# lecture 8

# MIPS assembly language 1

- what is an assembly language?
- addressing and Memory
- instruction formats (R, I, J)

February 3, 2016

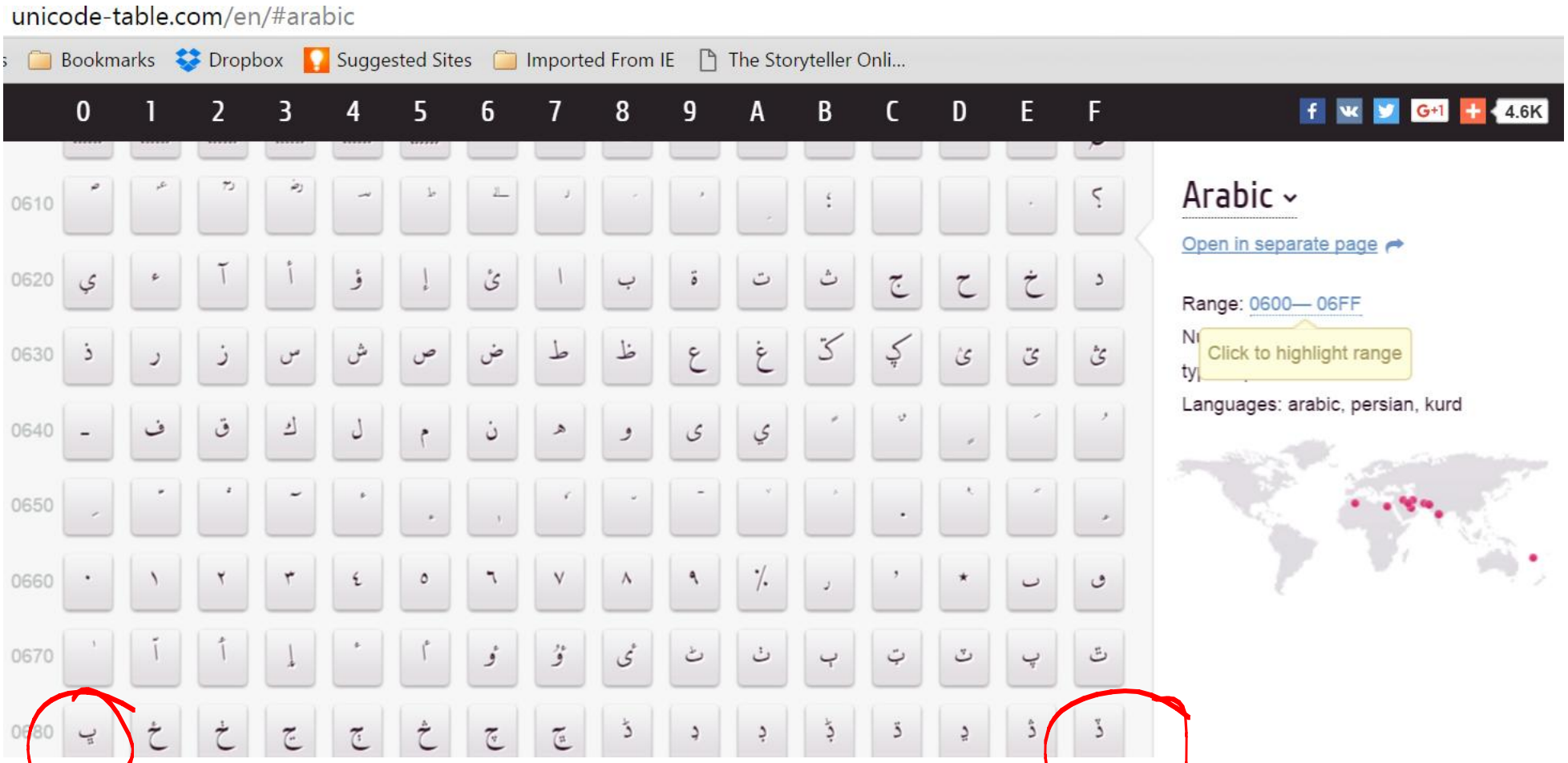
http://www.asciitable.com/

# ASCII: 8 bit (one byte) code

In fact, it only uses 7 of the 8 bits.

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Unicode is a 16 bit code which includes characters for most written natural languages. *[Correction: It is more complicated than that.]* https://en.wikipedia.org/wiki/Unicode



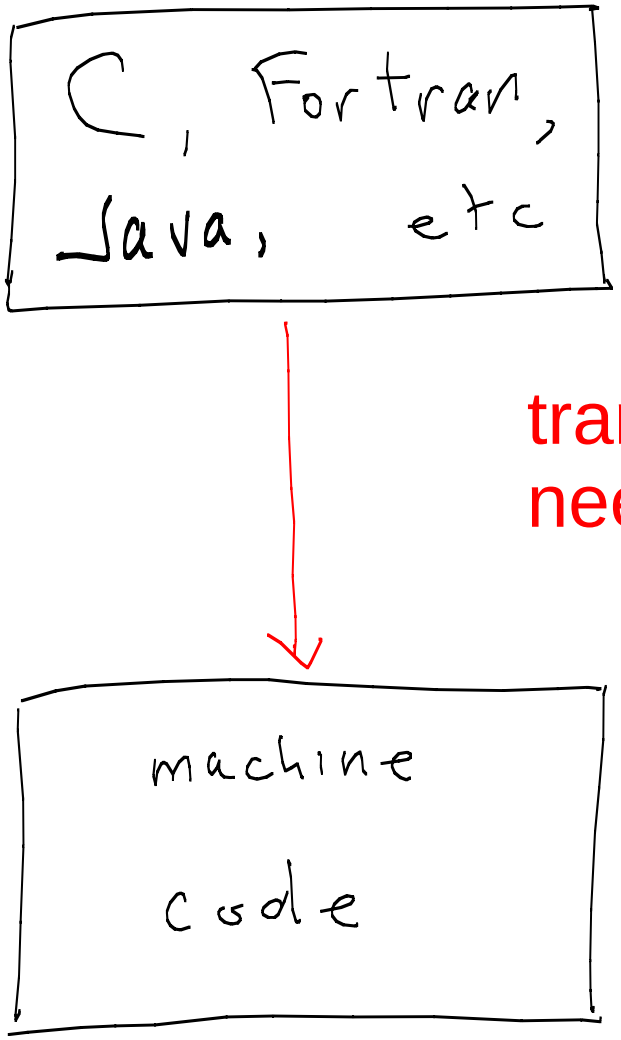

0x0680

0x068f

- written and read by humans

- not executable (ASCII)

