

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

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>

unicode-table.com/en/#arabic

Bookmarks Dropbox Suggested Sites Imported From IE The Storyteller Onli...

0 1 2 3 4 5 6 7 8 9 A B C D E F

0610 ؤ ء ة ة ة ة ة ة ة ة ة ة ة ة ة

0620 ي ء آ أ و إ ئ ا ب ة ت ث ج ح خ د

0630 ذ ر ز س ش ص ض ط ظ ع غ ك ك ي ئ ئ

0640 - ف ق ك ل م ن ه و ي ي ء ء ء ء

0650 ء ء ء ء ء ء ء ء ء ء ء ء ء ء ء

0660 ٠ ١ ٢ ٣ ٤ ٥ ٦ ٧ ٨ ٩ % ر ' \* ء ء

0670 ' آ أ ل ء م و و م ي ث ث ب ب ت پ ت

0680 پ خ خ ج ج ح ج ح ج ج ج ج ج ج ج

Arabic ▾

[Open in separate page](#)

Range: 0600—06FF

Click to highlight range

Languages: arabic, persian, kurd

0x0680

0x068f

- written and read by humans
- not executable (ASCII)
- not machine specific

C, Fortran,  
Java, etc

translation  
needed

machine  
code

- executable
- machine specific

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

C, Fortran  
Java

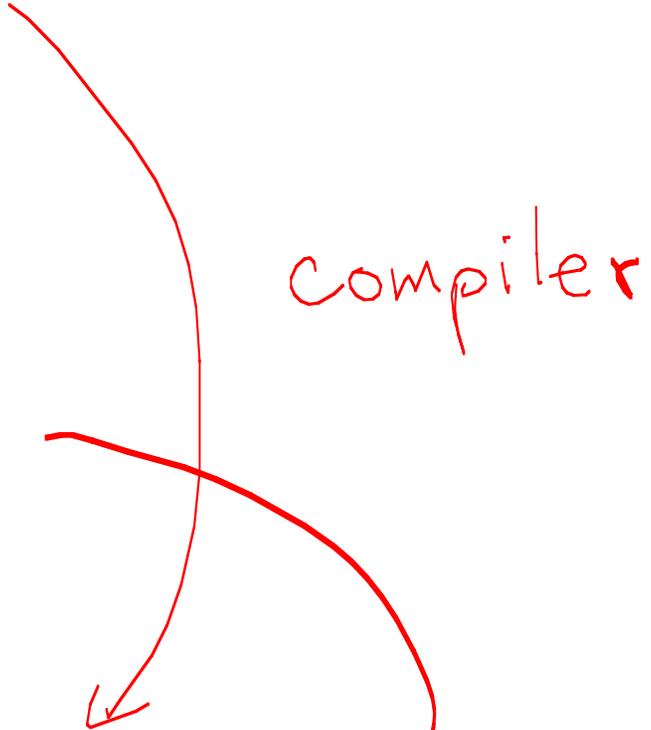
human writeable &  
readable machine  
code (in ASCII)

assembly  
language

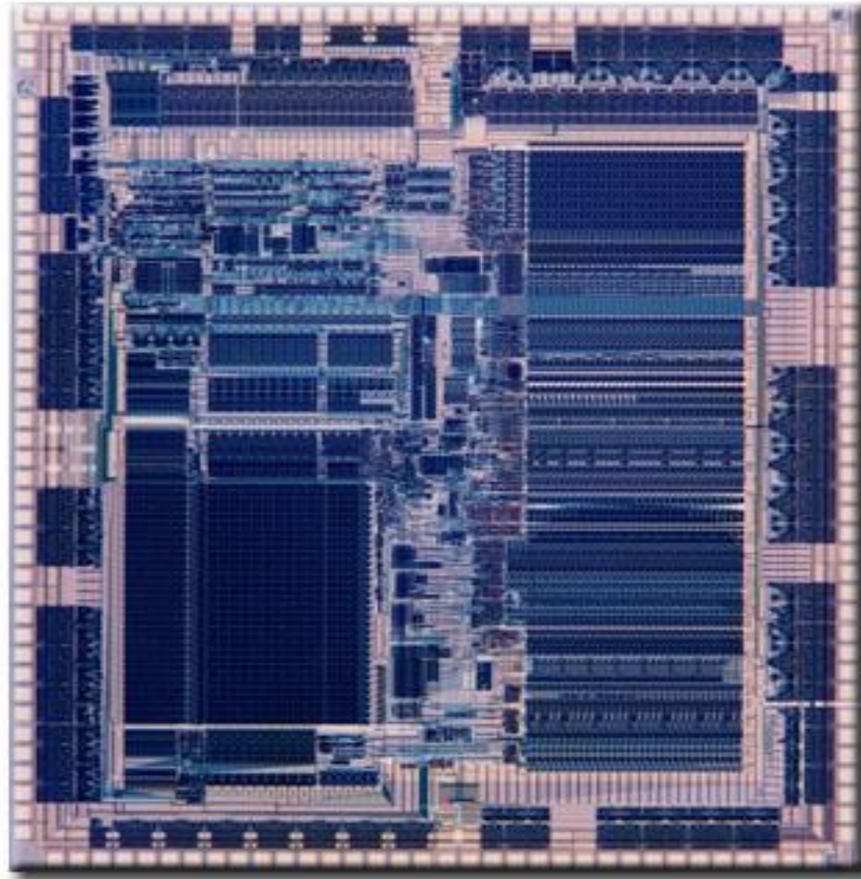


machine code

01101011  
00100101  
...



