(Micro computer without interlocked Pipeline stages)

## MIPS instruction Set architecture

variables $s0, $s1 $s2, $s3, $s4

↳ that corresponds to variables in C and Java

$t0, $t1, $t2 ... for temporary registers

To command a computer's hardware, you must speak its language.

The words of the computer language are called instructions

And its vocabulary is called an instruction set.

Common goal of a computer designers:
to find a language that makes it easy to build the hardware and the compiler while maximising performance and minimising cost.

Design Principle 1: Simplicity favours regularity
2. Smaller is faster
3. Make the common case fast → constants.
4. Good design demands compromise

Date __/__/____ Page No: ____

Q. A statement contains 5 variables f, g, h, i, j

$f = (g+h) - (i+j)$:

what might a C compiler produce?

→ Only one operation is performed per MIPS instruction.

→ first mips instruction ~~creates~~ calculates the sum of g and h.

→ We must place the result somewhere, so the computer creates a temporary variable, called to

```
add to, g, h
add t1, i, j
sub f, to, t1
```

## MIPS assembly language

| | | | |
|---|---|---|---|
| Arithmetic | add | add a, b, c | a = b+c |
| | subtract | sub a, b, c | a = b-c |

# Operands of the computer hardware

Date / / Page No:

→ Unlike programs in high level language, the operands of arithmetic instructions are restricted; they must be from a limited number of special locations built directly in hardware called registers.

→ Registers are bricks of computer construction

→ The size of register in the MIPS architecture is 32 bits.

Design Principle :- Smaller is faster

A very large number of registers may increase the clock cycle time simply because it takes electronic signals longer when they must travel farther.

→ Arithmatic operations only occur in registers in MIPS instructions

↳ data transfer instructions

Load → lw

Load data from memory to register

Q. Let's assume that A is an array of 100 words and that the compiler has associated the variables g and h with registers $S_1$ and $S_2$ as before. Lets also assume that starting address, or base address, of the array is in $S_3$. Compile this C assignment statement:

```
g = h + A[8];
```

Answer: Although there is a single operation in this assignment statement, one of the operands is in memory, so we must transfer A[8] to a register.

```
lw  $t0, 8($s3)
add $s1, $s2, $t0
```

The constant in a data transfer instruction is called the offset, and the register added to form the address is called base register.

sw → Store.

Date __/__/____ Page No: ____

Q. Assume variable h is associated with register $s2 and the base address of the array A is in $s3. What is the MIPS assembly code for the C assignment statement below?

```
A[12] = h + A[8];
```

```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)
```

Date / / Page No:

→ Many programs have more variables than computer have registers. Consequently, the compiler tries to keep the most frequently used variables in registers and place the rest in memory. The process of putting less commonly used variables (or those needed later) into memory called spilling registers.

To add constant in register

```
lw $t0, AddrConstant4($s1)
add $s3, $s3, $t0
```

OR

```
add $s3, $s3, 4
```

↳ Immidiate instructions illustrate the third hardware design principle,

Make the common case fast.

→ Constant operands occur frequently, and by including constants inside arithmetic instructions, they are much faster than if constants were loaded from memory.

| Name | Example | |
|---|---|---|
| 32 registers | $s0, $s1 ... $t0, $t1 ... | fast locations for data, data must be in registers to perform arithmetic. |
| $2^{30}$ memory | Memory[0] Memory[4] Memory[42...] | Access only by data transfer instruction in MIPS. Memory holds data structures, arrays, & spilled registers. |

Intel IA-32

MIPS-32, MIPS-64

## Representing Instructions in the computer

Instructions kept in computer as a series of high and low electronic signals and may be represented as numbers.

Each piece of an instruction can be considered as individual number, and placing these number side by side forms

Date __/__/____ Page No:

the instruction.

Since registers are part of almost all instructions, there must be a convention to map registers names into numbers.

In MIPS assembly language, registers $s0 to $s7 map onto registers 16 to 23 and registers $t0 to $t7 map onto registers 8 to 15.