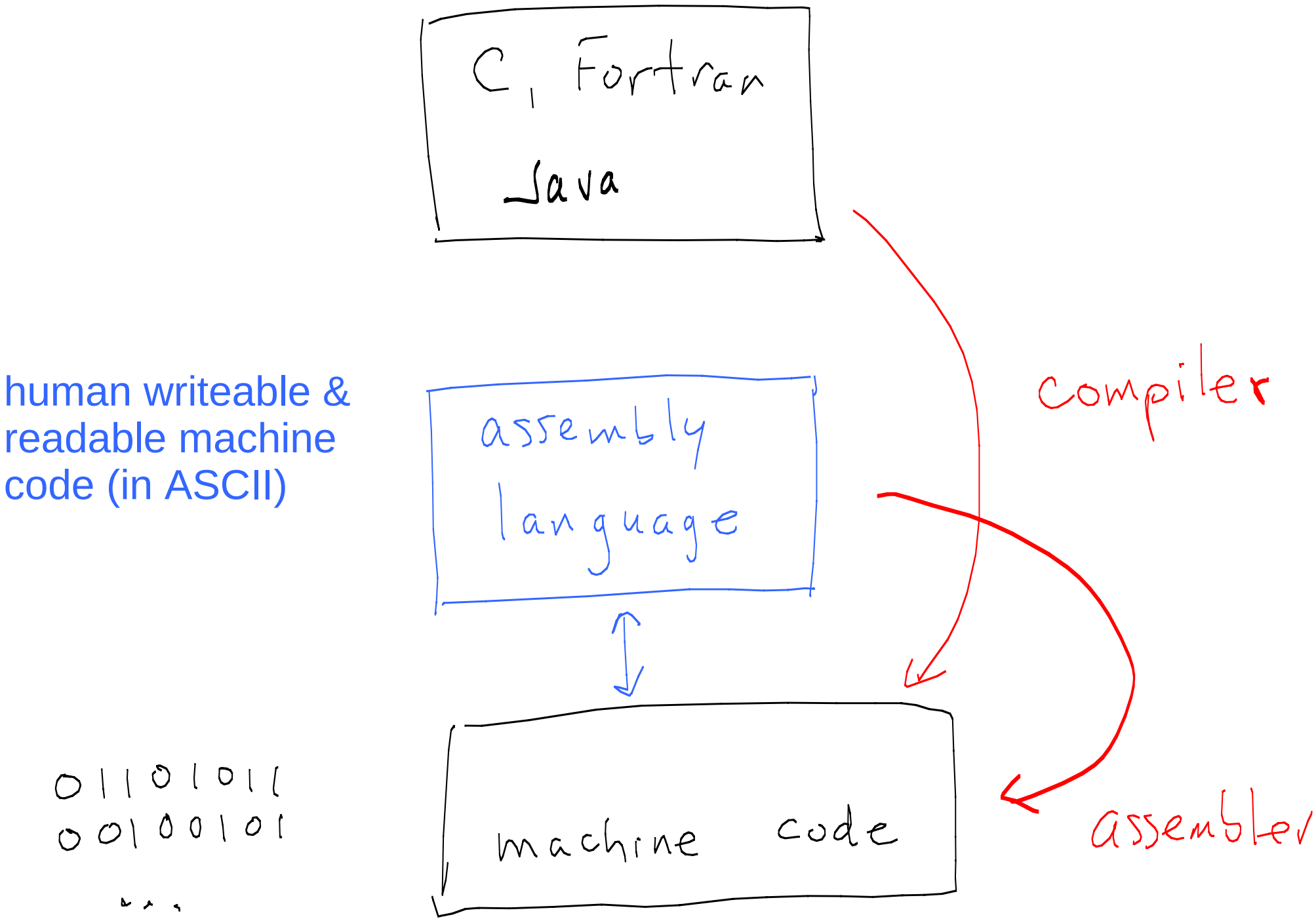
- not machine specific



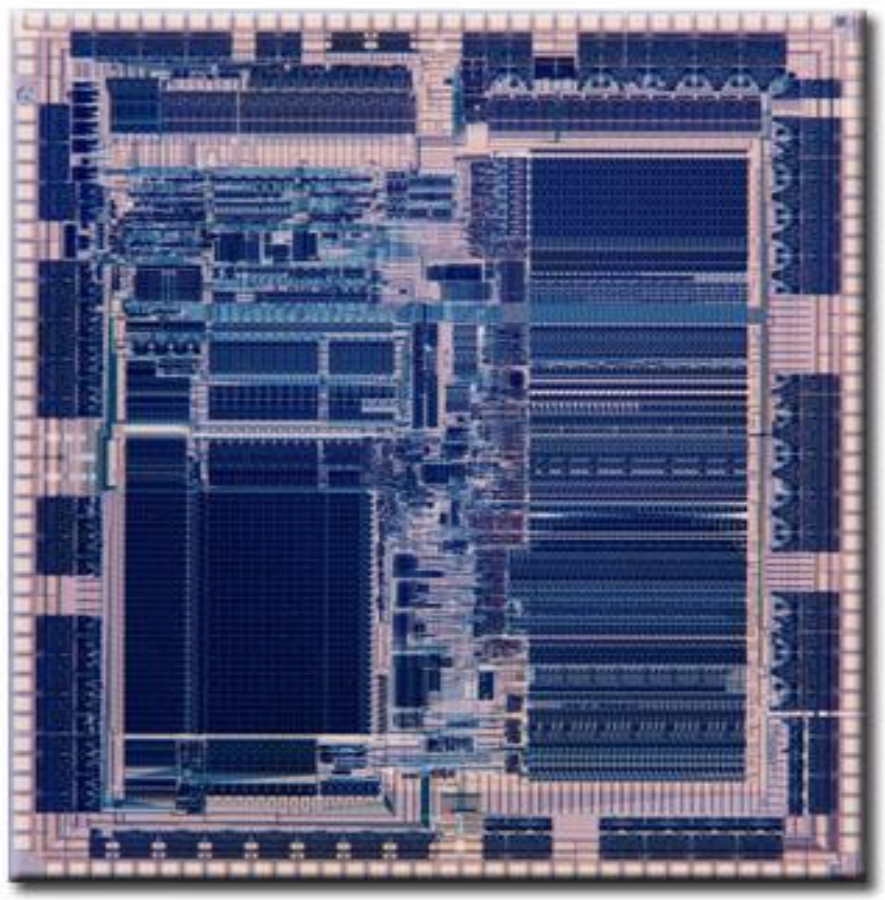

- executable

- machine specific

(...although what we mean by "machine" here is subtle.
e.g. Java Virtual Machine.


C, Fortran
Java
human writeable &
readable machine
code (in ASCII)
assembly
language
compiler
0 1 1 0 1 0 1 1
0 0 1 0 0 1 0 1
...
machine code
assembler

# MIPS R2000 CPU (1985)



~ 1 cm

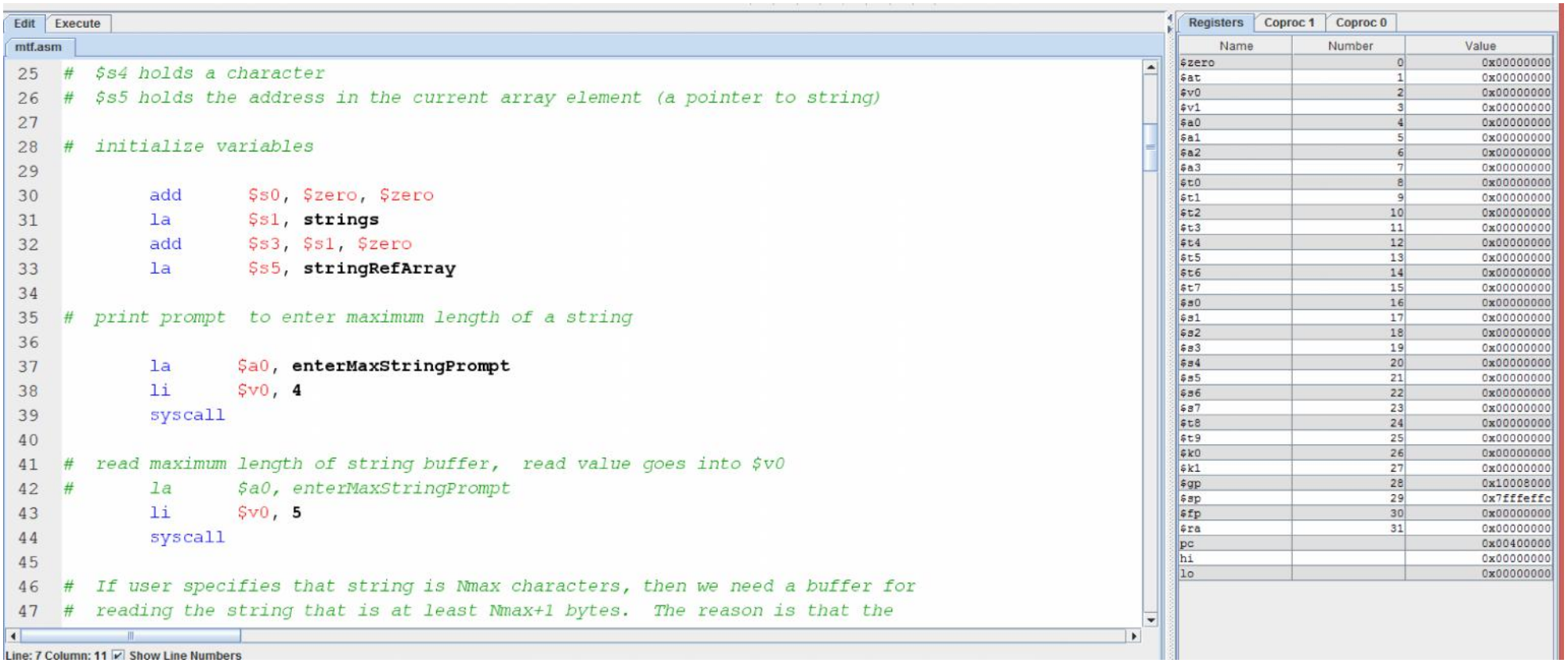
"Reduced Instruction Set Computer" (RISC)

MIPS is not so different from today's ARM processors e.g. in your cell phone or tablet.