# 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

```
25 # $s4 holds a character
26 # $s5 holds the address in the current array element (a pointer to string)
27
28 # initialize variables
29
30     add    $s0, $zero, $zero
31     la    $s1, strings
32     add    $s3, $s1, $zero
33     la    $s5, stringRefArray
34
35 # print prompt to enter maximum length of a string
36
37     la    $a0, enterMaxStringPrompt
38     li    $v0, 4
39     syscall
40
41 # read maximum length of string buffer, read value goes into $v0
42 #     la    $a0, enterMaxStringPrompt
43     li    $v0, 5
44     syscall
45
46 # If user specifies that string is Nmax characters, then we need a buffer for
47 # reading the string that is at least Nmax+1 bytes. The reason is that the
```

Registers		
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Line: 7 Column: 11  Show Line Numbers

# MARS simulator

## Execute mode

The screenshot displays the MARS simulator interface in Execute mode. The main window is divided into three panes:

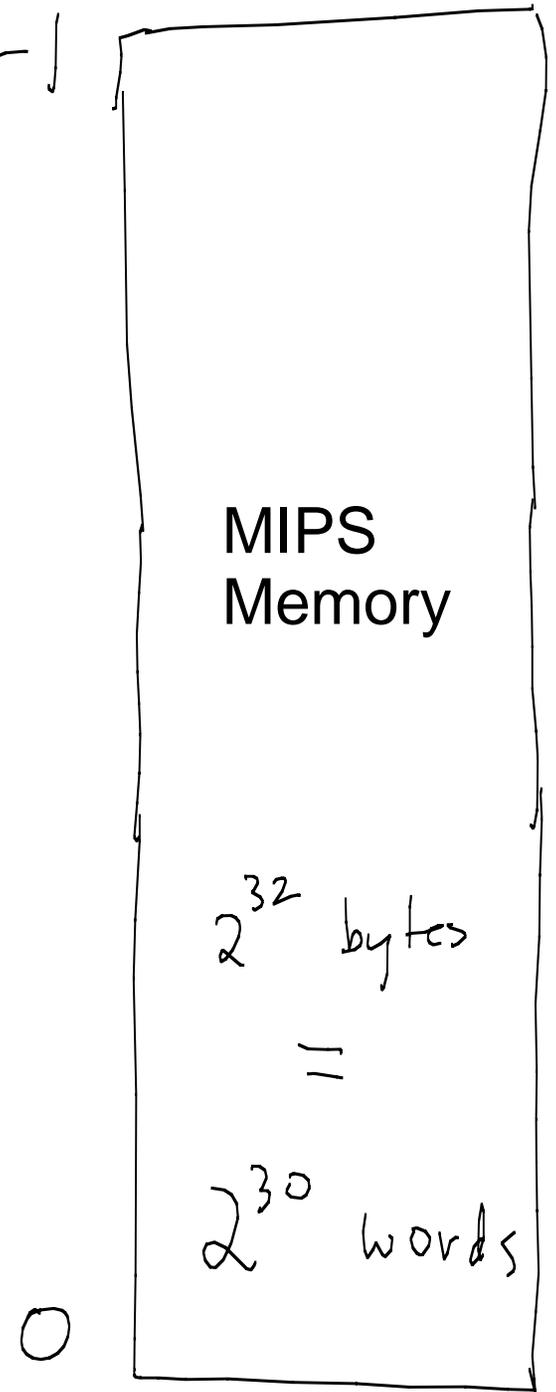
- Text Segment:** Shows assembly code with columns for Bkpt, Address, Code, Basic, and Source. The code includes instructions like `add $16, $0, $0`, `lui $1, 0x00001001`, `ori $17, $1, 0x0000002c`, `add $19, $17, $0`, `lui $1, 0x00001001`, `ori $21, $1, 0x00000004`, `lui $1, 0x00001001`, `ori $4, $1, 0x0000066c`, `addiu $2, $0, 0x00000004`, `syscall`, `addiu $2, $0, 0x00000005`, `syscall`, `addi $v0, $v0, 1`, `lui $1, 0x00001001`, `ori $8, $1, 0x00000000`, `sw $v0, 0($t0)`, and `lui $1, 0x00001001`.
- Data Segment:** Shows a memory dump with columns for Address and Value (+0) through (+1c). All values are currently 0x00000000.
- Registers:** Shows the state of CPU registers. The registers listed are \$zero through \$lo, with values ranging from 0x00000000 to 0x7ffffeffc.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