Hence $s0 → Register 16
$s1 → 17
$s2 → 18
⋮
$t0 → 8
$t1 → 9

add $t0, $s1, $s2

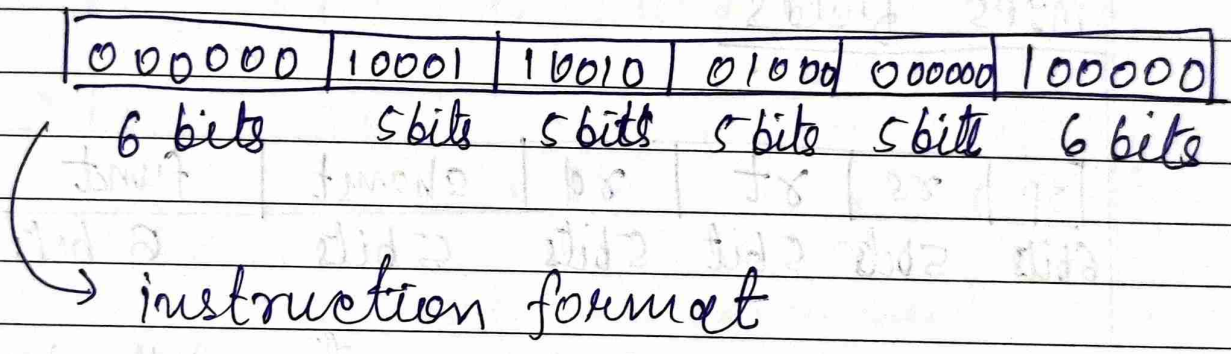
| 0 | 17 | 18 | 8 | 0 | 32 |
|---|---|---|---|---|---|

Each of the segment of an instruction is called a field.

The first and last fields containing (0 and 32 in this case) in combination tell the MIPS computer that this instruction performs addition.

Second field gives the number of register that is the first source operand ($17 = \$s_1$) and third field gives other source operand for the addition ($18 = \$s_2$) fourth field contains number of register that is to recieve the sum. ($8 = \$t_0$)

fifth field is unused so it is set to 0.




→ instruction format

Numeric version of instruction
→ machine language.

& sequence of such instructions
→ machine code.

Hexadecimal (base 16)

OMKALA NOTEBOOK

Date __/__/____ Page No. ____

Convert the following hexadecimal to binary.

eca6420 hex

Convert binary to hexadecimal.

0001 0011 0101 0111 1001 1011 1101 1111 two

MIPS fields

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

op :- opcode / basic operation of the instrn

rs : first register source operand
rt : the second register source operand.
rd : the register destination operand.
It gets the result of the operation.

shamt : Shift amount.
(for shift instruction)

funt : function. This field selects the specific variant of the operation in op field & is sometimes called function code.

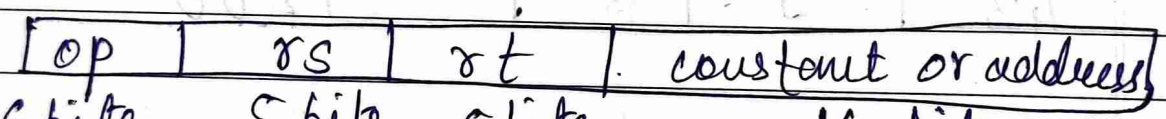
Date / / Page No:

We have conflict in between the desire to keep all instructions the same length & desired to have a single instruction format. This leads us to the final hardware design principle.

"Good design demands good compromises".

R-type → for register

I-type → (for immediate)




16 bit address means.

$\pm 2^{15}$ or 32,768 bytes ($\pm 2^{13}$)

or

8192 words.

---

Q If $t1, has base of the array A & $s2 corresponds to h, the assignment statement

A[300] = h + A[300]

Date __/__/____ Page No:____

```
lw   $to, 1200($t1)
add  $to, $s2, $to
sw   $to, 1200($t1)
```

What is mips machine language.

| op | rs | rt | rd | address/shmt | funct |
|---|---|---|---|---|---|
| 35 | 9 | 8 | 1200 | | |
| 0 | 18 | 8 | 8 | 0 | 32 |
| 43 | 9 | 8 | 1200 | | |

Logic Operations

| | C | Java | |
|---|---|---|---|
| shift left | << | << | sll |
| shift right | >> | >>> | srl |
| Bit-by-Bit AND | & | & | and, andi |
| Bit-by-Bit OR | \| | \| | or, ori |
| Bit-by-Bit NOT | ~ | ~ | nor |

shift left logical

$s0

They move all the bits in a word to the left or right, filling the emptied bits with 0s.

