MIPS32 AL – Instruction Encoding
(early glimpse @ MIPS addressing modes)

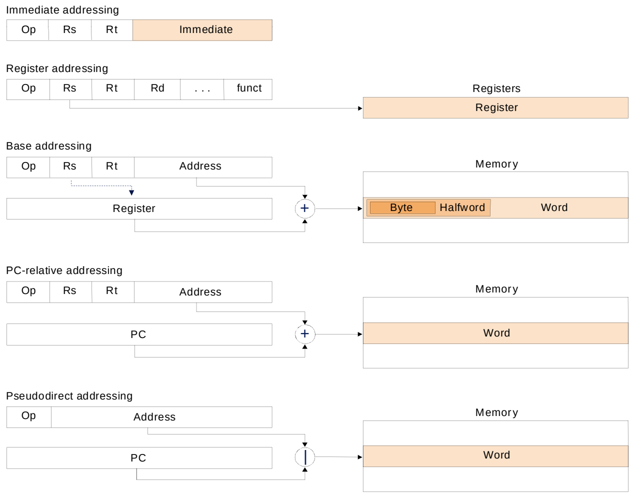
Slide #16
006 ComputerOrg&DesignOverview01

- Register addressing
  - Operand is a register
- Immediate addressing
  - Operand is a constant within instruction itself
- Base or displacement addressing
  - Operand is at memory location whose address is sum of a register and a constant in instruction
- PC-relative addressing
  - Branch address is sum of PC and a constant in instruction
- Pseudodirect addressing
  - Jump address is 26 bits of instruction concanated with upper bits of PC

MIPS32 AL – Instruction Encoding
(simplified comparative view of MIPS addressing modes)

EXAMPLE

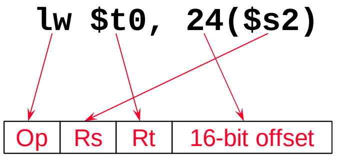
```
addi $t1, $t0, -5

add $t2, $t0, $t1

lw $t1, 12($t0)

beq $t1, $t0, label

j label
```



MIPS32 AL – Instruction Encoding

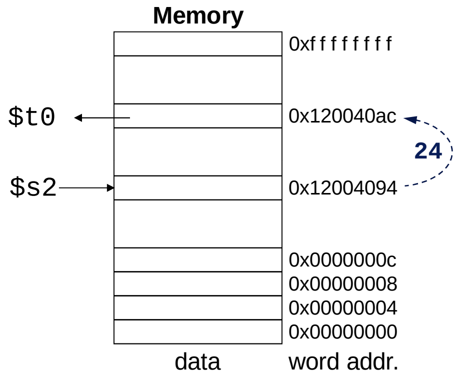
(how load/store memory address is formed)

- By adding contents of base address register to *byte* offset
  - 16-bit offset (maybe +ve or –ve) limits access to memory locations within ±2^{15} or 32,768 bytes (±2^{13} or 8,192 words) of base register address



$24_{10} + \$s2 =$

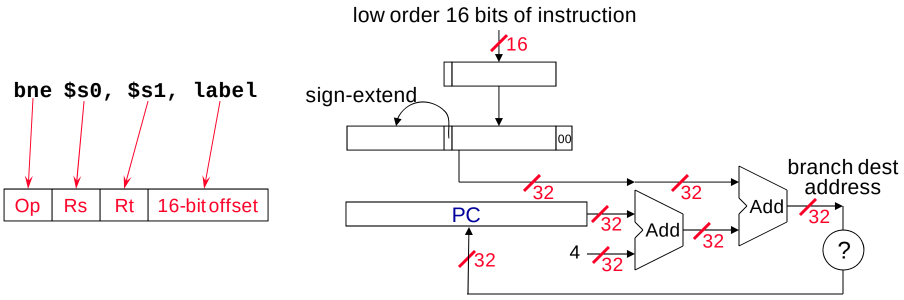
```
      . . . 0001 1000
+     . . . 1001 0100
-----------------------
      . . . 1010 1100

         = 0x120040ac
```