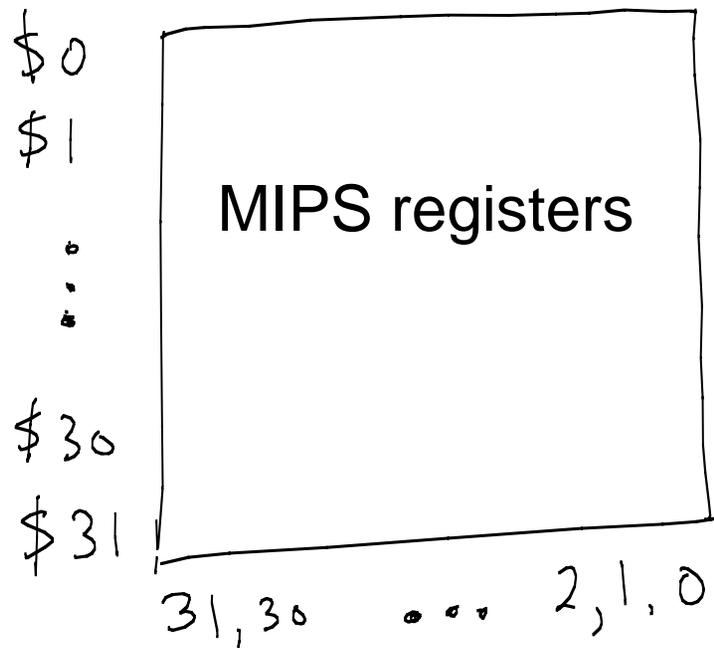
# Addressing in MIPS



$$2^{32} - 1$$

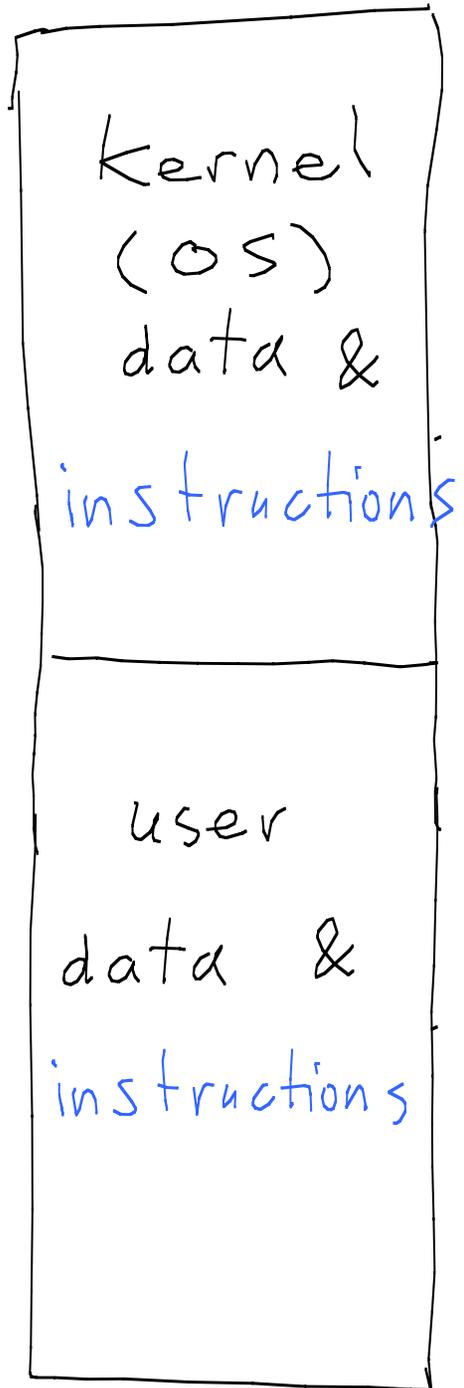


1 word = 4 bytes  
= 32 bits



$$2^{32} - 1$$

MIPS  
Memory



Each MIPS instruction is 32 bits.