# MARS simulator

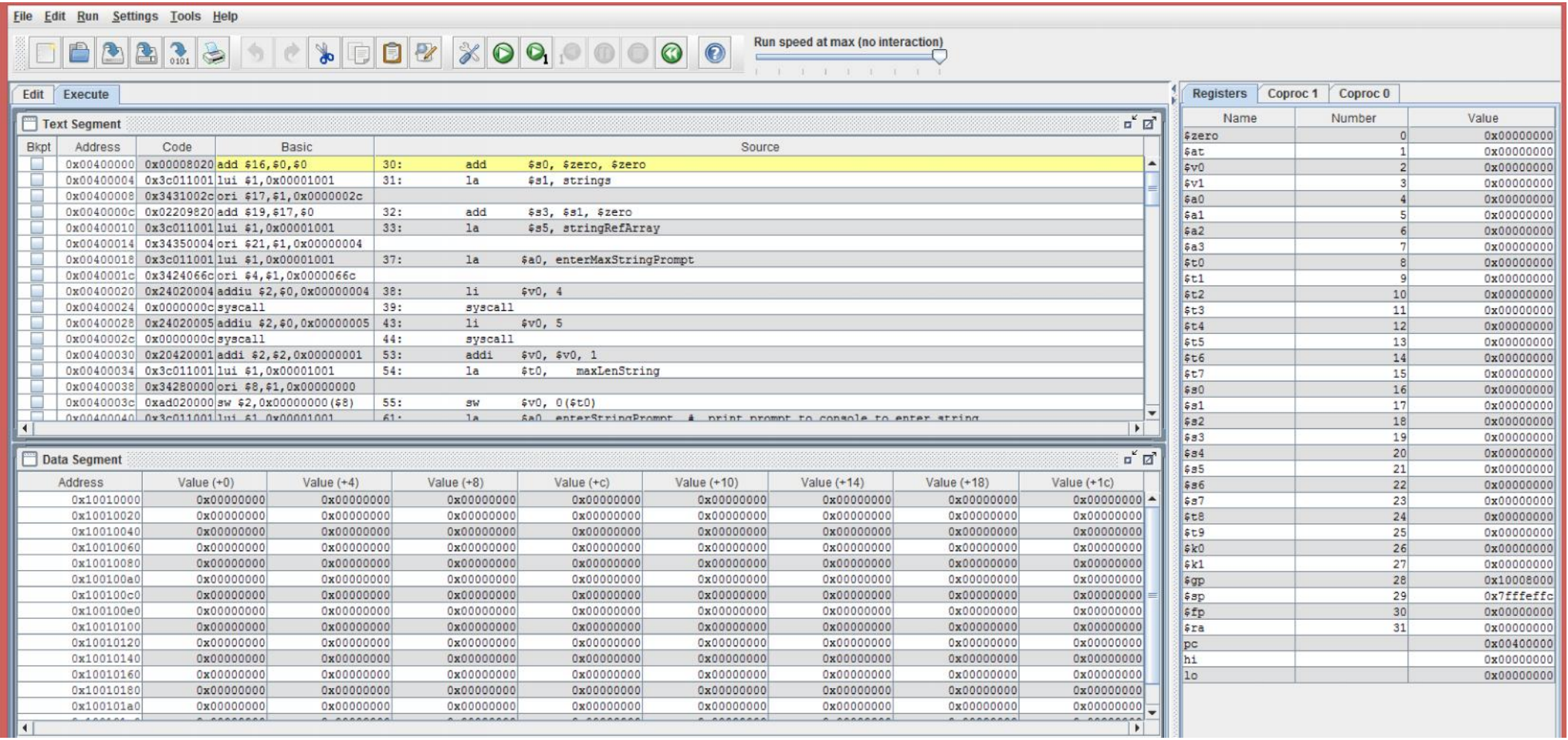
http://courses.missouristate.edu/KenVollmar/MARS/index.htm

