MIPS assembly language

Pioneered RISC (reduced instruction set architecture) in the 1980s

The most widely taught assembly language

Easy to transition from MIPS to ARMv8

Recently acquired by Wave Computing, will be aligned with RISC-V

Hello World in MIPS

Two sections:

.data - for static data

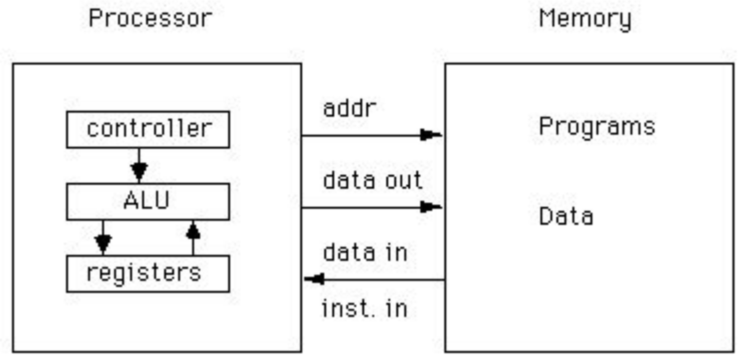
.text - for code

- Program ends with a syscall to end the program
- Think of this like return(0) in C
- Syscalls need the call number in $v0

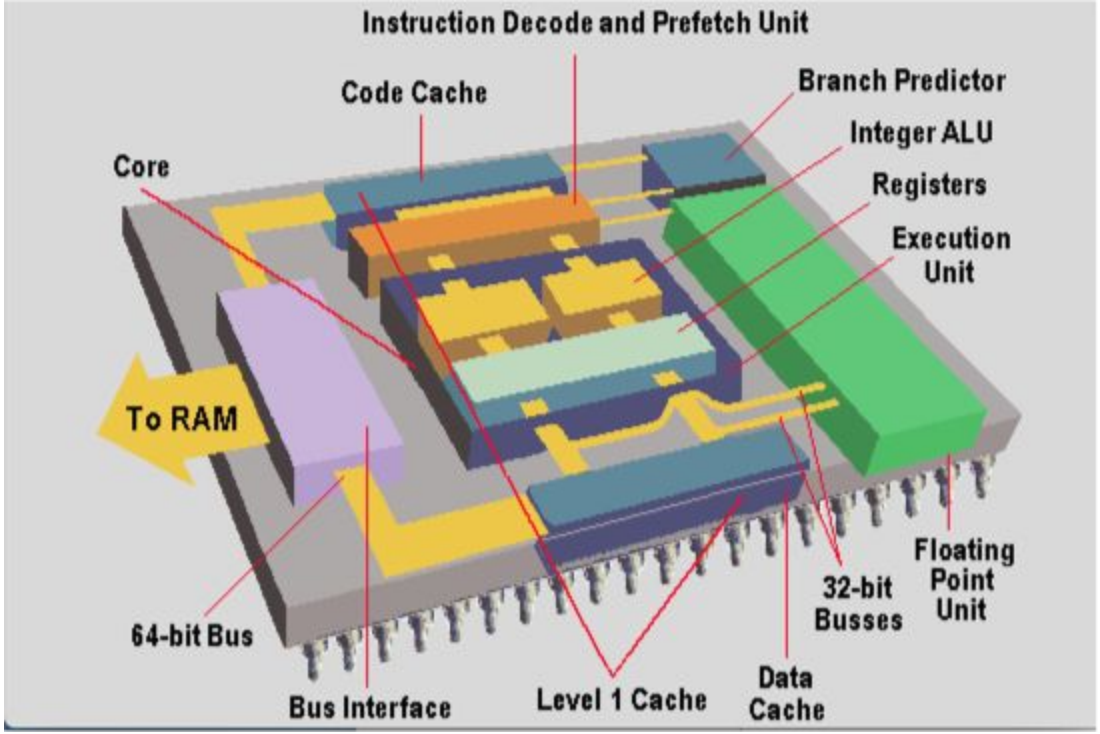
```
# Hello World

.data
msg:     .asciiz          "Hello world!"

.text
main:
         li $v0, 4
         la $a0, msg
         syscall          # syscall to display message

         # exit program
         li $v0, 10
         syscall

```

registers

- In a higher-level language we use variables to hold data
- In assembly language we use registers to hold data