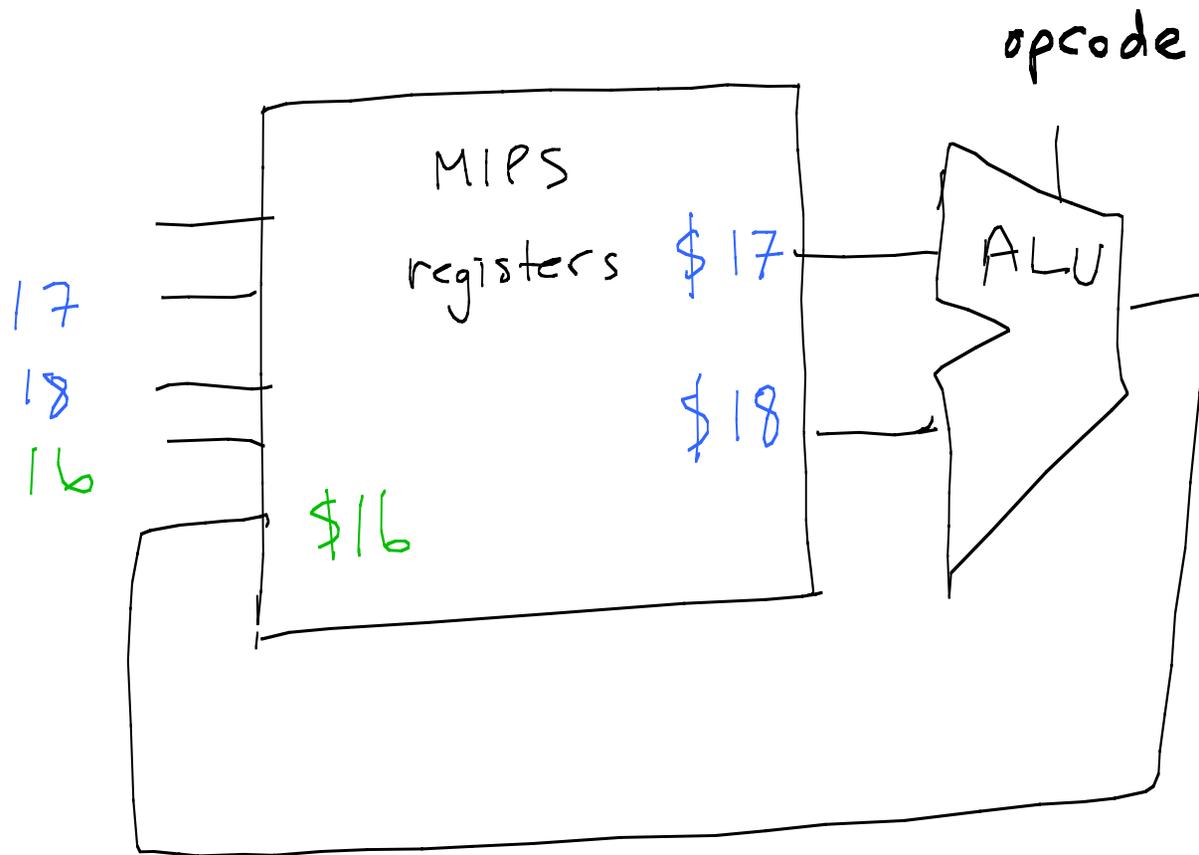
# Examples of MIPS instructions

add \$16, \$17, \$18

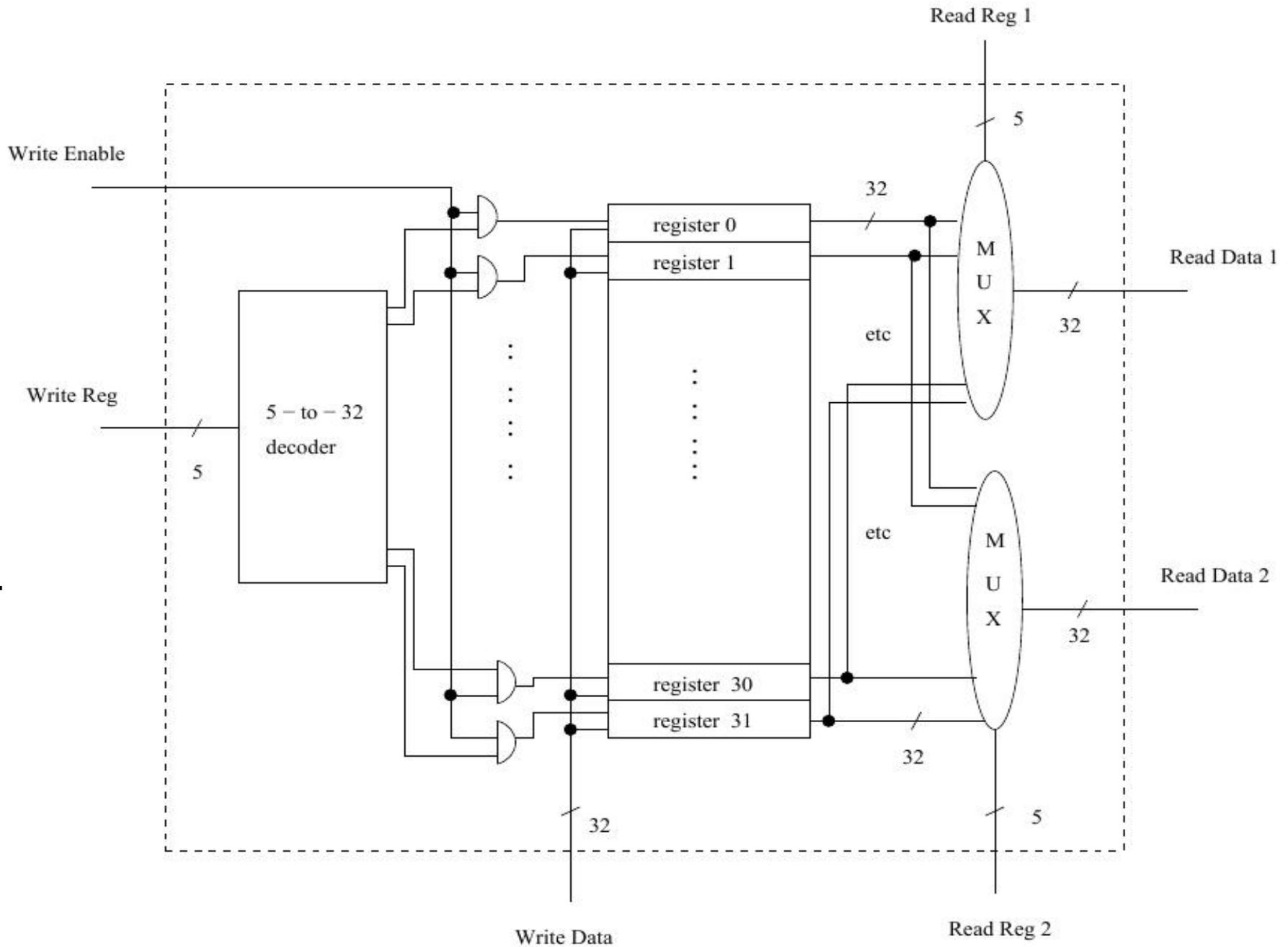
## register 16 is assigned  
## the sum of registers  
## 17 and 18

↑  
Comment symbol

add \$16, \$17, \$18



# recall lecture 6



# Arithmetic and logic instructions

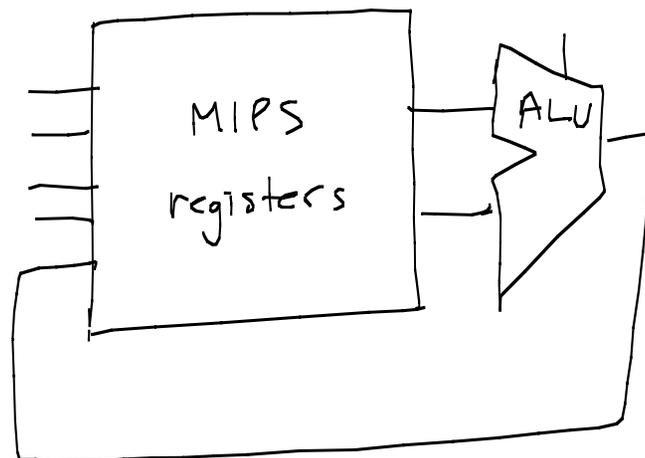
add \$16, \$17, \$18

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)