MIPS32 AL – Instruction Encoding

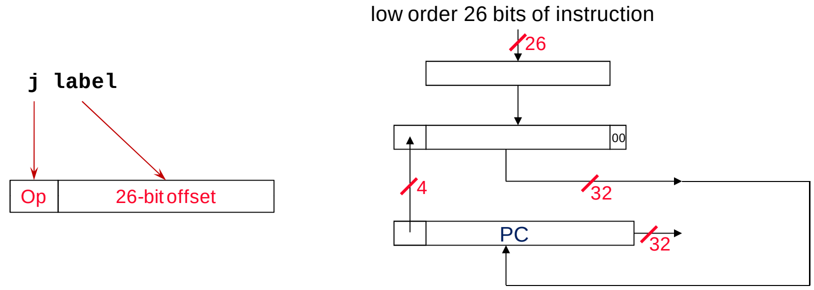
(how branch destination address is formed)

- By adding 16-bit *word* offset to address in a register (much like in `lw` and `sw`)
  - But which register? → PC
    - ☞ PC updated to PC+4 during fetch cycle so it holds *address of next instruction*
  - Limits branch distance to about 32K (-2^{15} to 2^{15}-1) *instructions* from "instruction after branch instruction" → good for typically local branches



MIPS32 AL – Instruction Encoding
(how jump destination address is formed)

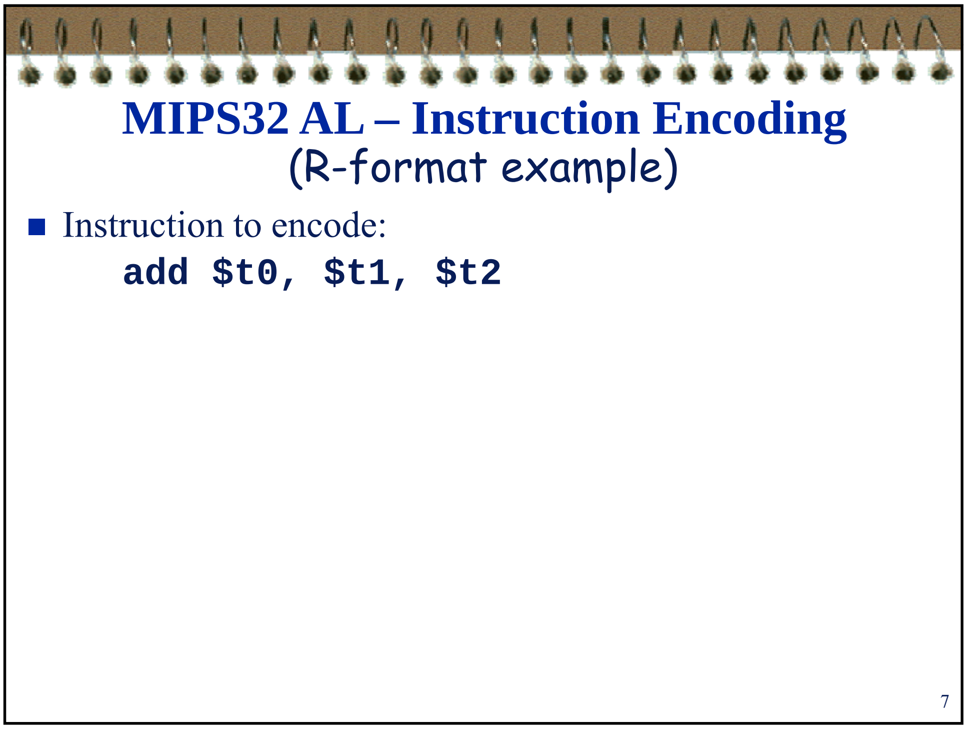
- By concatenating upper 4 bits of PC to 26-bit *word* offset
  - Limits jump target to within block of 256M (*2^{28}*) *addresses* whose upper 4 bits match those of PC (whose upper 4 bits are used in concatenation)