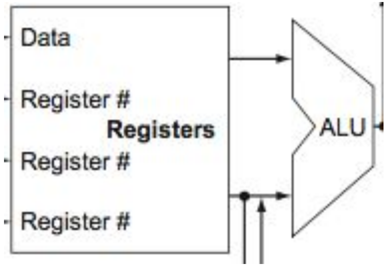


Registers in the Pentium



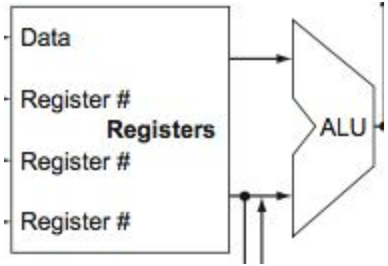
registers

- Used to hold data
- Used to hold addresses



registers

- Registers hold data for operations



Generic processor:

```
add R1, R2, R3
```

MIPS:

```
add $t0, $t1, $t2  # t0 = t1 + t2
```

MIPS instruction format

- All arithmetic/logic instructions have this format:

```
Opcode    operand, operand, operand
```

- the first operand is the destination

- the last two are source operands

- opcode specifies what action needs to happen

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

MIPS registers

- MIPS has 32 registers
- Each register is 32-bits (1 word, 4 bytes)
- For operands, we most often use:
- The “temporary” registers $t0 - $t9
- The “saved” registers $s0 - $s7
- The “zero” register $zero which always contains 0 and is read-only

.data

We defined and initialized 4 words (integers)

This is somewhat like declaring a variable, but there is no “type”

A memory location can contain integers, floats, characters, it’s up to you to remember what it is

```
# example 1 load a and b, store into c and d

        .data
a:      .word   3
b:      .word   4
c:      .word   9
d:      .word   9

        .text
main:
        lw      $t1, a          # load
        lw      $t2, b
        sw      $t1, c          # store
        sw      $t2, d

exit:
        li      $v0, 10         # terminate program
        syscall
```

MIPS program form

- Labels end with :
- Later we'll use these for jumps

- Program ends with a syscall to end the program
- Think of this like return(0) in C
- Syscalls need the call number in $v0

```
# example 1 load a and b, store into c and d

        .data
a:      .word   3
b:      .word   4
c:      .word   9
d:      .word   9

        .text
main:
        lw      $t1, a          # load
        lw      $t2, b
        sw      $t1, c          # store
        sw      $t2, d

exit:
        li      $v0, 10         # terminate program
        syscall
```

Load-Store (data transfer) instructions

.text

“lw” loads (copies) a word from memory into a register

“sw” stores (copies) a word from a register into memory

MIPS is a load-store architecture

- Cannot “add c, a, b”
- Cannot “sw c, a”

```
# example 1 load a and b, store into c and d

        .data
a:      .word   3
b:      .word   4
c:      .word   9
d:      .word   9

        .text
main:
        lw      $t1, a          # load
        lw      $t2, b
        sw      $t1, c          # store
        sw      $t2, d

exit:
        li      $v0, 10         # terminate program
        syscall
```

Load-Store (data transfer) instructions

.text

- Array version of previous program
- “la” loads address

Load/store instruction format:

```
lw  $t0, 8($t1)
```

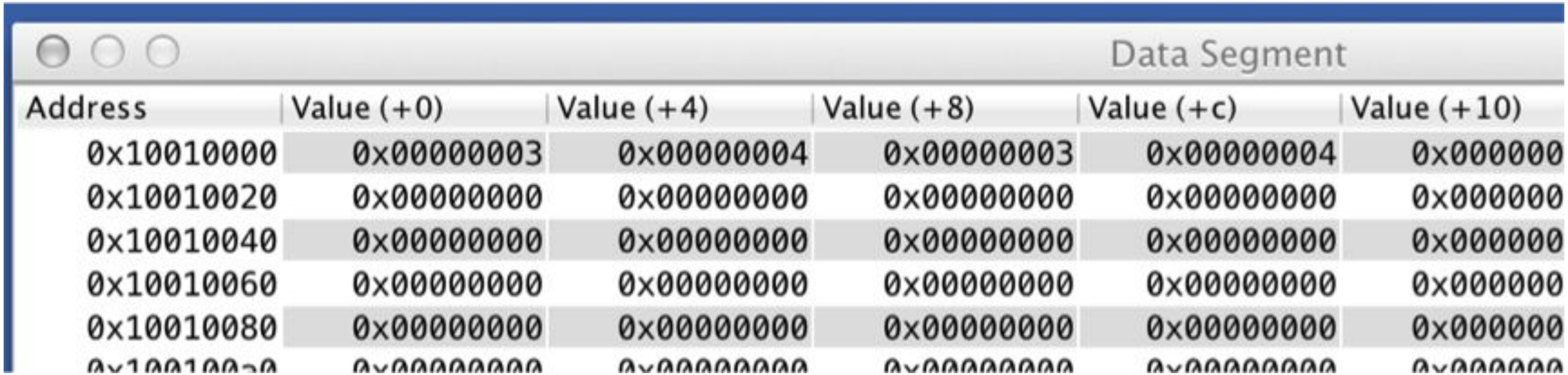
Load memory location $t1+8 into $t0

```
        .data
arr1:   .word   3, 4
arr2:   .word   9, 9

        .text
main:
        la      $t1, arr1
        la      $t2, arr2
        lw      $t0, ($t1)
        sw      $t0, ($t2)
        lw      $t0, 4($t1)
        sw      $t0, 4($t2)

exit:   li      $v0, 10
        syscall
```