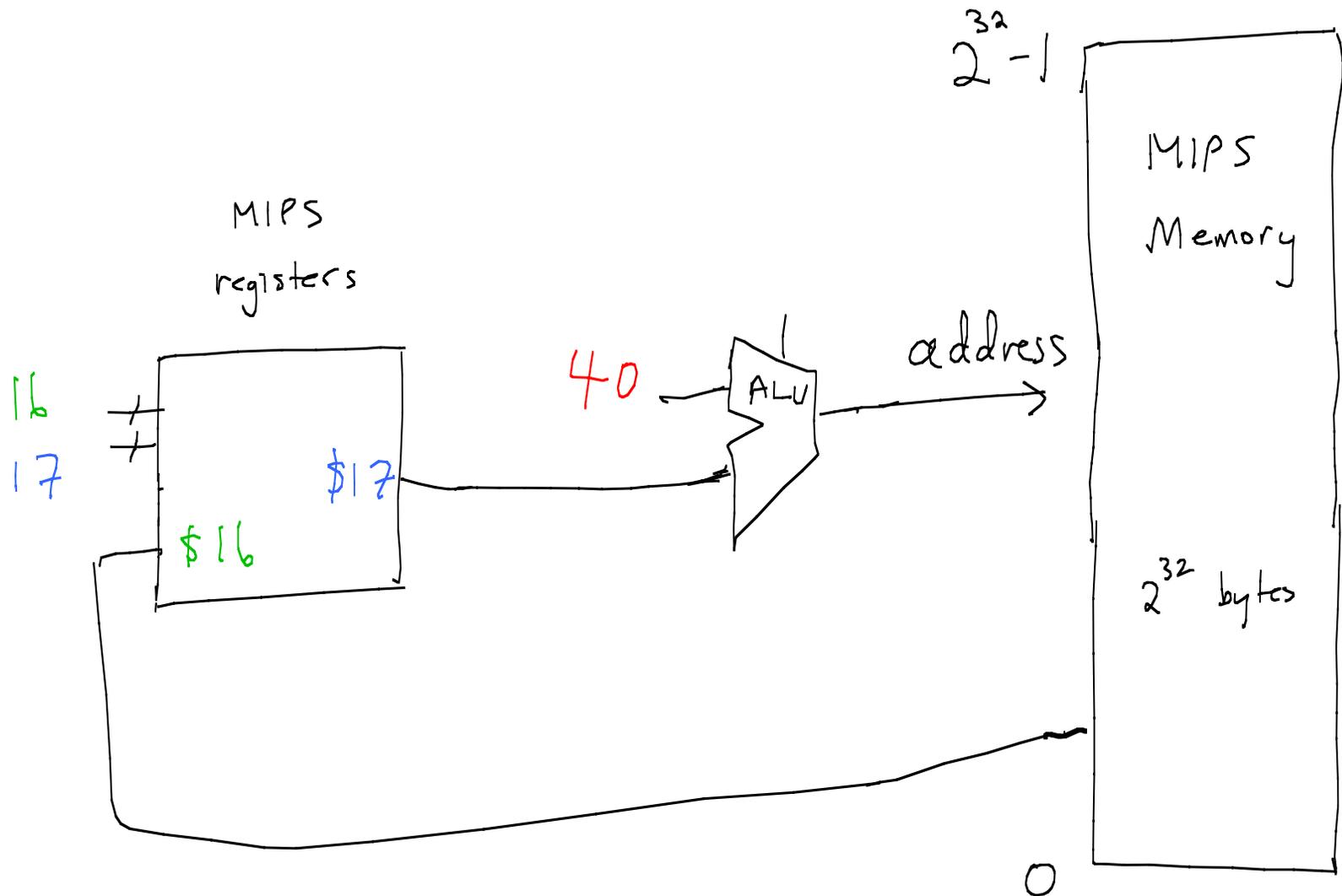


copy word from  
base Memory  
address (in \$17) +  
offset (40) to \$16

offset

Memory address  
(base)

lw \$16, 40(\$17)