- For typical memory layout (Slide #4 of ***008 MIPS32ArchitectureOverview***):
  - can jump to *anywhere* in ***text segment*** (goes from 0x**0**0400000 to 0x**0**FFFFFFC)
- **Q:** What if there's need to jump to anywhere in memory (2^{32} ≡ 4G address space)?
  A: *Load 32-bit address in register* (how?) and do **`jr <register>`**

MIPS32 AL – Instruction Encoding
(lecture note supplement)

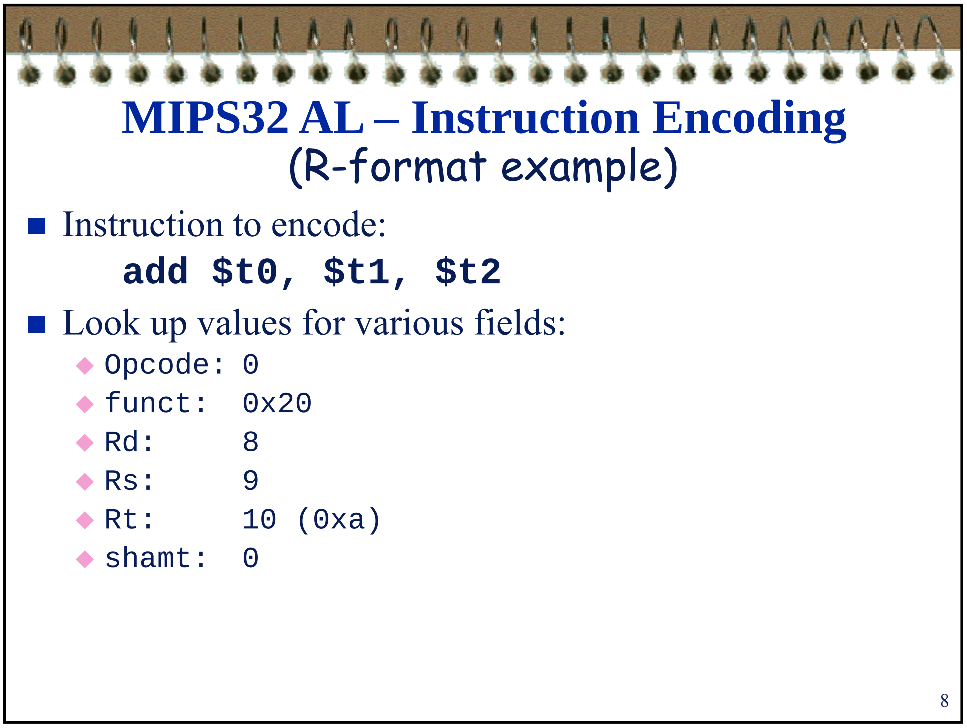
- ***012 MIPS32AssemblyLanguageInstructionEncodingSup01***
  - Tabulates encoding information for "true" MIPS instructions covered in ***010 MIPS32AssemblyLanguageDoingBasics01***
  - Also has
    - *register_name–register_number* correspondence table
    - breakdown of opcode ranges (in decimal) for the R, I and J instructions
- For use with instruction encoding examples in following slides
- (something similar will be provided during exams)



MIPS32 AL – Instruction Encoding
(R-format example)

- Instruction to encode:

```
add $t0, $t1, $t2
```



MIPS32 AL – Instruction Encoding
(R-format example)

- Instruction to encode:

```
add $t0, $t1, $t2
```

- Look up values for various fields:
  - `Opcode: 0`
  - `funct:  0x20`
  - `Rd:     8`
  - `Rs:     9`
  - `Rt:     10 (0xa)`
  - `shamt:  0`

MIPS32 AL – Instruction Encoding
(R-format example)

- Fill in values for each field:

0	9	0xa	8	0	0x20

- Instruction in binary:

000000	01001	01010	01000	00000	100000

- Instruction in hex: **`012A4020`**

000000	01001	01010	01000	00000	100000

MIPS32 AL – Instruction Encoding
(I-format example)

- Instruction to encode:

  `addi $s5, $s6, -50`

- Look up values for various fields:
  - `Opcode: 8`
  - `Rs:     22 (0x16)`
  - `Rt:     21 (0x15)`
  - `imm:    -50 (0xffce)`

