CMPE 200
Computer Architecture & Design

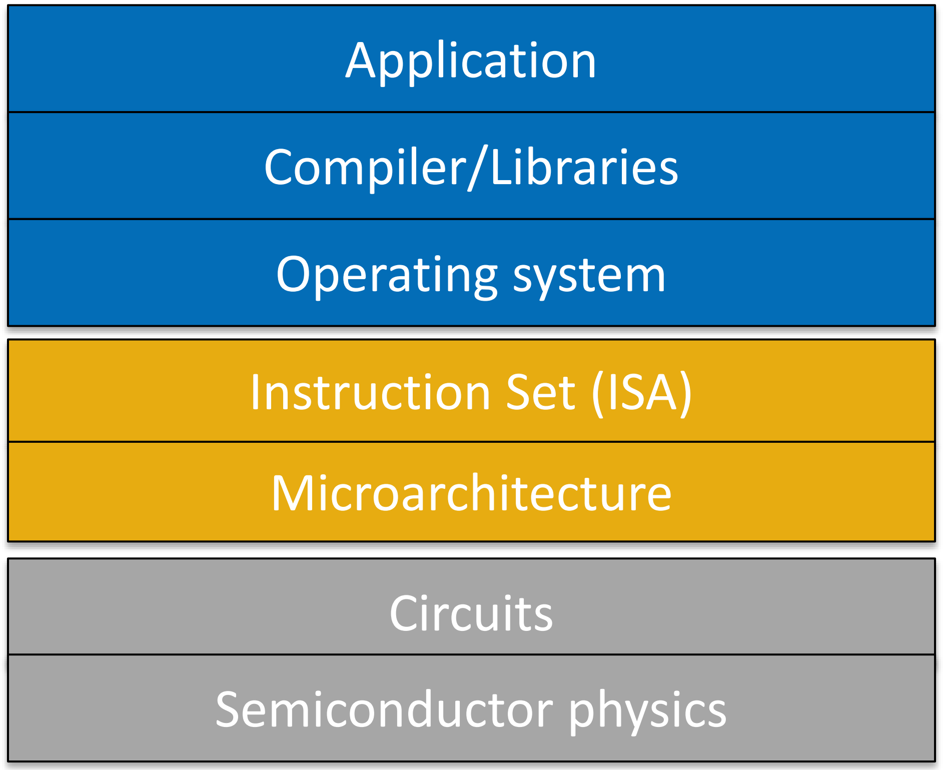
Lecture 2. Processor Instruction Set Architecture & Language (1)

Haonan Wang

SJSU SAN JOSÉ STATE UNIVERSITY

Instruction Set Architecture & Microarchitecture

- **Instruction Set Architecture**
  - the programmer’s view of the computer, defined by a set of instructions that each specifies an operation and its operand(s)
- **Microarchitecture**
  - the way that the ISA is implemented in hardware



What Does the ISA Deal With Specifically?

Example: a C program that reads two integer values from "file.txt" file and prints the sum of them.

```
#include <stdio.h>
#include <string.h>

int numbers[2];

void myfunction(void)
{
   FILE *fp;
   int size = 2;
   int sum = 0;

   /* Open file for reading */
   fp = fopen("mynumbers.txt", "r");

   /* Read and display data */
   fread(numbers, sizeof(int), size, fp);
   fclose(fp);
   sum = numbers[0] + numbers[1];
   printf("Sum = %d\n", sum);
}

int main (void) {
   myfunction();
   return(0);
}
```

Processor (CPU)

Understands and executes each line of the code.

Uses fast on-chip memories

Memory (DRAM)

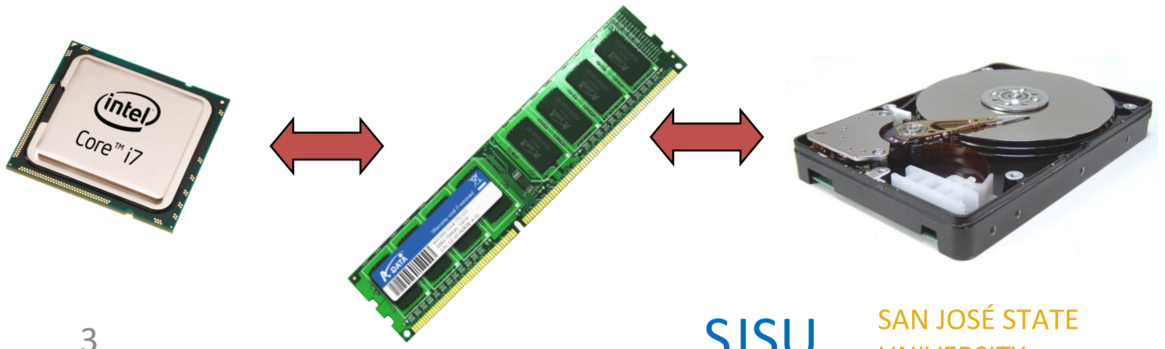
Provides operands to CPU

(*fp, size, sum, numbers[2])

Storage (HDD)