$$\# \quad X = y + z$$

registers ↑ in Memory

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

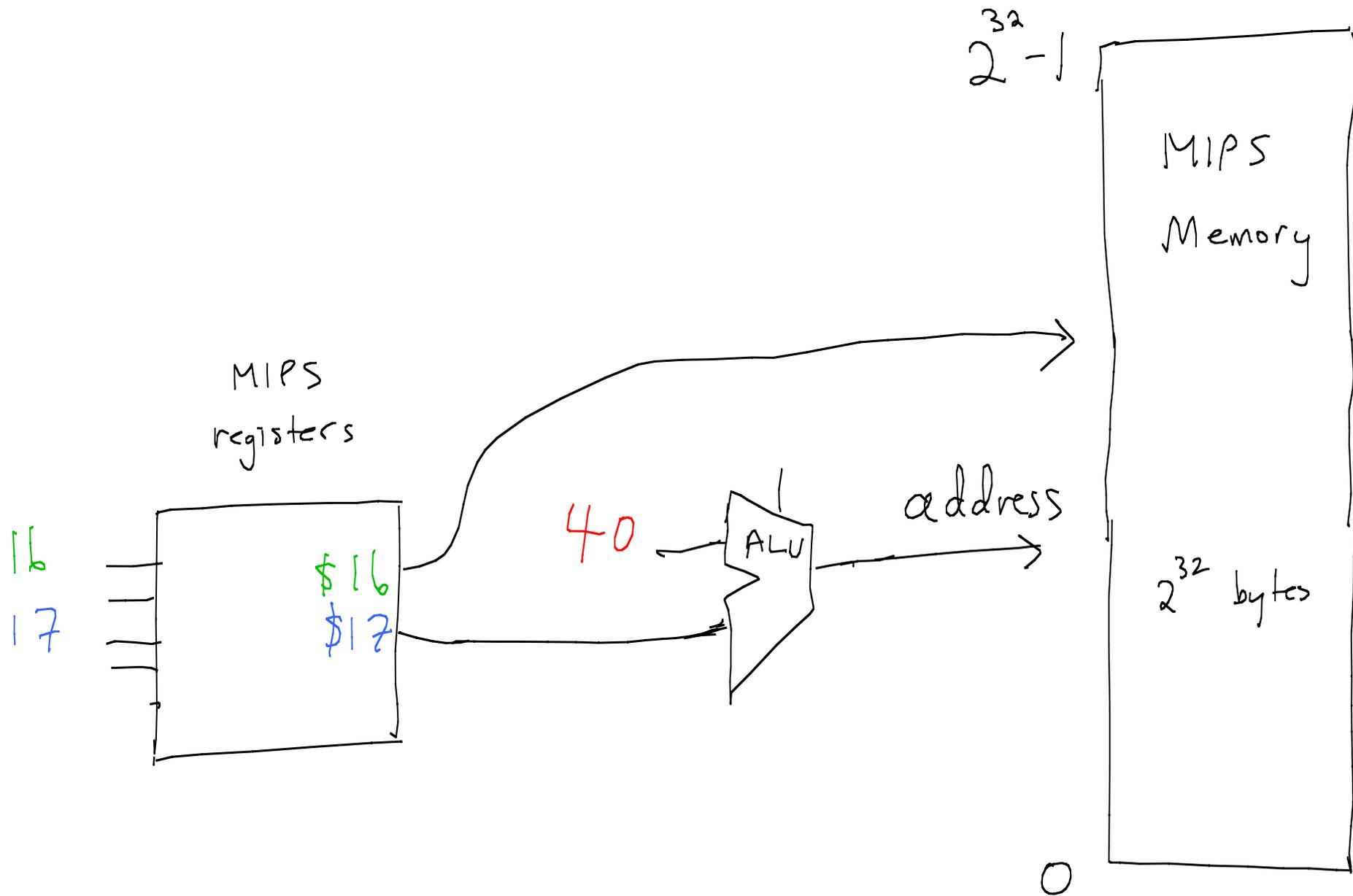


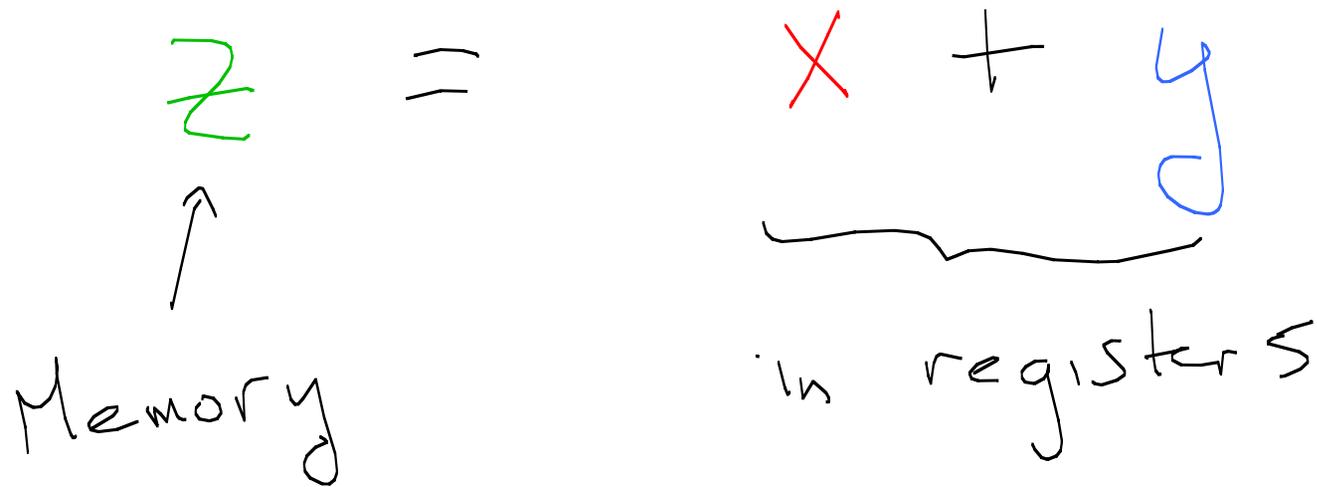
copy word from `$16` to  
Memory address specified  
by base (in `$17`) + offset (`40`)

offset

base

SW      \$16, 40 (\$17)





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)

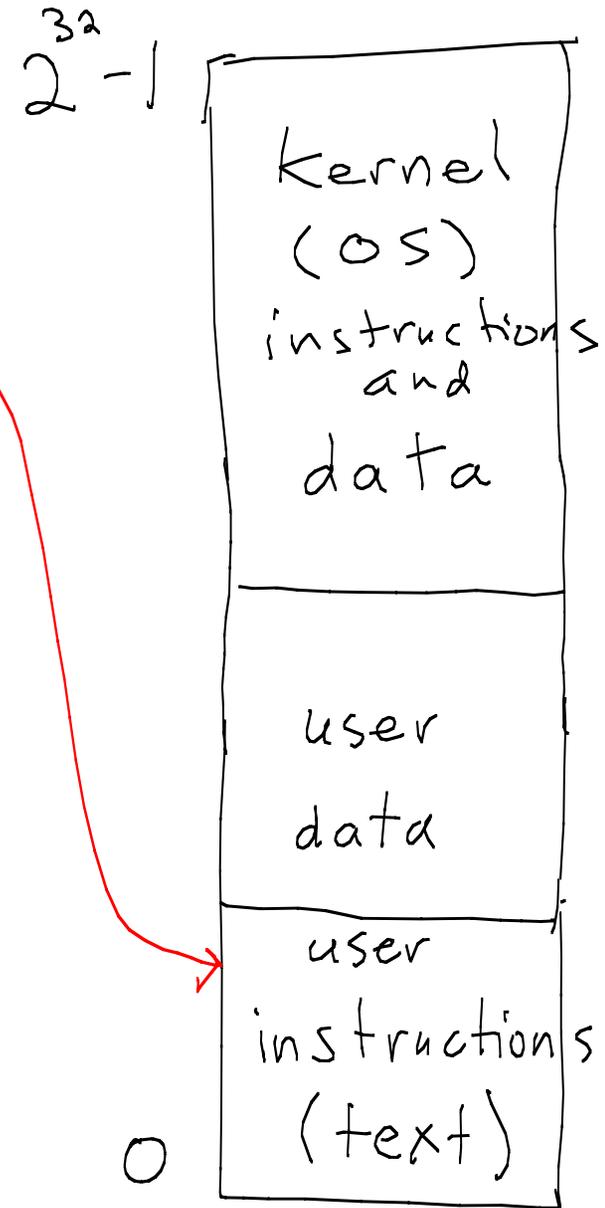
How does the computer keep track of which instruction is currently executing ?