MIPS32 AL – Instruction Encoding
(I-format example)

- Fill in values for each field:

8	**0x16**	**0x15**	**0xffce**

- Instruction in binary:

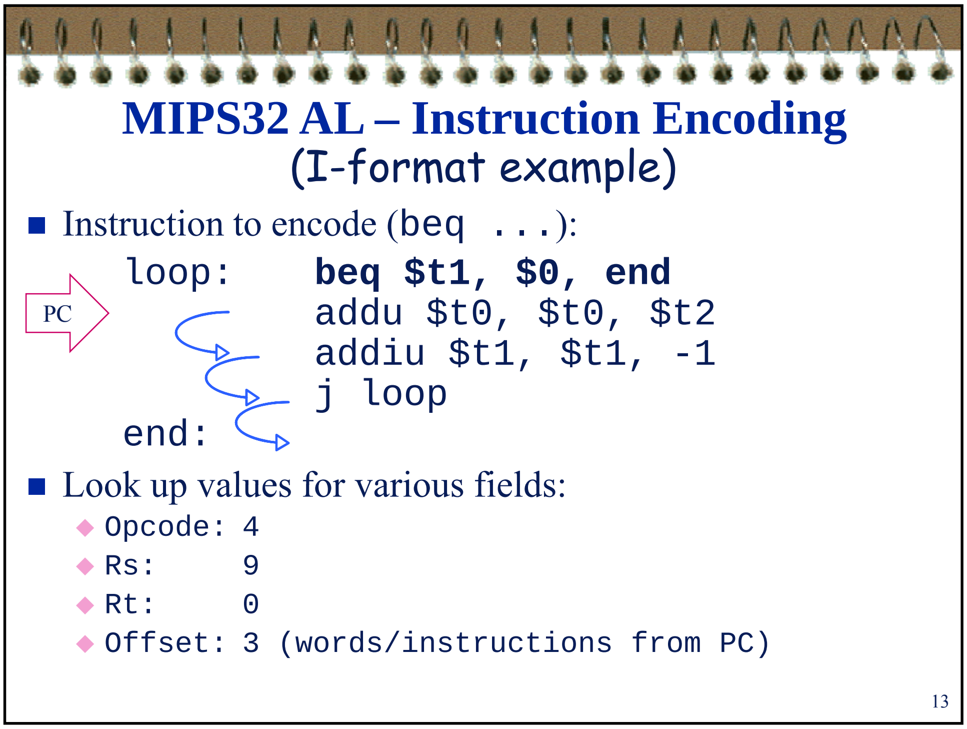
001000	**10110**	**10101**	**1111111111001110**

- Instruction in hex: **22D5FFCE**

MIPS32 AL – Instruction Encoding
(I-format example)

- Instruction to encode (`beq ...`):

```
loop:    beq $t1, $0, end
         addu $t0, $t0, $t2
         addiu $t1, $t1, -1
         j loop
end:
```