0000 0000 0000 0000 1001 $_{two}$ = 9 $_{ten}$

←

0000 0000 0000 0000 1001 0000 $_{two}$ = 144 $_{ten}$

The dual of shift left is shift right.

sll $t2, $s0, 4 # reg $t2 = reg $s0 << 4 bits

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 0 | 0 | 16 | t2 | 4 | 0 |

and $t0, $t1, $t2

reg $t0 = reg $t1 & reg $t2

or $t0, $t1, $t2

reg $t0 = reg $t1 | reg $t2

nor $t0, $t1, $t3

reg$t0 = ~(reg$t1 | reg$t3)

andi $s1, $s2, 100

ori $s1, $s2, 100

## Instructions for making decisions.

beq register 1, register 2, L1

↳ go to the statement labelled L1 if the value in register 1 equals the value in regis

beq means branch if equal

bne ↳ branch if not equal

In the following code segment f, g, h, i & j are variables. If the five variables f through j corresponds to the five registers $s0 through $s4 what is the compiled MIPS code for this C if - statement?

```
if (i == j)
f = g + h;
else
f = g - h;
```

Unconditional branch

```
bne $s3, $s4, Else
add $s0, $s1, $s2   (skipped if i ≠ j)
j Exit  # go to exit
Else: sub $s0, $s1, $s2   (skipped if i == j)
Exit.
```

---

## Loops

Compiling a while loop in C

```
while (save[i] == k)
    i += 1;
```

Assume that i & k corresponds to registers $s3 & $s5 and the base of the array save is in $s6. What is MIPS assembly code corresponding to

this C segment ?

```
Loop: sll $t1, $s3, 2
                # temp reg $t1 = 4 * i
```

> shifting left by i bits gives the same results as multiplying by $2^i$

To get the address of save[i], we need to add $t1 and the base of save in $s6

```
add $t1, $t1, $s6   # t1 = address of save[i]
lw $t0, 0($t1)      # Temp reg $t0 = save[i]
bne $t0, $s5, Exit  # goto Exit if save[i] ≠ k
```

The next instr^n add 1 to i

```
add $s3, $s3, i     # i = i + 1
j Loop              # go to Loop
Exit:
```

Date / / Page No:

slt $t0, $s3, $s4

↳ set to less than

t0 = 1 if $s3 < $s4 otherwise t0 = 0

slti $t0, $s2, 10

## Supporting Procedures in Computer Hardware

$a0 $a1 $a2 $a3 → four argument register

$v0 $v1 → two value registers in which to return values.

$ra: → one return address register to return to the point of origin.

jump and link instruction: An instruction that jumps to an address and simultaneously saves the address of the following instruction in register (ra)

(jal)

jal ProcedureAddress

jr ra → jump register

Using more registers

↳ spilling in memory

$sp

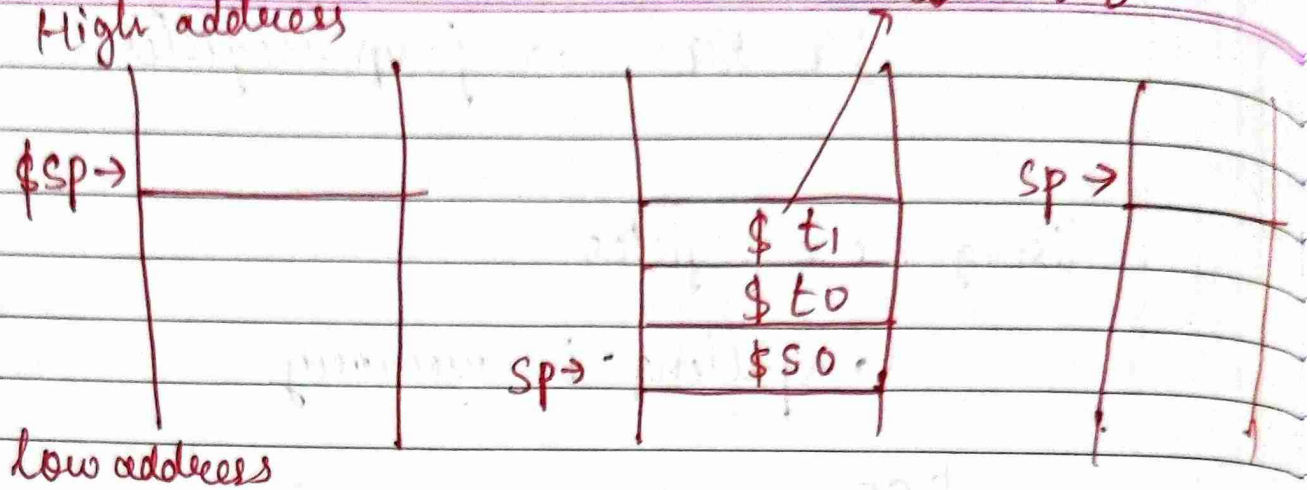
---

Q.
```
int leaf_example (int g, int h, int i,
                  int j)
{
  int f;
  f = (g+h) - (i+j);
  return f;
}
```

g, h, i, j corresponds to $a0, $a1, $a2, $a3

f corresponds to $s0

```
leaf_example :
    add $sp, $sp, -12
        # adjust stack to make room for 3
          items
    sw $t1, 8($sp)
    sw $t0, 4($sp)
    sw $s0, 0($sp)
```