## Edit mode



# MARS simulator

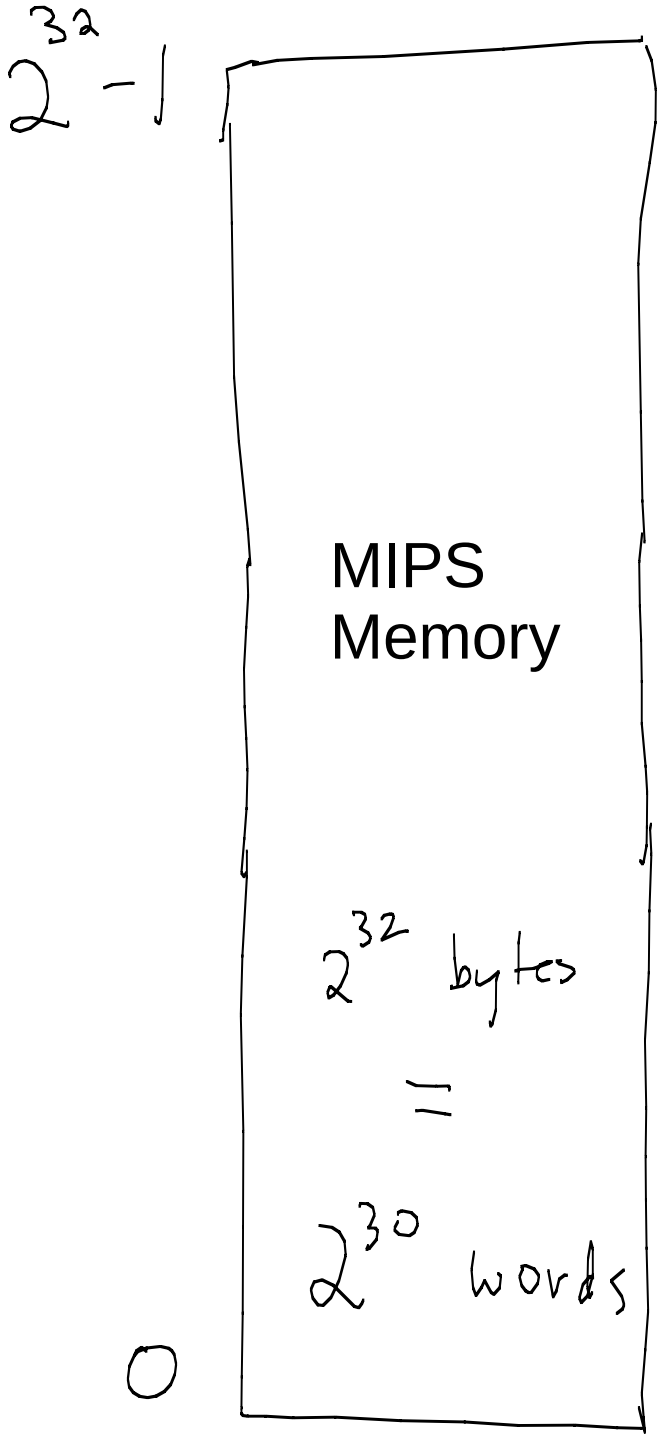
## Execute mode

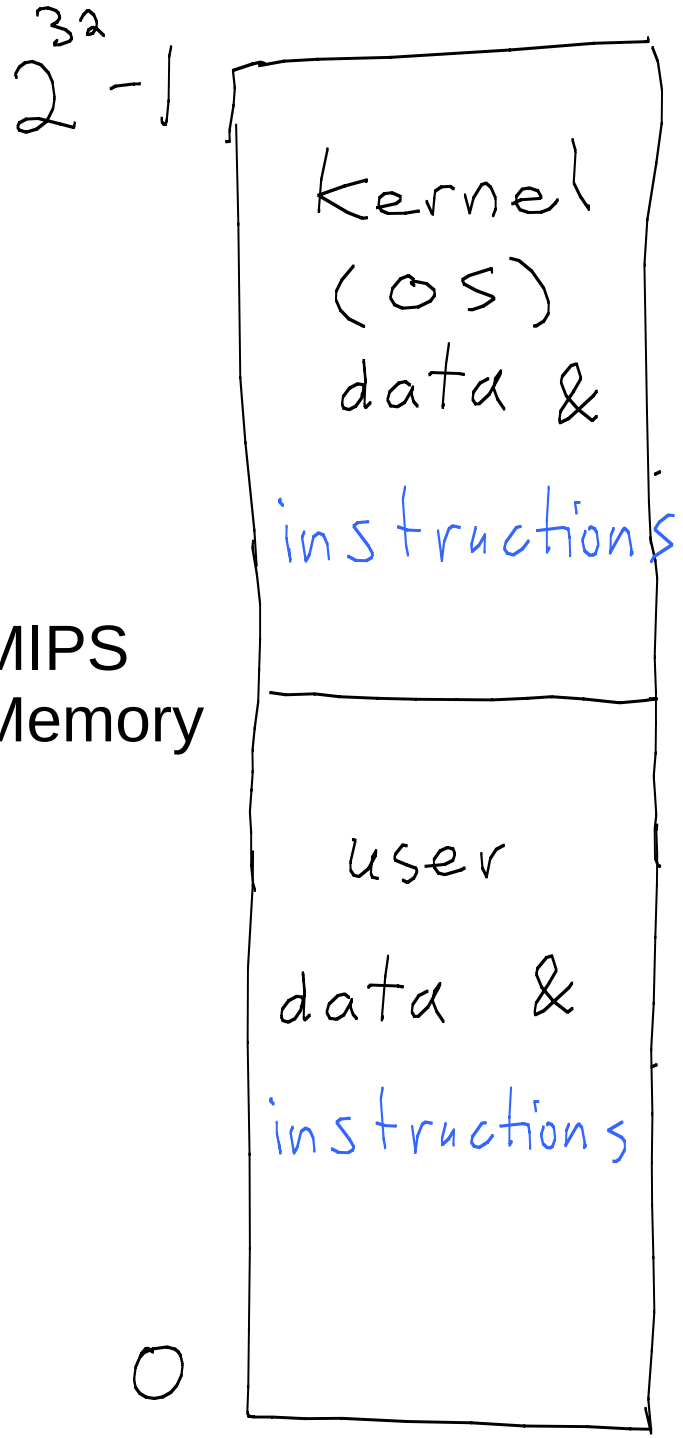


# Addressing in MIPS


1 word = 4 bytes

= 32 bits



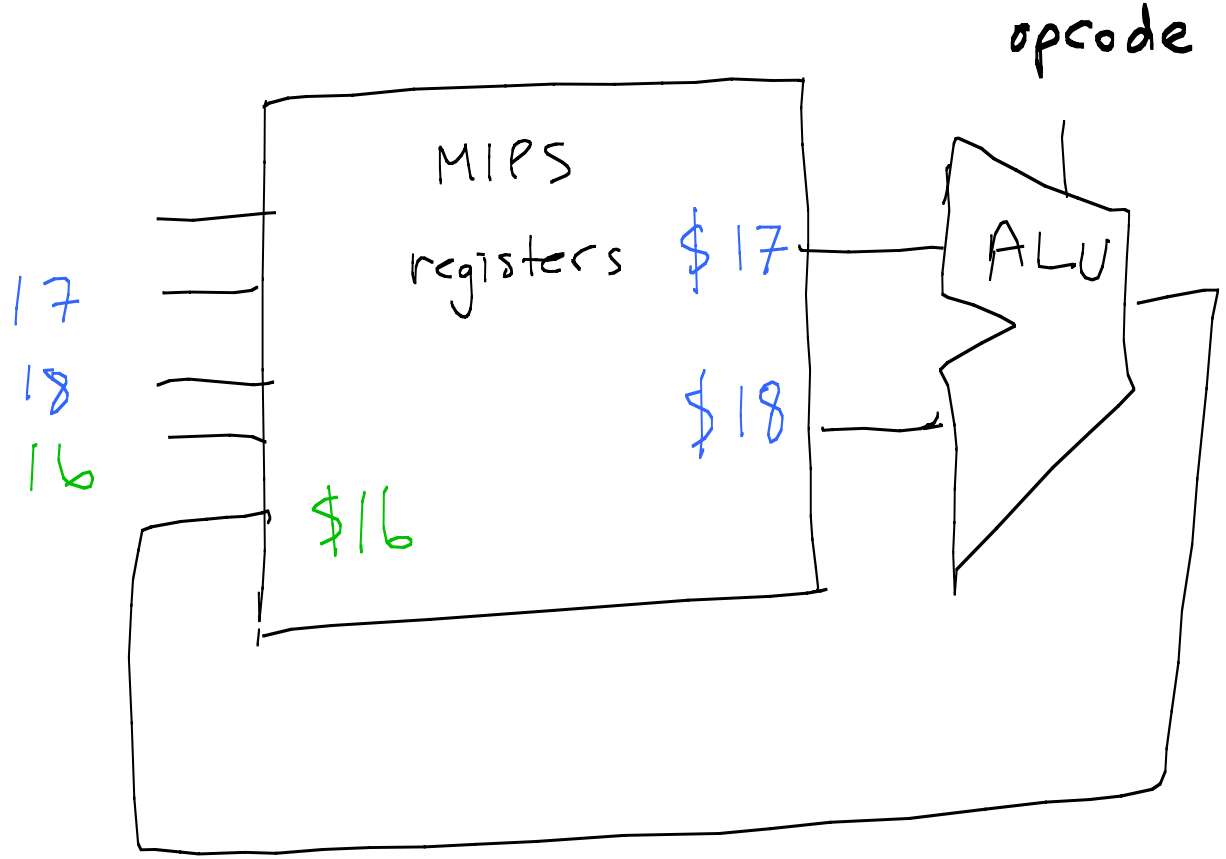

Each MIPS instruction is 32 bits.

# Examples of MIPS instructions

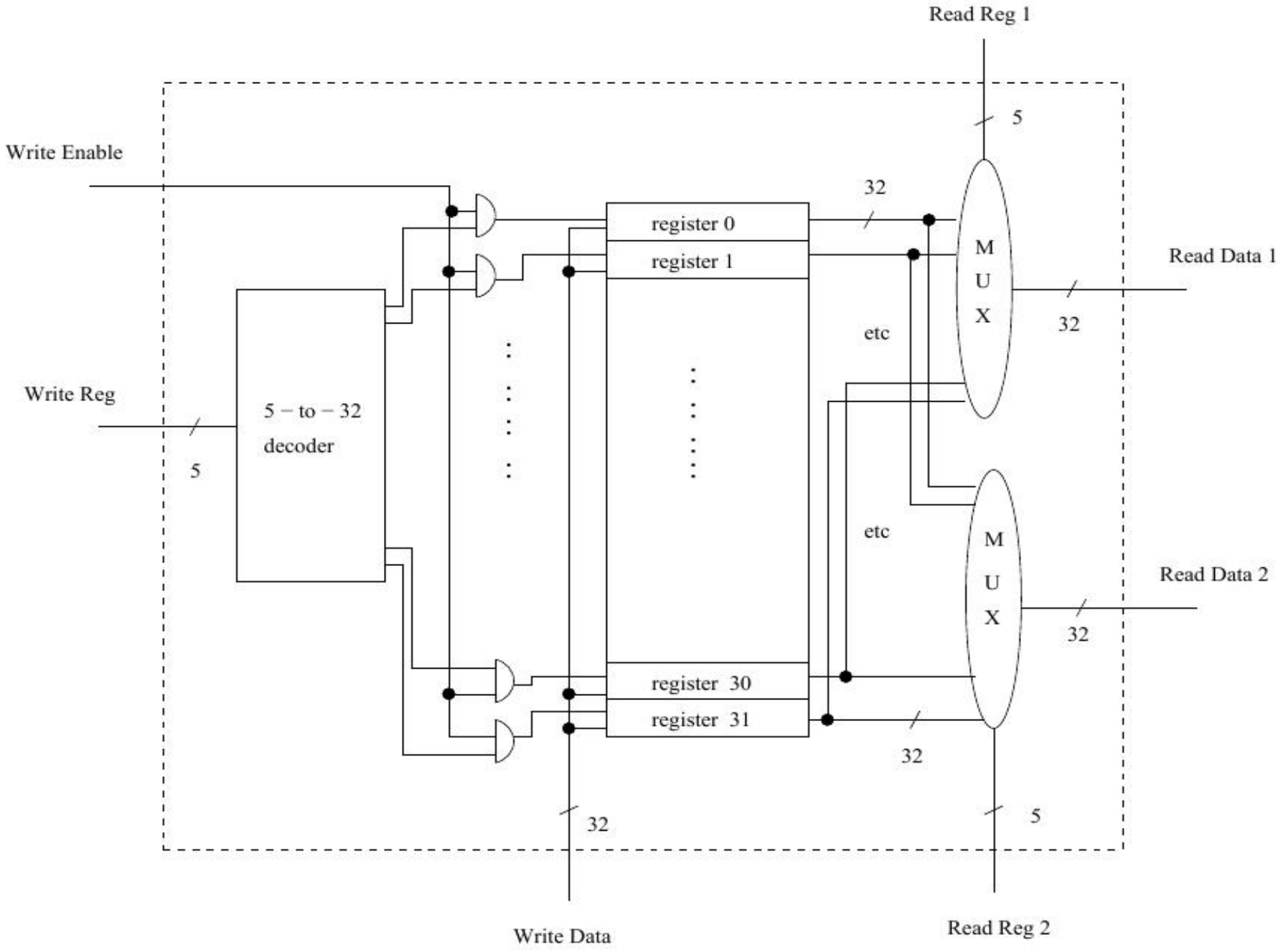
```
add   $16, $17, $18

#     register 16 is assigned
#     the sum of registers
#     17 and 18
```

↑ Comment symbol

add $16, $17, $18

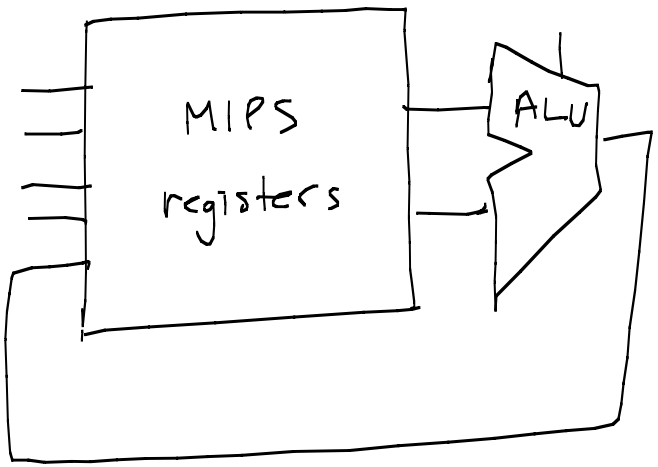


# recall lecture 6



# Arithmetic and logic instructions

```
add   $16, $17, $18
sub   $19, $15, $19
and   $17, $17, $16
or    $16, $17, $18
nor   $16, $17, $20
```