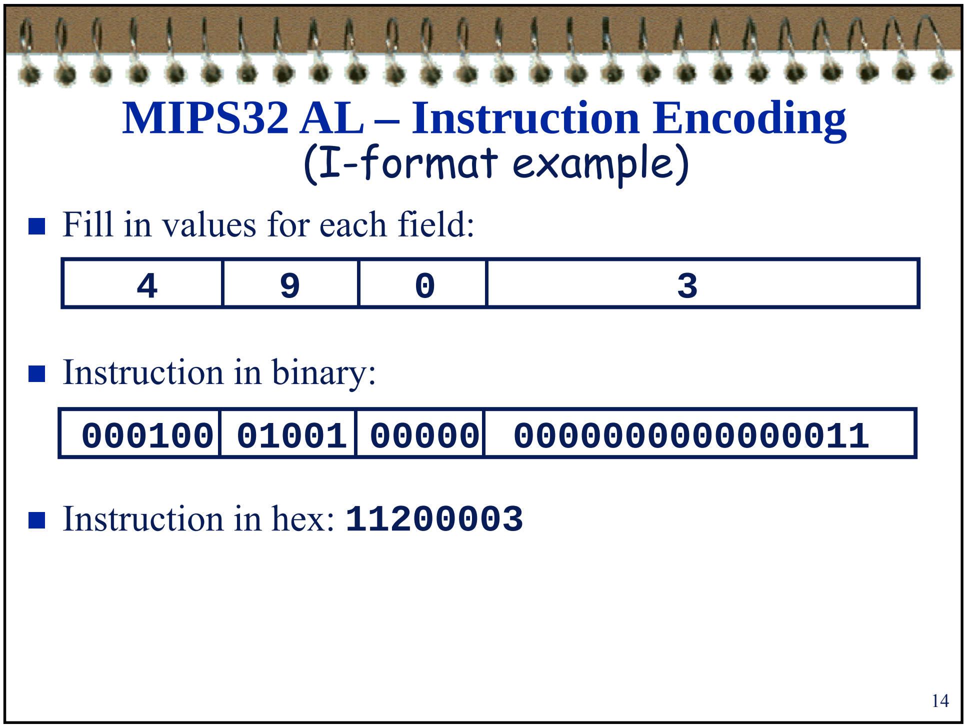


MIPS32 AL – Instruction Encoding
(I-format example)

- Instruction to encode (`beq ...`):

```
        loop:    beq $t1, $0, end
                 addu $t0, $t0, $t2
                 addiu $t1, $t1, -1
                 j loop
        end:
```

- Look up values for various fields:
  - `Opcode: 4`
  - `Rs:     9`
  - `Rt:     0`
  - `Offset: 3 (words/instructions from PC)`



MIPS32 AL – Instruction Encoding
(I-format example)

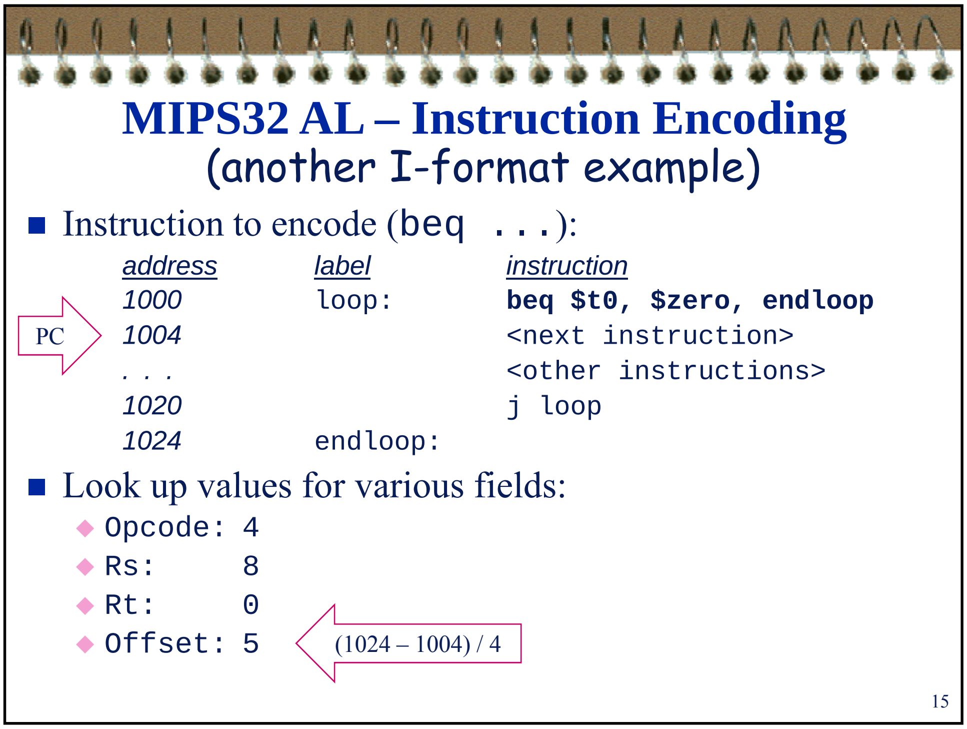
- Fill in values for each field:

4	9	0	3

- Instruction in binary:

000100	01001	00000	0000000000000011

- Instruction in hex: **11200003**



MIPS32 AL – Instruction Encoding
(another I-format example)

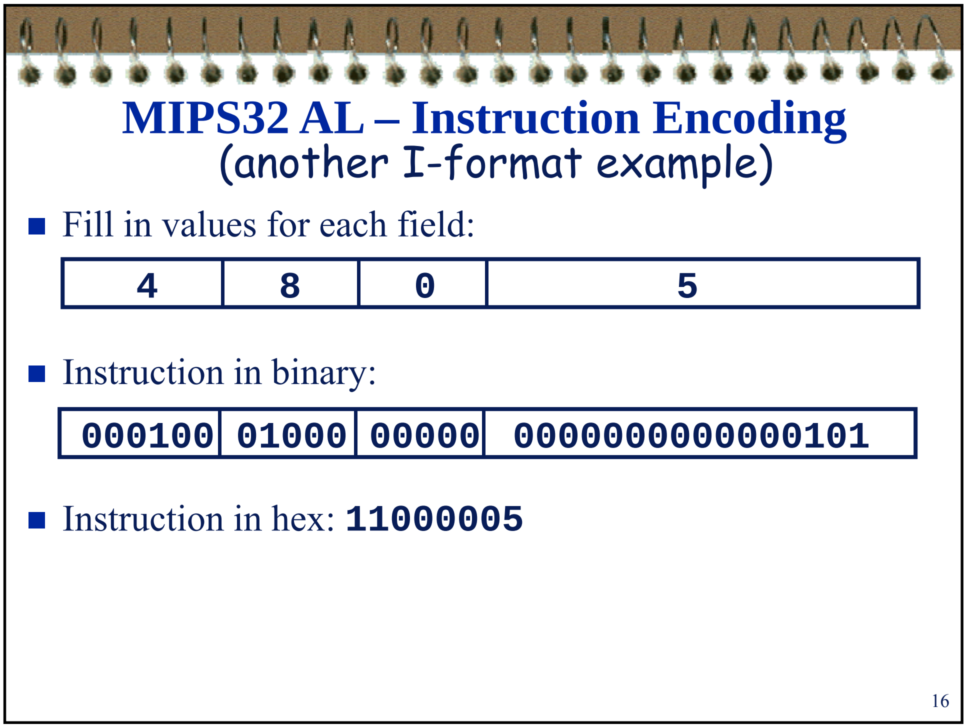
- Instruction to encode (`beq ...`):

address	*label*	*instruction*
1000	`loop:`	**`beq $t0, $zero, endloop`**
1004		`<next instruction>`
. . .		`<other instructions>`
1020		`j loop`
1024	`endloop:`	

PC

- Look up values for various fields:
  - `Opcode: 4`
  - `Rs:     8`
  - `Rt:     0`
  - `Offset: 5`

(1024 – 1004) / 4



MIPS32 AL – Instruction Encoding
(another I-format example)

- Fill in values for each field:

4	**8**	**0**	**5**

- Instruction in binary:

000100	**01000**	**00000**	**0000000000000101**

- Instruction in hex: **`11000005`**

MIPS32 AL – Instruction Encoding
(J-format example)

- Instruction to encode (`j ...`):

address	*label*	*instruction*
1000	`loop:`	`beq $t0, $zero, endloop`
1004		`<next instruction>`
. . .		`<other instructions>`
1020		**`j loop`**
PC → *1024*	`endloop:`	

- Look up values for various fields:
  - `Opcode: 2`
  - `target: 250` ← 1000 / 4

MIPS32 AL – Instruction Encoding
(J-format example)

- Fill in values for each field:

2	**250**

- Instruction in binary:

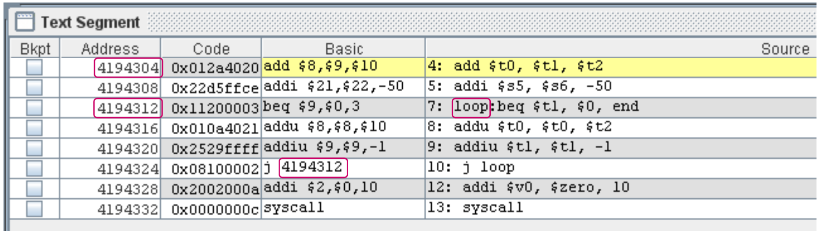
000010	**00000000000000000011111010**

- Instruction in hex: **`080000FA`**

MIPS32 AL – Instruction Encoding
(see them on MARS)

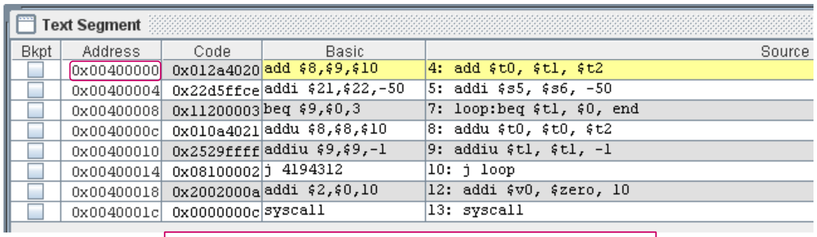
```
            .text
            .globl main
main:
            add $t0, $t1, $t2
            addi $s5, $s6, -50

loop:       beq $t1, $0, end
            addu $t0, $t0, $t2
            addiu $t1, $t1, -1
            j loop
end:
            addi $v0, $zero, 10
            syscall
```

MIPS32 AL – Instruction Encoding
(see them on MARS)



Text Segment

Bkpt	Address	Code	Basic	Source
□	4194304	0x012a4020	add $8,$9,$10	4: add $t0, $t1, $t2
□	4194308	0x22d5ffce	addi $21,$22,-50	5: addi $s5, $s6, -50
□	4194312	0x11200003	beq $9,$0,3	7: loop:beq $t1, $0, end
□	4194316	0x010a4021	addu $8,$8,$10	8: addu $t0, $t0, $t2
□	4194320	0x2529ffff	addiu $9,$9,-1	9: addiu $t1, $t1, -1
□	4194324	0x08100002	j 4194312	10: j loop
□	4194328	0x2002000a	addi $2,$0,10	12: addi $v0, $zero, 10
□	4194332	0x0000000c	syscall	13: syscall



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x012a4020	add $8,$9,$10	4: add $t0, $t1, $t2
□	0x00400004	0x22d5ffce	addi $21,$22,-50	5: addi $s5, $s6, -50
□	0x00400008	0x11200003	beq $9,$0,3	7: loop:beq $t1, $0, end
□	0x0040000c	0x010a4021	addu $8,$8,$10	8: addu $t0, $t0, $t2
□	0x00400010	0x2529ffff	addiu $9,$9,-1	9: addiu $t1, $t1, -1
□	0x00400014	0x08100002	j 4194312	10: j loop
□	0x00400018	0x2002000a	addi $2,$0,10	12: addi $v0, $zero, 10
□	0x0040001c	0x0000000c	syscall	13: syscall

NOTE: $2^{20} + 2 = 1048578 = 4194312 / 4$

MIPS32 AL – Instruction Encoding
(von Neumann quiz of sort)

- Which instruction has same representation as decimal 35?
  - `add $0, $0, $0`
  - `subu $s0, $s0, $s0`
  - `lw $0, 0($0)`
  - `addi $0, $0, 35`
  - `subu $0, $0, $0`
  - `Trick question! Instructions are not numbers`

MIPS32 AL – Instruction Encoding
(von Neumann quiz of sort)

- Which instruction has same representation as decimal 35?

`add $0, $0, $0`	0	0	0	0	0	32
`subu $s0, $s0, $s0`	0	16	16	16	0	35
`lw $0, 0($0)`	35	0	0	0		
`addi $0, $0, 35`	8	0	0	35		
`subu $0, $0, $0`	0	0	0	0	0	35

- None of the above (*i.e.*, this is a trick question since instructions are not numbers)