MARS (MIPS Assembler and Runtime Simulator)

- Registers on the right
- Toggle Edit/Execute
- Drop-down buttons on bottom left to expand window
- Edit file, save with .asm
- Assemble icon on top

After the run


Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010000	0x00000003	0x00000004	0x00000003	0x00000004	0x000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x000000

Practice

Modify program to swap the contents of a and b

```
# example 1 load a and b, store into c and d

        .data
a:      .word   3
b:      .word   4
c:      .word   9
d:      .word   9

        .text
main:
        lw      $t1, a          # load
        lw      $t2, b
        sw      $t1, c          # store
        sw      $t2, d

exit:
        li      $v0, 10         # terminate program
        syscall
```

ADD and SUB instructions

```
add   rd, rs, rt             # rd = rs + rt

sub     rd, rs, rt           # rd = rs - rt

addi  rd, rs, constant       # rd = rs + constant
```

Simple addi example

```
# simple addi example
.data
var1:   .word   4               # variable var1 = 4
.text
main:   li      $t1, 2          # $t1 = 2
        addi    $t1, $t1, 3     # now $t1 = 2 + 3
        addi    $t1, $t1, 4     # now $t1 = 2 + 3 + 4
        sw      $t1, var1       # store $t1 in var1

        # exit
        li  $v0, 10
        syscall
```

Practice

Write a program to load 3 integers, stored as var1, var2, and var3, into registers $t1, $t2, and $t3. Reserve a word for 'result' and initialize it to 9.

compute $t1 + $t2 - $t3, this will take 2 instructions

Store the result in 'result'

More practice

Convert this C expression into MIPS code

```
result = (var2 - var1) + (var3 - var1)
```

syscalls

The syscall instruction calls the operating system to perform some task that a program would not have permission to do, such as I/O.

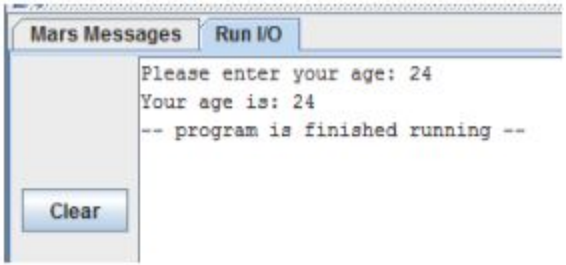
Supported syscalls in MIPS:

- http://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html

Program termination syscall

```
li $v0, 10 # terminate program
syscall
```

I/O syscall demo




```
# MARS syscalls

.data
age:     .word   0
msg1:    .asciiz "Please enter your age: "
msg2:    .asciiz "Your age is: "

.text
main:
         # prompt user for age
         la      $a0, msg1
         li      $v0, 4
         syscall
         # get int from user
         li      $v0, 5
         syscall
         sw      $v0, age

         # echo data to user
         la      $a0, msg2
         li      $v0, 4
         syscall
         lw      $a0, age
         li      $v0, 1
         syscall

exit:    li      $v0, 10
         syscall
```

summary

- What are registers?
- Name a MIPS register and describe it.
- What kind of data can it contain? Integer? Characters? Address?
- What are opcodes?
- What are operands?
- What kinds of operands have we seen?

Coding Practice

For next class, write a program to:

- get the user's name
- get the user's age
- get a neighbor's name
- get the neighbor's age
- print a message echoing both names and the combined years of wisdom
- print a message with the difference in ages