# Memory transfer instructions

# load word from Memory

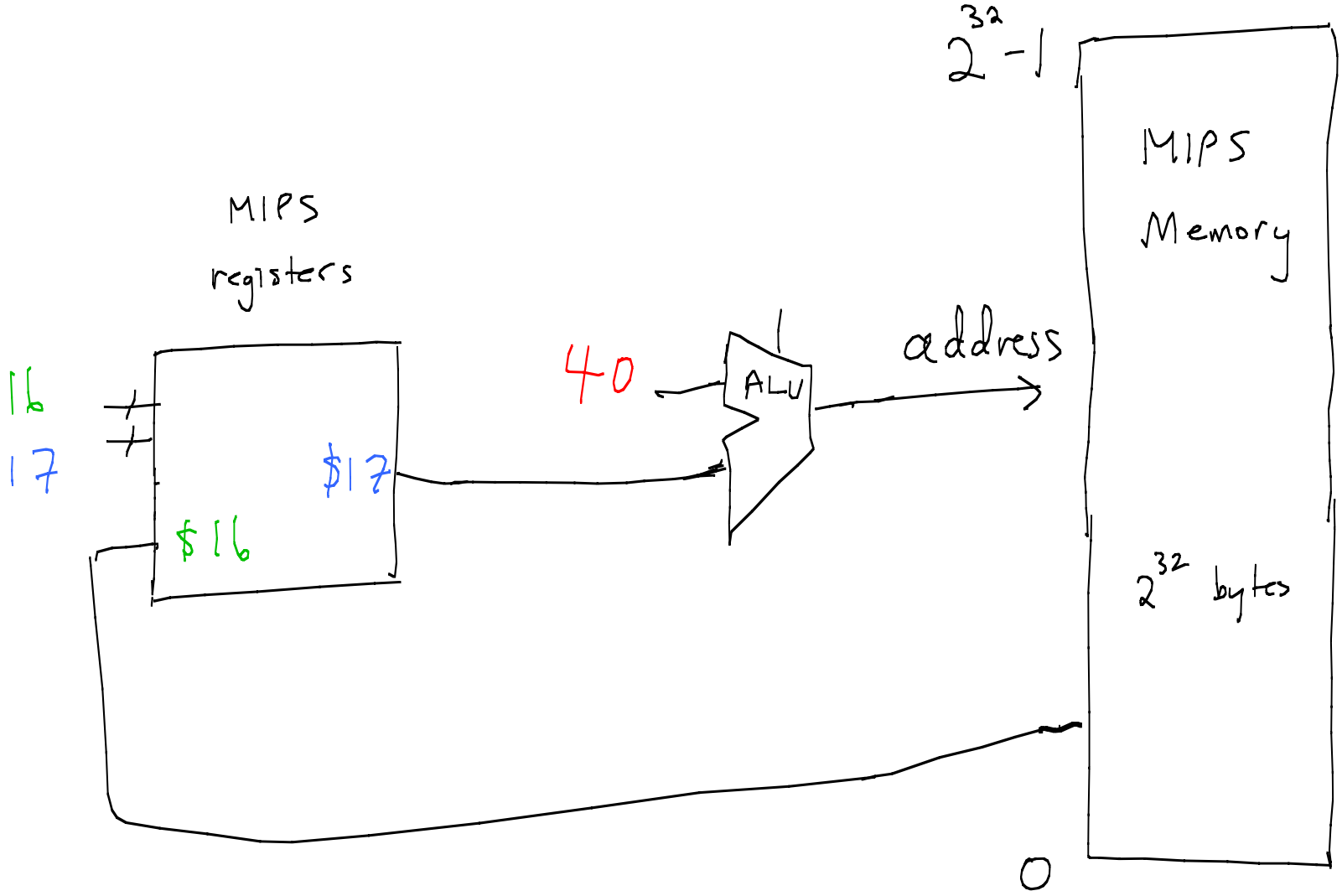
```
lw   $16, 40($17)
```

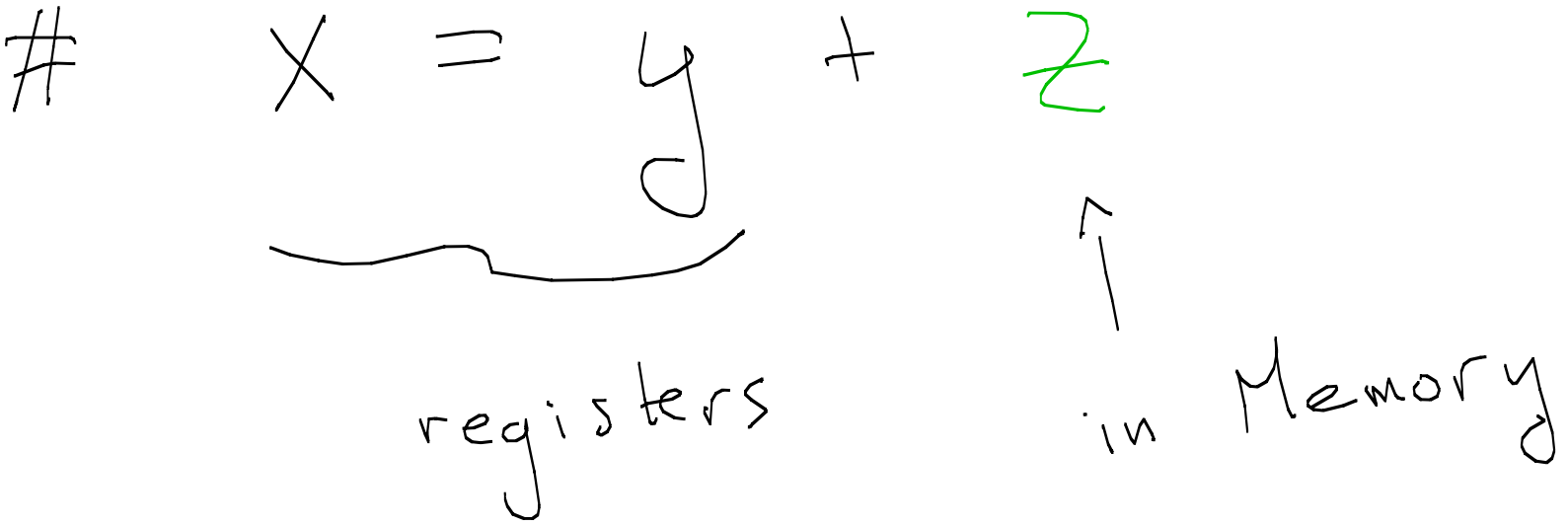
copy word from base Memory address (in $17) + offset (40) to $16

offset

Memory address (base)

lw $16, 40($17)


$2^{32}-1$
MIPS
Memory
$2^{32}$ bytes
0
MIPS
registers
16
17
$17
$16
40
ALU
address

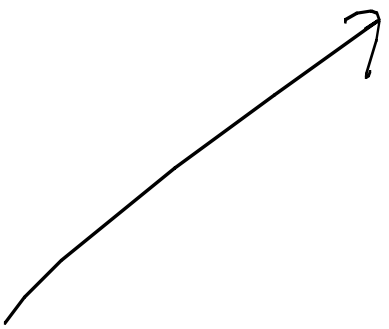



Suppose the values of x and y are assigned to registers (say $18 and $20) but the value of z is stored in Memory. Then we need to bring z into a register (say $16) in order to do the addition.

```
lw   $16, 40($17)
add  $18, $20, $16
```