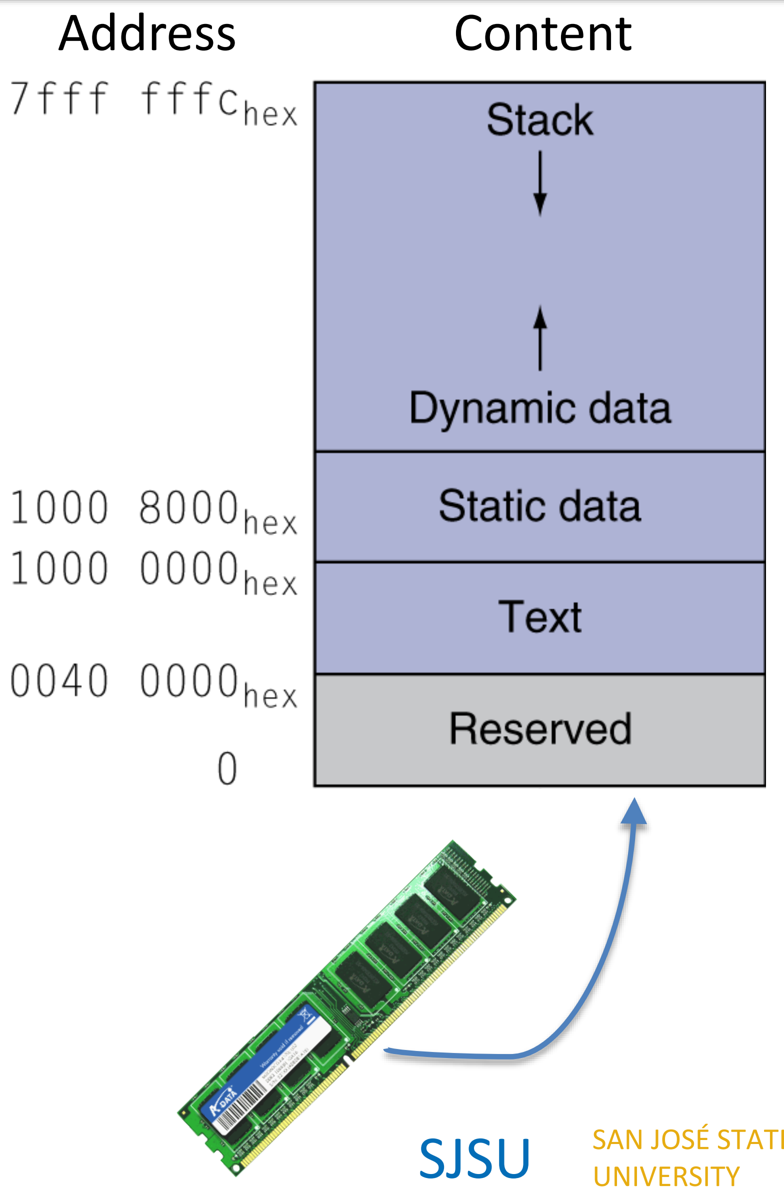
Provides file inputs and program code

(file.txt)



Memory Layout

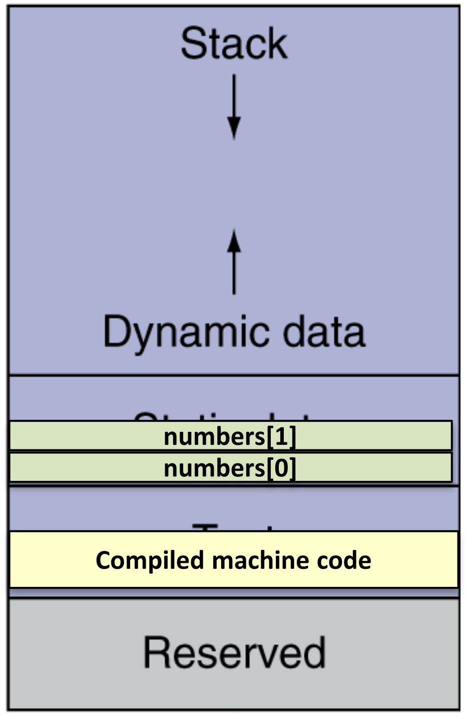
- **Text:** program code
- **Static data:** global variables
  - e.g., static variables in C, constant arrays and strings
- **Dynamic data:** heap
  - e.g., malloc in C, new in Java
  - Grows from bottom (lower address) to top (higher address)
- **Stack:** temporal storage for functions
  - e.g., return address of sub-functions, local variables
  - Grows from top to bottom