# 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

if ( a  $\neq$  b )  
    f = g + h  
else  
    f = g - h

How is conditional branching done in MIPS ?

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

#

if (a ≠ b)

#

f = g + h

**beq**

\$17, \$18, Exit 1

add

\$19, \$20, \$21

Exit 1 :

...

# "jump" (j)

#

#

#

#

```
if (a != b)
    f = g + h
else
    f = g - h
```

```
beg    $17, $18, Exit1
```

```
add    $19, $20, $21
```

```
j      Exit2
```

Exit 1:

```
sub
```

```
$19, $20, $21
```

Exit 2:

human writeable &  
readable machine  
code (in ASCII)

MIPS  
assembly  
language



MIPS  
machine  
code

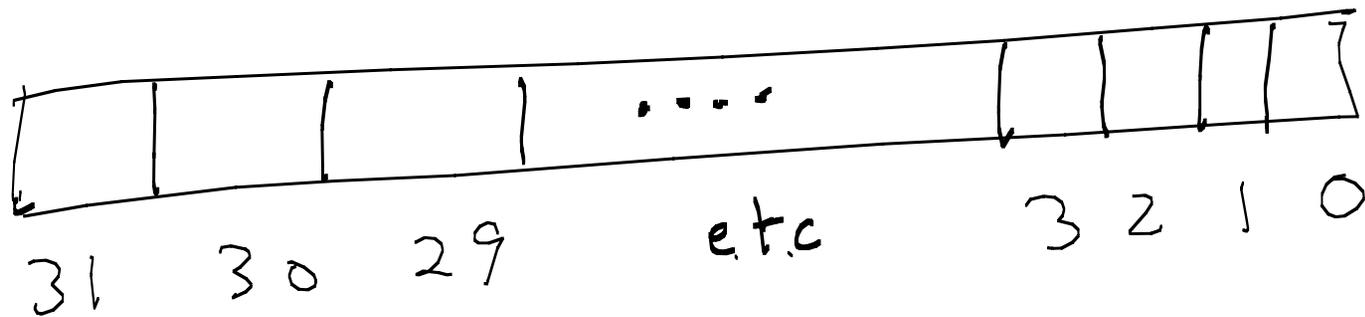
01101011  
00100101  
...

```
beg    $17, 18, Exit1  
add    $19, $20, $18  
Exit 1: sub  $19, $20, $18  
        j    Exit 2  
Exit 2:
```