# store word to Memory

```
sw $16, 40($17)
```

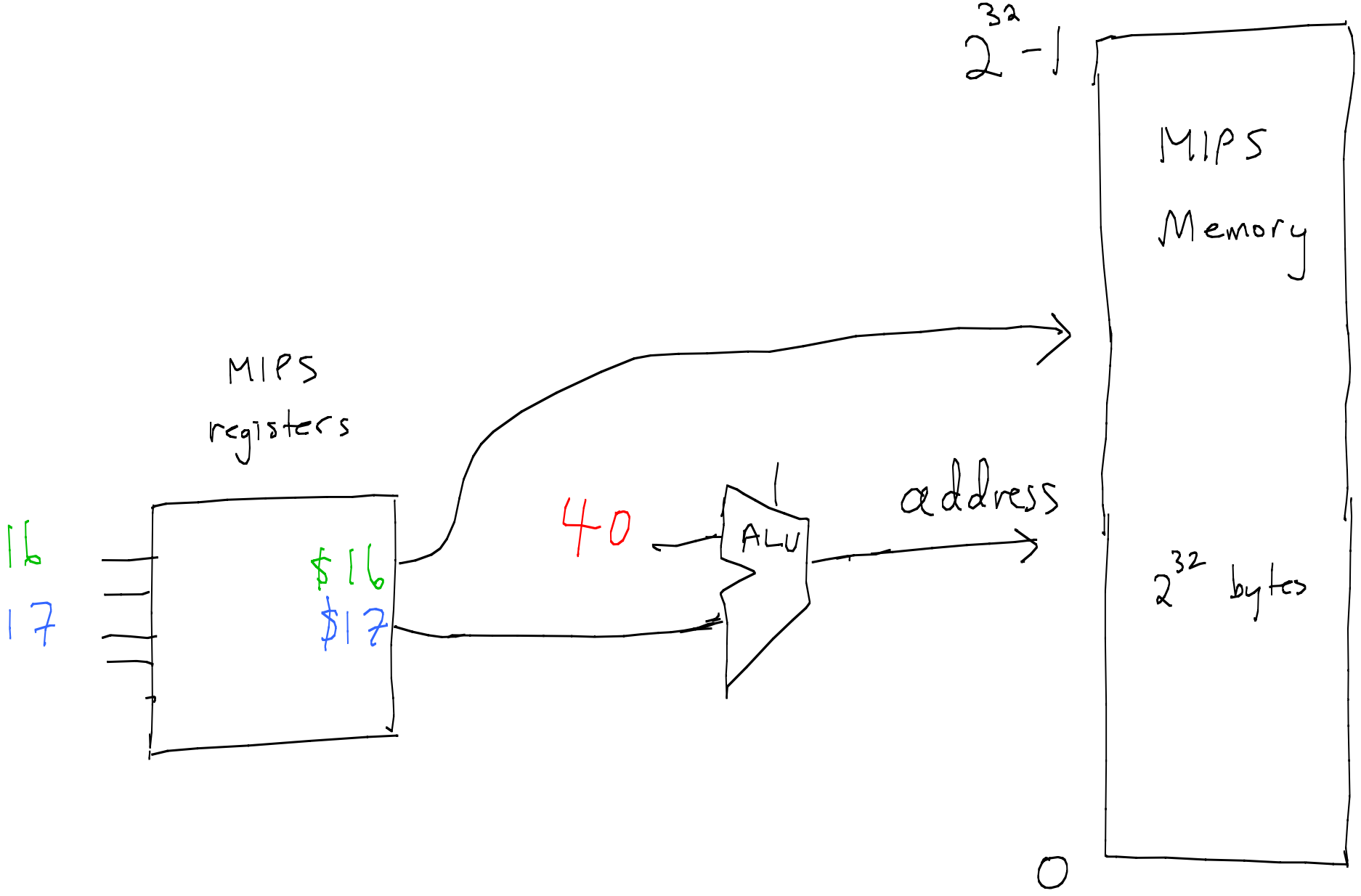


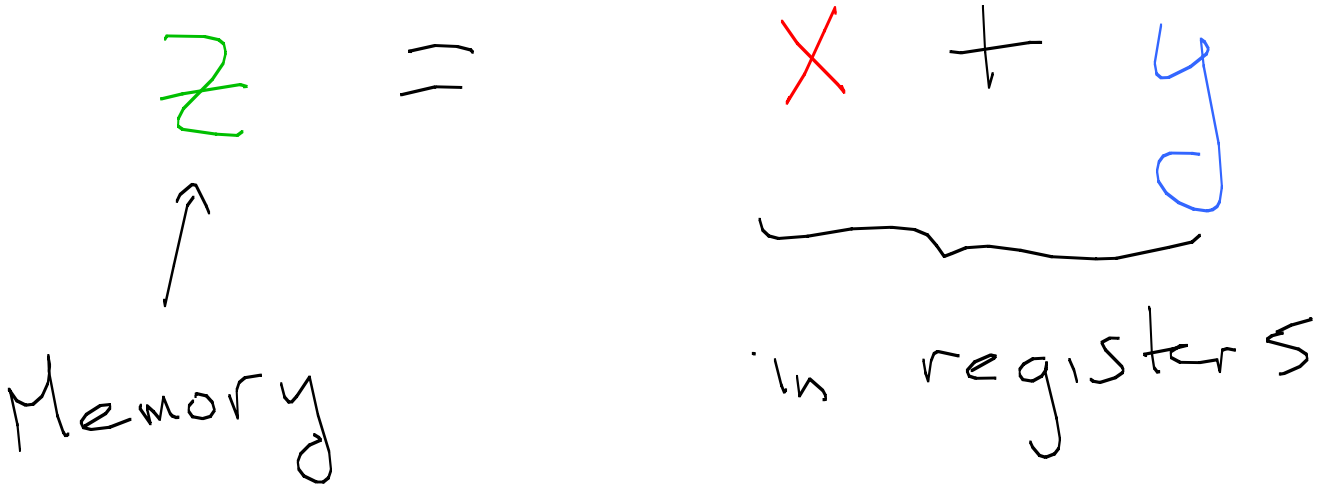
offset

base

copy word from $16 to
Memory address specified
by base (in $17) + offset (40)

sw $16, 40($17)


$2^{32}-1$
MIPS
Memory
$2^{32}$ bytes
0
MIPS
registers
16
17
$16
$17
40
ALU
address




If you want to store the result of the sum in a Memory address (you might need it later) then you must use a register.

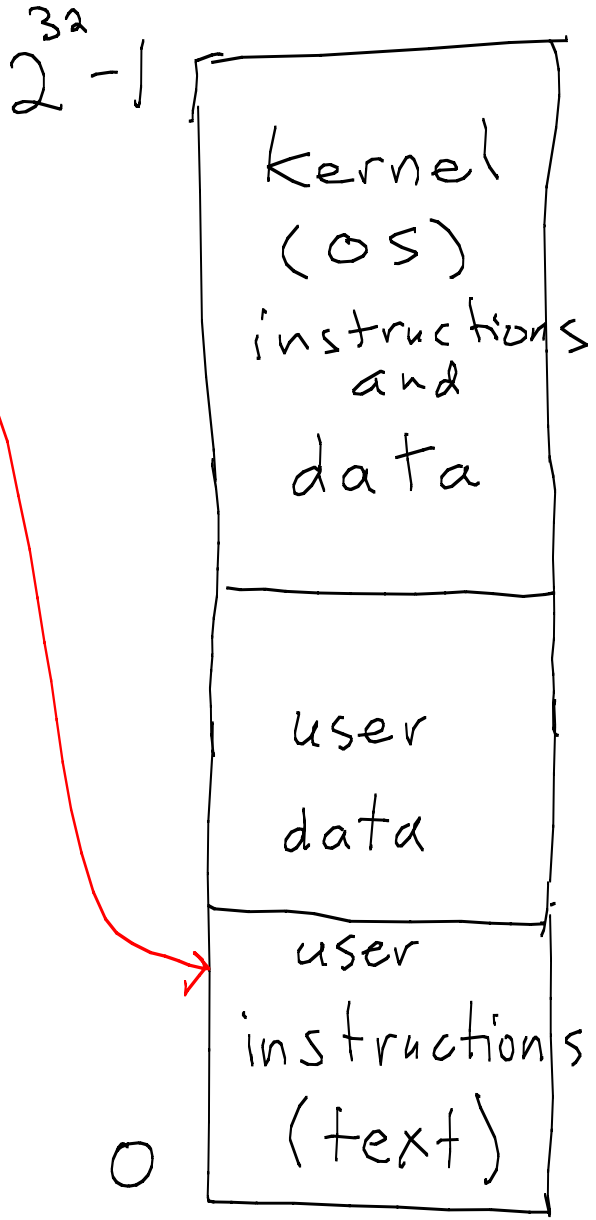
In MIPS, you cannot take the result of a summation from the ALU and put it directly in Memory.

```
add  $8, $16, $20
sw   $8, 40($17)
```

How does the computer keep track of which instruction is currently executing ?

