e800
000a : 5000
```

What is this instruction?

6 10 12 1

0110 1010 1100 0001



arithmetic r2 r2 r3 0x1

opcode	instruction
0x0	beq
0x1	bne
0x2	blt
0x3	bge
0x4	cal
0x5	hlt
0x6	arithmetic/logical instruction
...	
0xe	sto
0xf	loa

arithmetic/logical auxcode

auxcode	instruction
0x0	add
0x1	sub
0x2	
0x3	
0x4	and
0x5	orr
0x6	
0x7	
0x8	logical shift left
0x9	logical shift right
...	

Instructions to binary

Data View

```
0000 : I/O
0002 : f800
0004 : fc00
0006 : 6ac1
0008 : e800
000a : 5000
```

What is this instruction?

6 10 12 1

0110 1010 1100 0001

4	2	2	2	6
opcode	rX	rY	rZ	auxcode

sub r2 r2 r3

opcode	instruction
0x0	add
0x1	sub
0x2	
0x3	
0x4	and
0x5	orr
0x6	
0x7	
0x8	logical shift left
0x9	logical shift right
...	

instructions to binary

Data View

```
0000 : I/O
0002 : f800
0004 : fc00
0006 : 6ac1
0008 : e800
000a : 5000
```

Instruction View

```
0000 : I/O
0002 : f800   loa r2 r0
0004 : fc00   loa r3 r0
0006 : 6ac1   sub r2 r2 r3
0008 : e800   sto r2 r0
000a : 5000   hlt
```

Examples from this lecture

http://www.cs.pomona.edu/classes/cs51/cs51machine/