# 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

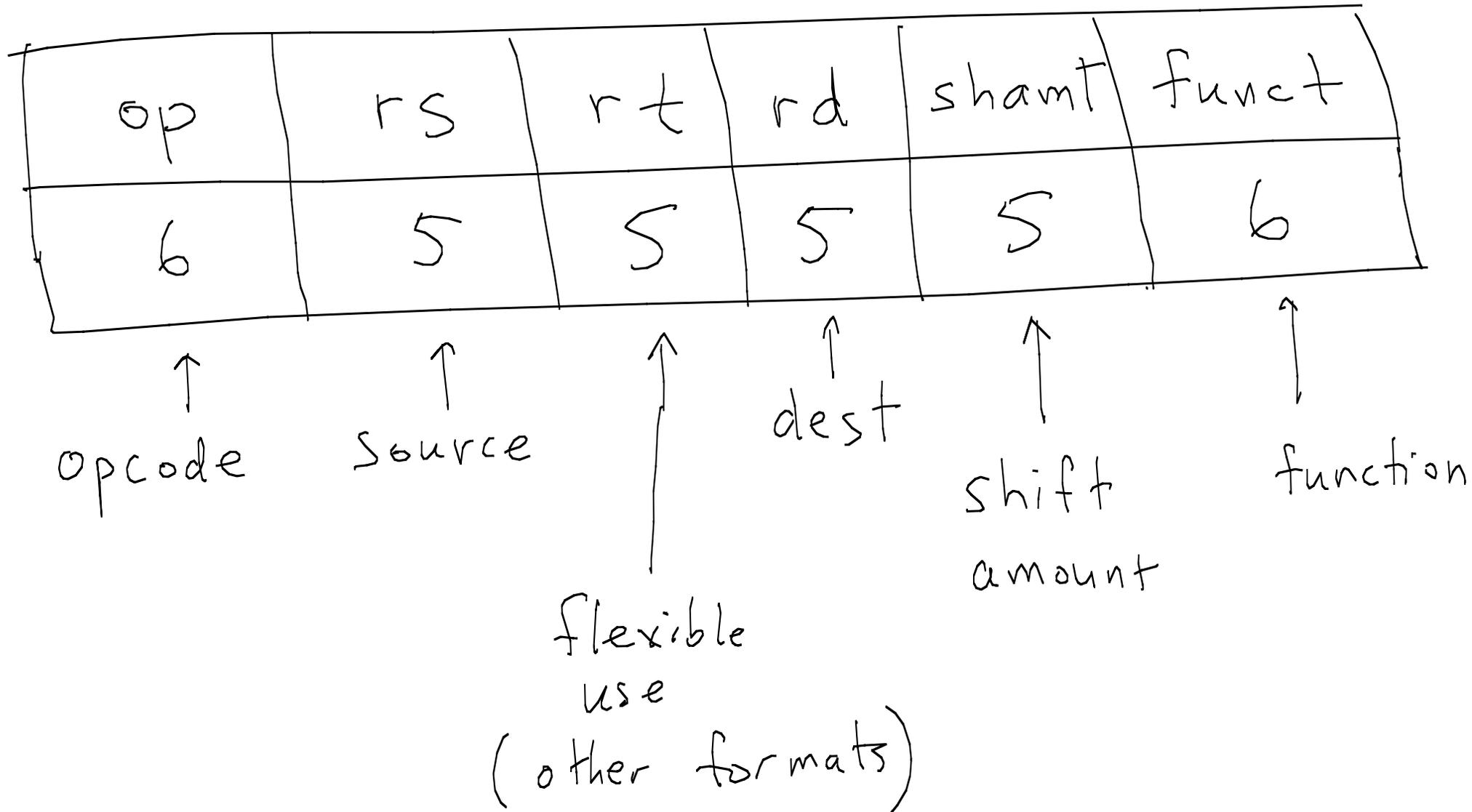
I

op	rs	rt	immediate
6	5	5	16

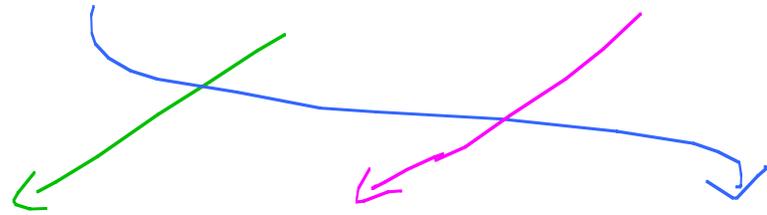
J

op	address
6	26

# R format instructions



add \$16, \$17, \$18

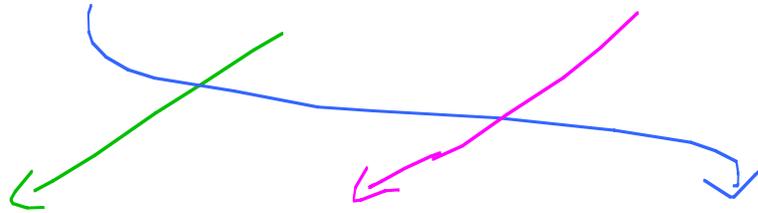


000000 10001 10010 10000 00000 100000

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

All R-format instructions have 000000 opcode.

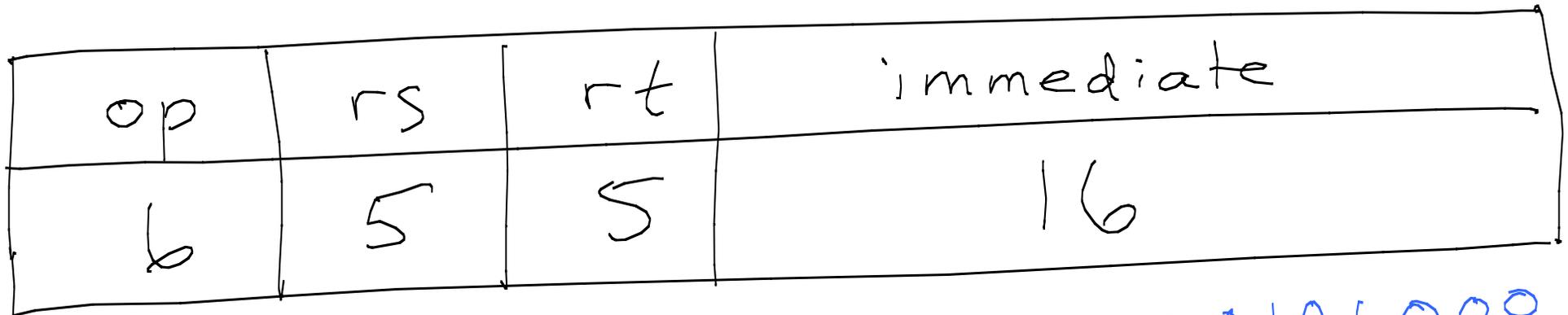
sub \$16, \$17, \$18



000000 10001 10010 10000 00000 100010

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

# I format instructions ("immediate")



010011 10001 10000 000000000000101000

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

011011 10001 10000 000000000000101000

sw      \$16, 40 (\$17)

op	rs	rt	immediate
6	5	5	16

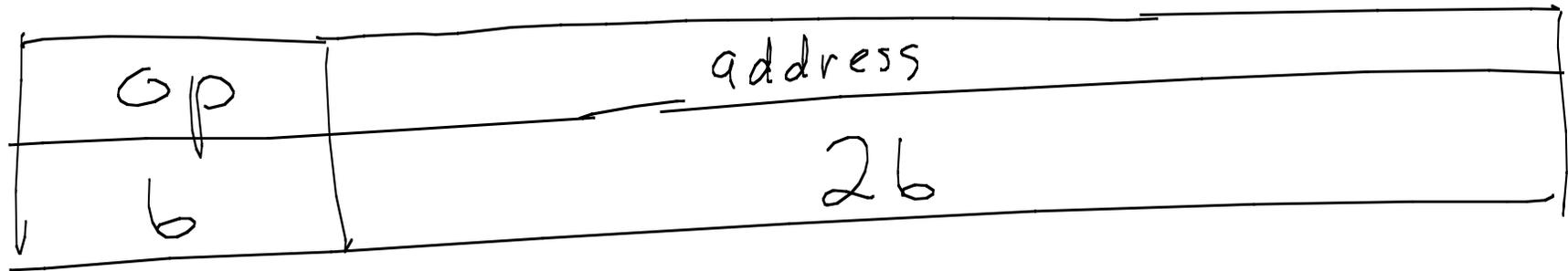
000100 10010 10001

offset

(number of words from  
current instruction)

beg \$18, \$17, Exit!

# J format instructions ("jump")



000010

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 Exit2

# 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)