# Program Counter

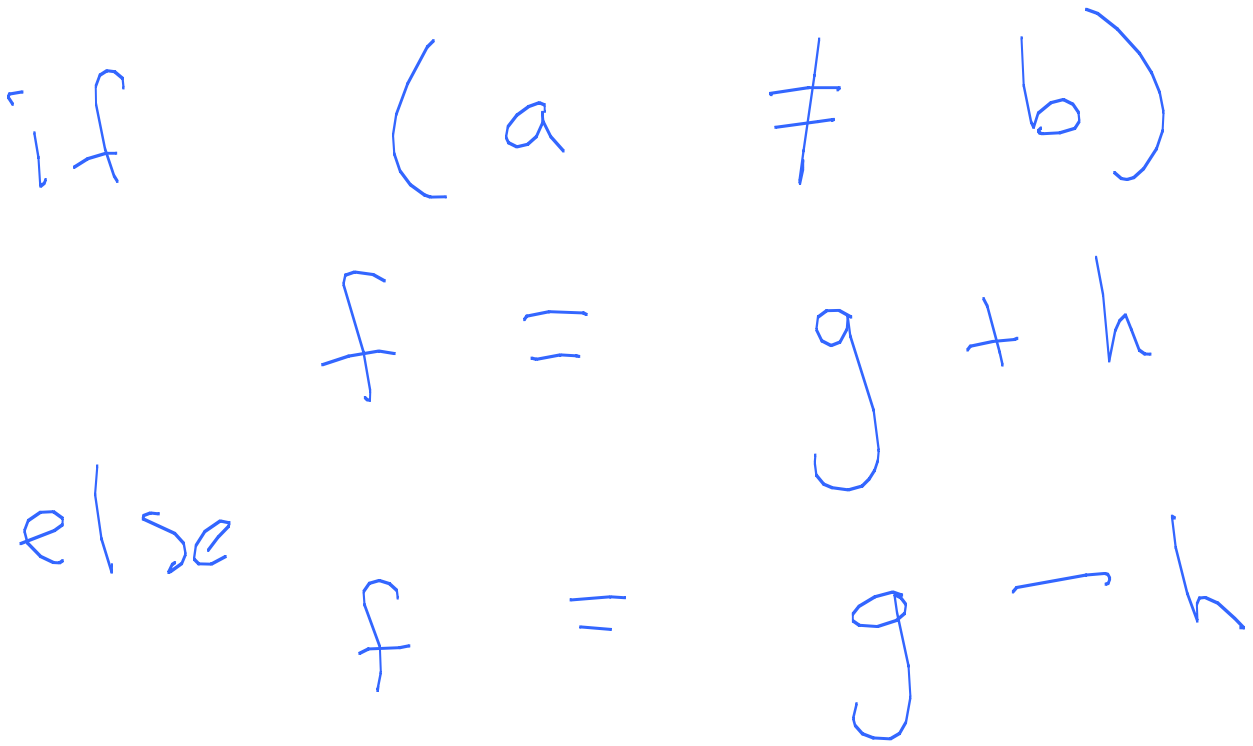
Program Counter (PC)




The Program Counter (PC) register specifies the Memory address of the instruction that is currently being executed.

The default is that the computer advances to the next instruction. But there are also branches allowed (next slide)....

# Branching Instructions




How is conditional branching done in MIPS ?

# e.g. "branch equals" (beq)

\#

\#

if ( a ≠ b )

f = g + h

```
beq   $17, $18, Exit
add   $19, $20, $21
...
Exit: ...
```

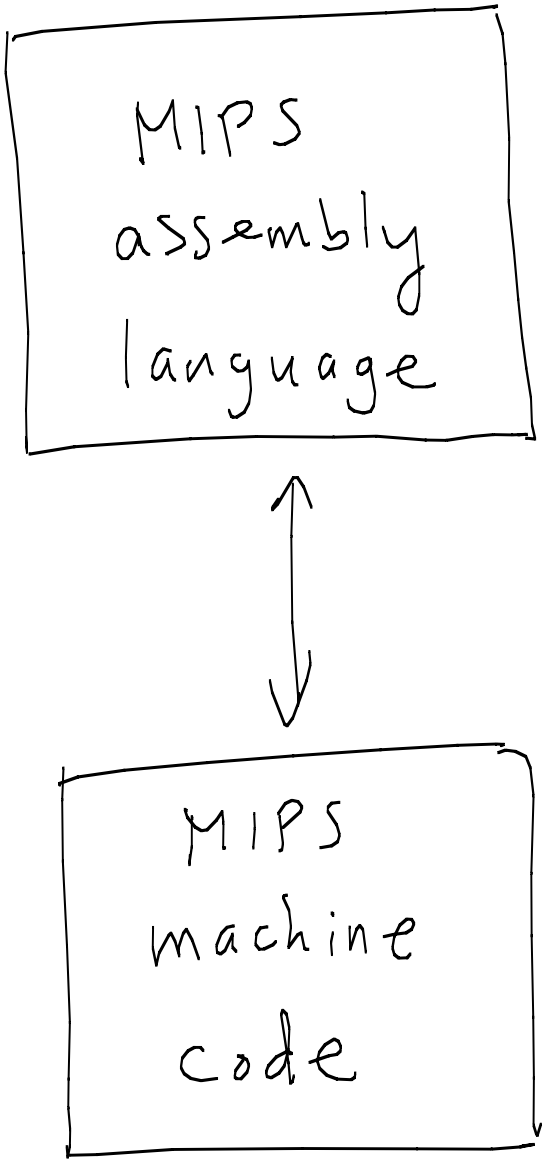
# "jump" ( j )

#
#
#
#

if ( a ≠ b)
f = g + h
else
f = g − h

```
beq     $17, $18, Exit1
add     $19, $20, $21
j       Exit2
sub     $19, $20, $21
Exit1:
Exit2:
```

human writeable & readable machine code (in ASCII)



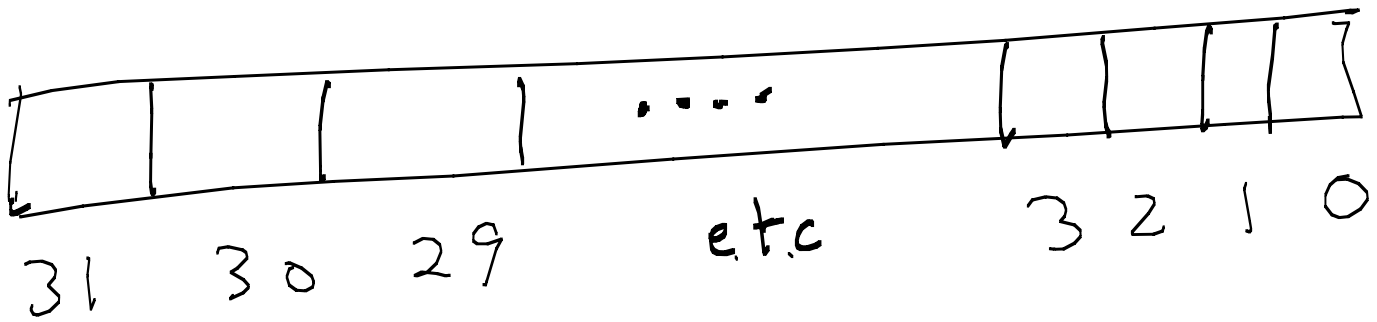

```
        beq   $17, 18, Exit1
        add   $19,$20, $18
Exit1:  sub   $19, $20, $18
        j     Exit2
Exit2:
```

0 1 1 0 1 0 1 1
0 0 1 0 0 1 0 1
. . .

?

# MIPS instructions (machine code)

- always 32 bits (one word)




- relatively few

(RISC: "reduced instruction set computer")

