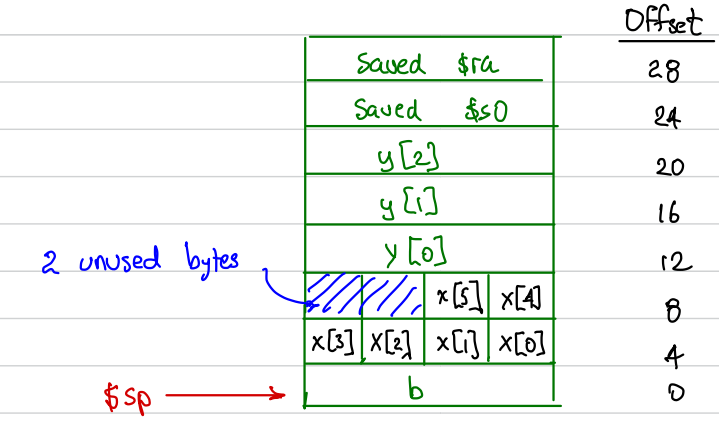
Copied from Monday ...

C-code :

```
int   a, b;
char  x[6];
int   y[3];

a = g(x, 6);
b = h(y, 3, a);
```

Stack frame:



Continuing ... [and midterm #1 next Wednesday, February 12 !]

Translations:

```
a = g(x, 6);      addi  $a0, $sp, 4      # $a0 = &x[0]
                  addi  $a1, $zero, 6    # $a1 = 6
                  jal   g
                  add   $s0, $zero, $v0  # a = ret. val.

b = h(y, 3, a);   addi  $a0, $sp, 12     # $a0 = &y[0]
                  addi  $a1, $zero, 3    # $a1 = 3
                  add   $a2, $zero, $s0  # $a2 = a
                  jal   h
                  sw    $v0, ($sp)       # b = ret. val.
```

Logical instructions

24 zeros

sll – shift left logical
srl – shift right logical

E.g., Let $t0 = [24 zeros]_0101_0111$_2$

Suppose: srl $t1, $t0, 3

$t1 gets: [24 zeros]_0000_1010$_2$

3 zeros shifted in

111 rolled off into oblivion

or, ori — logical OR

Recall x OR y

x y	x OR y
0 0	0
0 1	1
1 0	1
1 1	1

The or and ori instructions perform 32 simultaneous bit-wise OR operations on 32 pairs of bits:

E.g., Let $t0 = 1010_[24 zeros]_0011

$t1 = 0001_[24 zeros]_0101

$t2 = 1011_[24 zeros]_0111 ← [correction, Feb 11!]

or $t2, $t0, $1

ori — one source is a GPR, other is a 16-bit constant

e.g., ori $t4, $t3, 0x895a

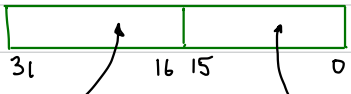
Let $t3 = 0xab00_0034

$t3 = 1010_1011_0000_0000_0000_0000_0011_0100

16 zeros _1000_1001_0101_1010

$t4 = 1010_1011_0000_0000_1000_1001_0111_1110

lui instruction — load upper immediate



copies 16-bit constant to here

zeros

e.g., lui $t0, 0xf7b3 ⟶ $t0 = 0xf7b3_0000

Putting 32-bit constants in a GPR

Can't be done in one instruction – can only do 16 bits at a time.
E.g., a commonly used pseudo-instruction:

```
li    $t1, 0xe9b1ae09        (load immediate)
```

↑ sticks this 32-bit number into $t1

Here's what's actually implemented:

```
lui   $t1, 0xe9b1
ori   $t1, $t1, 0xae09
```

li is identical to la.

and, andi, nor, xor, xori — logical AND, NOR, and XOR functions.

Next: encoding jump and branch instructions

Let's check out the encoding of beq, bne, j, jal, jr.

beq / bne:

- 6: opcode for beq / bne
- 5: registers to compare
- 5: registers to compare
- 16 bits: how many instructions words to skip forward or backward.

E.g.,

```
L1:   beq   $t0, $t1, L2      ← (1)
      lw    $t2, ($s0)
      add   $t3, $t3, $t2
      beq   $zero, $zero, L1  ← (2)
L2:   sw    $t3, ($s0)
```

Encoding instruction (1):

000100 _ 01000 _ 01001 _ 0000_0000_0000_0011$_2$

beq opcode $t0 $t1

constant = 3. From the instruction immediately following the beq (i.e., the lw instruction), count 3 words ahead to L2.

000100 _ 01000 _ 01001 _

beq $t0 $t1

opcode

Encoding instruction (2)

000100 _ 00000 _ 00000 _ 1111_1111_1111_1100$_2$

beq $zero $zero

constant = -4 (2's-complement). From the sw instruction at L2, count 4 words back to L1.