(a) before (b) during (c) after

```
add $t0, $a0, $a1
        # register t0 contains c+b
add $t1, $a2, $a3
        # register t1 contains i+j
sub $s0, $t0, $t1
        # f=$t0-$t1
add $v0, $s0, $zero
        # returns f ($v0=$s0+0)
```

Before returning, we restore the three old values of the registers we saved by "popping" them from stack.

```
lw $s0, 0($sp)
lw $t0, 4($sp)
lw $t1, 8($sp)
add $p, $sp, 12

jr $ra
```

## Nested

Q.

Let's take a recursive procedure, showing nested Procedure linking

```
int fact (int n)
{
    if (n<1) return(1);
        else return (n * fact(n-1));
}
```

what is MIPS assembly code?

```
fact:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $a0, 0($sp)

    slti $t0, $a0, 1    # test for n<1
    beq $t0, $zero, L1

                        # if n>=1, ~~argument~~
    go to L1            ~~get (n-1)~~
```

Date __/__/____ Page No: ____

```
add $v0 , $zero, 1  #return 1
addi $sp, $sp, 8
jr $ra
```

If n is not less than 1, argument n decremented and fact is called again.

```
L1 : add $a0, $a0, -1
     jal fact
                    # n >= 1 argument gets (n-1)
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
```

Next, value register $v0 gets the product of old argument $a0 and the current value of the value register

```
mul $v0, $a0, $v0  #return n*fact(n-1)
jr $ra
```

Date __/__/____ Page No: ____

| | | |
|---|---|---|
| $zero | 0 | no |
| $v0 - $v1 | 2-3 | no |
| $a0 - $a3 | 4-7 | no |
| $t0 - $t7 | 8-15 | no |
| $s0 - $s7 | 16-23 | yes |
| $t8 - $t9 | 24-25 | no |
| $gp | 28 | yes |
| $sp | 29 | yes |
| $fp | 30 | yes |
| $ra | 31 | yes |

Q. What is assembly code language
00af8020 hex

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 000000 | 00101 | 01111 | 10000 | 00000 | 100000 |

add $s0, $a1, $t7

Allocating Space for new data on the Stack

The stack is also used to store variables that are local to the procedure that do not fit in register, such as local arrays or structures.
The segment of the stack containing a procedures saved registers and local variables is called a procedure frame or activation record.

$fp → some procedures used the frame pointer ($fp) to point to the first ~~frame~~ word of the frame of a procedure

Date __/__/____ Page No: ____

# 32 bit Immidiate Operands

lui $t0, 255

| 001111 | 00000 | 01000 | 0000 0000 1111 1111 |
|---|---|---|---|

↓ contents of register $t0 after executing lui $t0, 255

| 0000 0000 1111 1111 | 0000 0000 0000 0000 |
|---|---|

---

What is mips assembly code to load this 32-bit constant into register $S0 ?

0000 0000 0011 1101 0000 1001 0000 0000

lui $s0, 61

$s0 value after this

0000 0000 0011 1101 0000 0000 0000 0000

ori $s0, $s0, 2304

0000 0000 0011 1101 0000 1001 0000 0000

Date / / Page No:

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit:
```

If we assume we place the ~~starting~~ loop starting at location 80000 in memory, what is mips machine code for this loop?

| | | | | | | |
|---|---|---|---|---|---|---|
| 80000 | 0 | 0 | 19 | 9 | 4 | 0 |
| 80004 | 0 | 9 | 22 | 9 | 0 | 32 |
| 80008 | 35 | 9 | 8 | 0 | | |
| 80012 | 5 | 8 | 21 | 2 | | |
| 80016 | 8 | 19 | 19 | 1 | | |
| 80020 | 2 | 200008 | | | | |
| 80024 | | | | | | |
| ⋮ | | | | | | |