# MIPS instruction formats

R

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 6 | 5 | 5 | 5 | 5 | 6 |

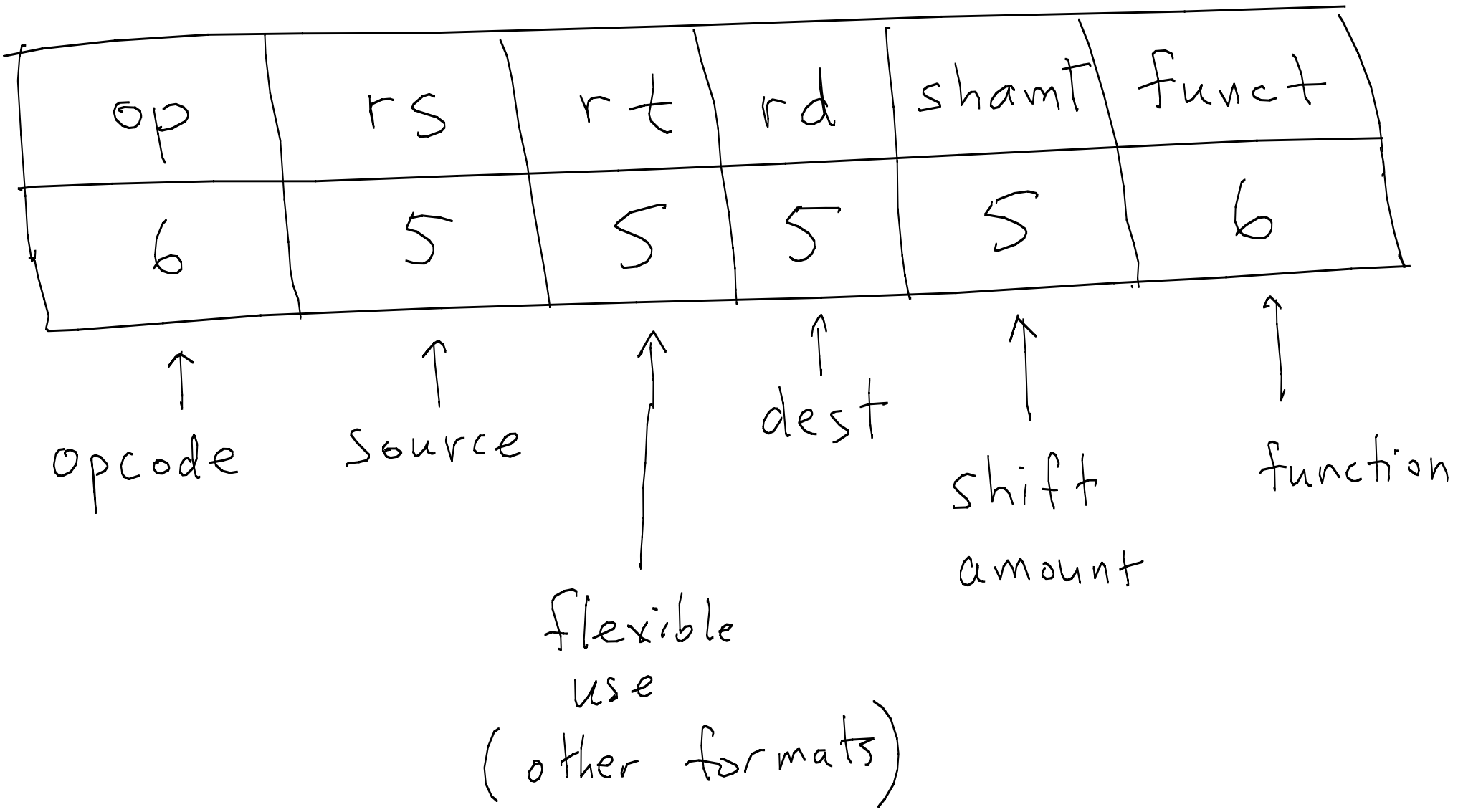
I

| op | rs | rt | immediate |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

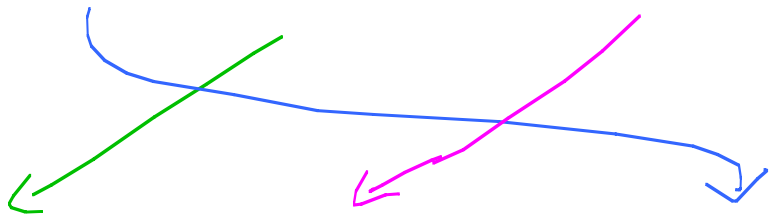
J

| op | address |
|---|---|
| 6 | 26 |

# R format instructions



add $16, $17, $18

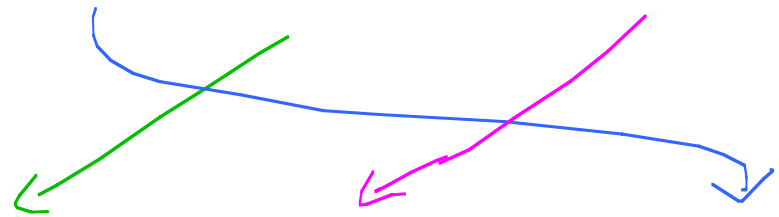


000000 10001 10010 10000 00000 100000

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 6 | 5 | 5 | 5 | 5 | 6 |

All R-format instructions have 000000 opcode.

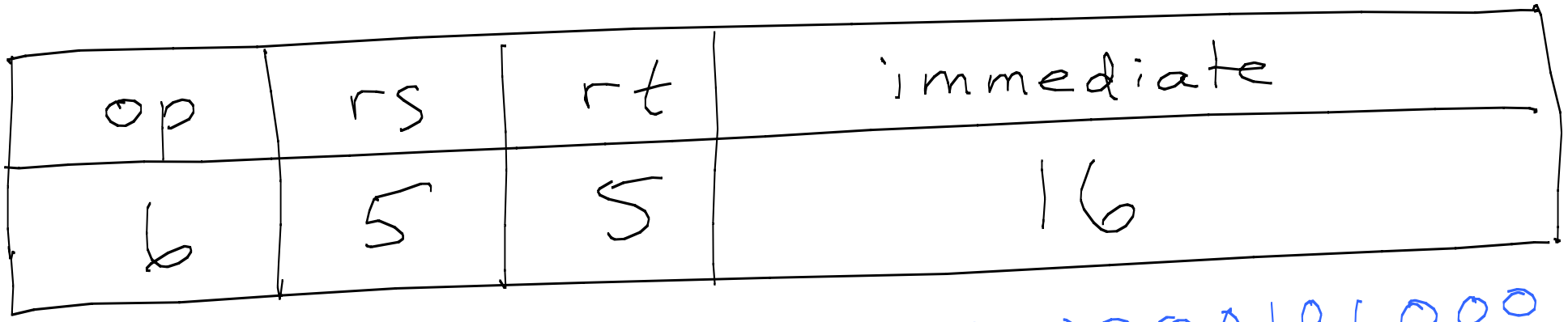
sub $16, $17, $18



000000 10001 10010 10000 00000 100010

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 6 | 5 | 5 | 5 | 5 | 6 |

# I format instructions ("immediate")




100011 10001 10000 0000000000101000

signed offset from address in $17

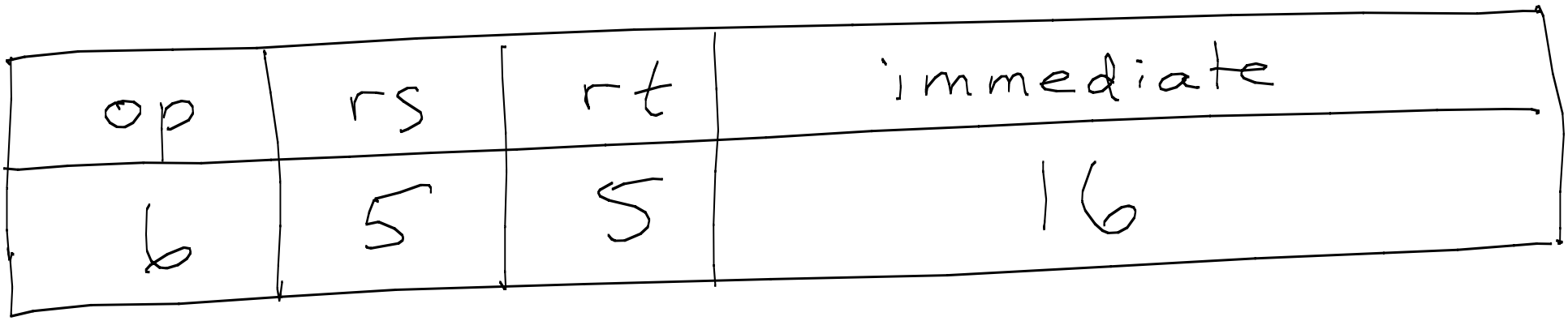
lw $16, 40($17)

rt plays the role of a"destination" register here

| op | rs | rt | immediate |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

101011 10001 10000 0000000000010000

sw $16, 40($17)

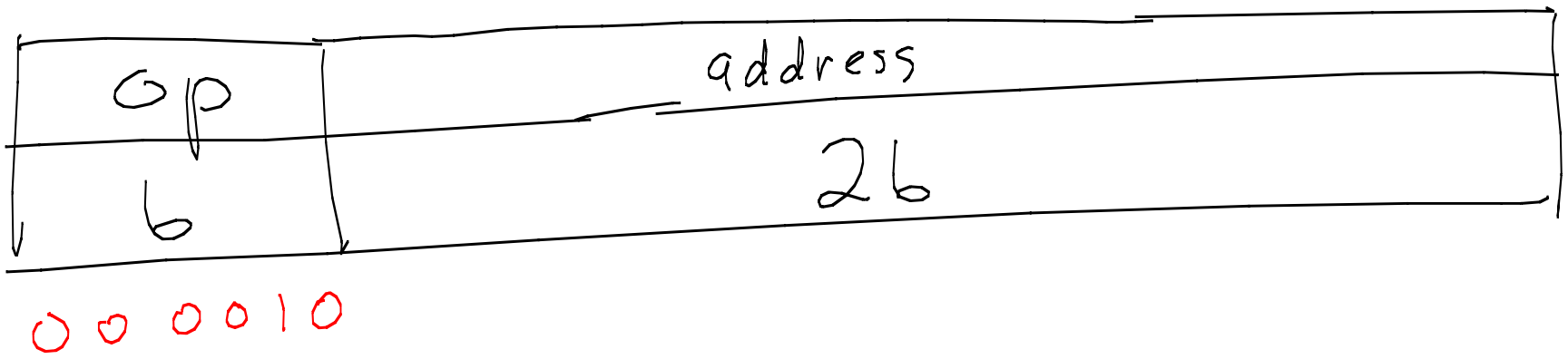



000100 10010 10001 offset

(number of words from current instruction)

beq $18, $17, Exit1

# J format instructions ("jump")




You might write the following instruction in a MIPS program. The assembler then will calculate what the offset is from the present instruction to the instruction that you have labelled Exit2. Note that there are now 26 bits of offset, which allows bigger jumps than the conditional branches.

j Exit 2

# Announcements

A1

- get started by learning basics of Logisim
  (construct simple circuits shown in class,
  e.g. left shift register)

- specification with START and DONE is the last thing you should be concerned with

Quiz 2 is Monday (lectures 3-6)