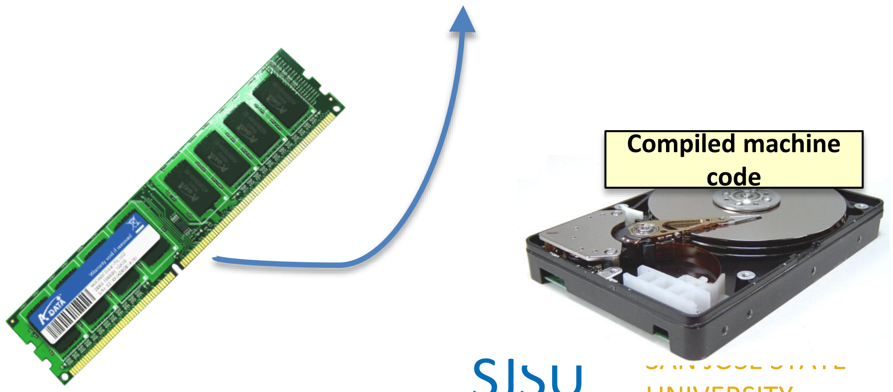


SJSU SAN JOSÉ STATE UNIVERSITY

Memory Layout

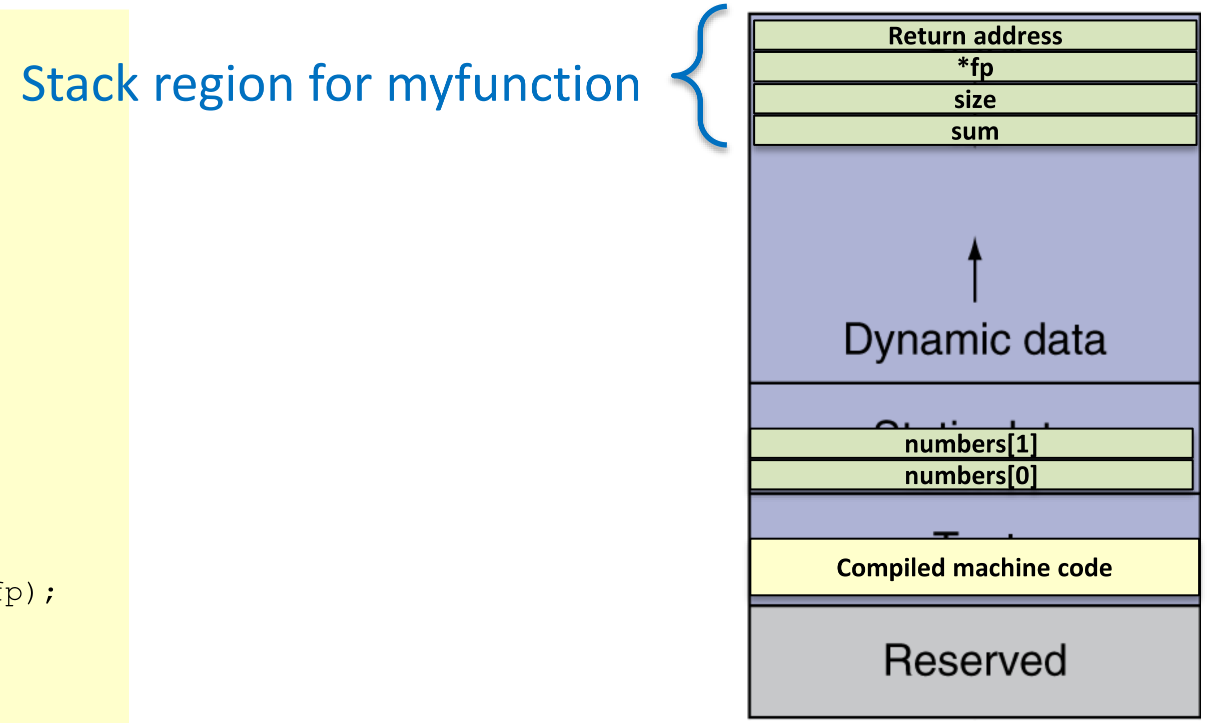
```
#include <stdio.h>
#include <string.h>

int numbers[2];

void myfunction(void)
{
   FILE *fp;
   int size = 2;
   int sum = 0;

   /* Open file for reading */
   fp = fopen(“mynumbers.txt", “r");

   /* Read and display data */
   fread(numbers, sizeof(int), size, fp);
   fclose(fp);
   sum = numbers[0] + numbers[1];
   printf(“Sum = %d\n”, sum);
}

int main (void) {
   myfunction();
   return(0);
}
```

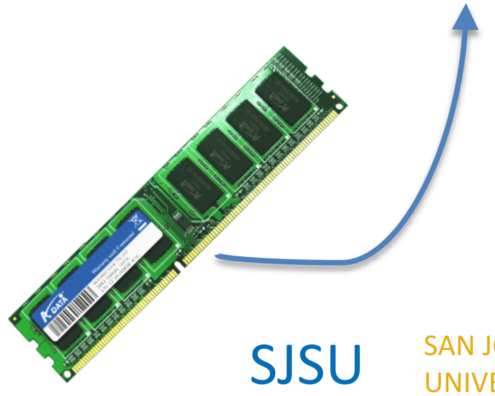




Memory Layout

```
#include <stdio.h>
#include <string.h>

int numbers[2];

void myfunction(void)
{
   FILE *fp;
   int size = 2;
   int sum = 0;

   /* Open file for reading */
   fp = fopen(“mynumbers.txt", “r");

   /* Read and display data */
   fread(numbers, sizeof(int), size, fp);
   fclose(fp);
   sum = numbers[0] + numbers[1];
   printf(“Sum = %d\n”, sum);
}

int main (void) {
   myfunction();
   return(0);
}
```

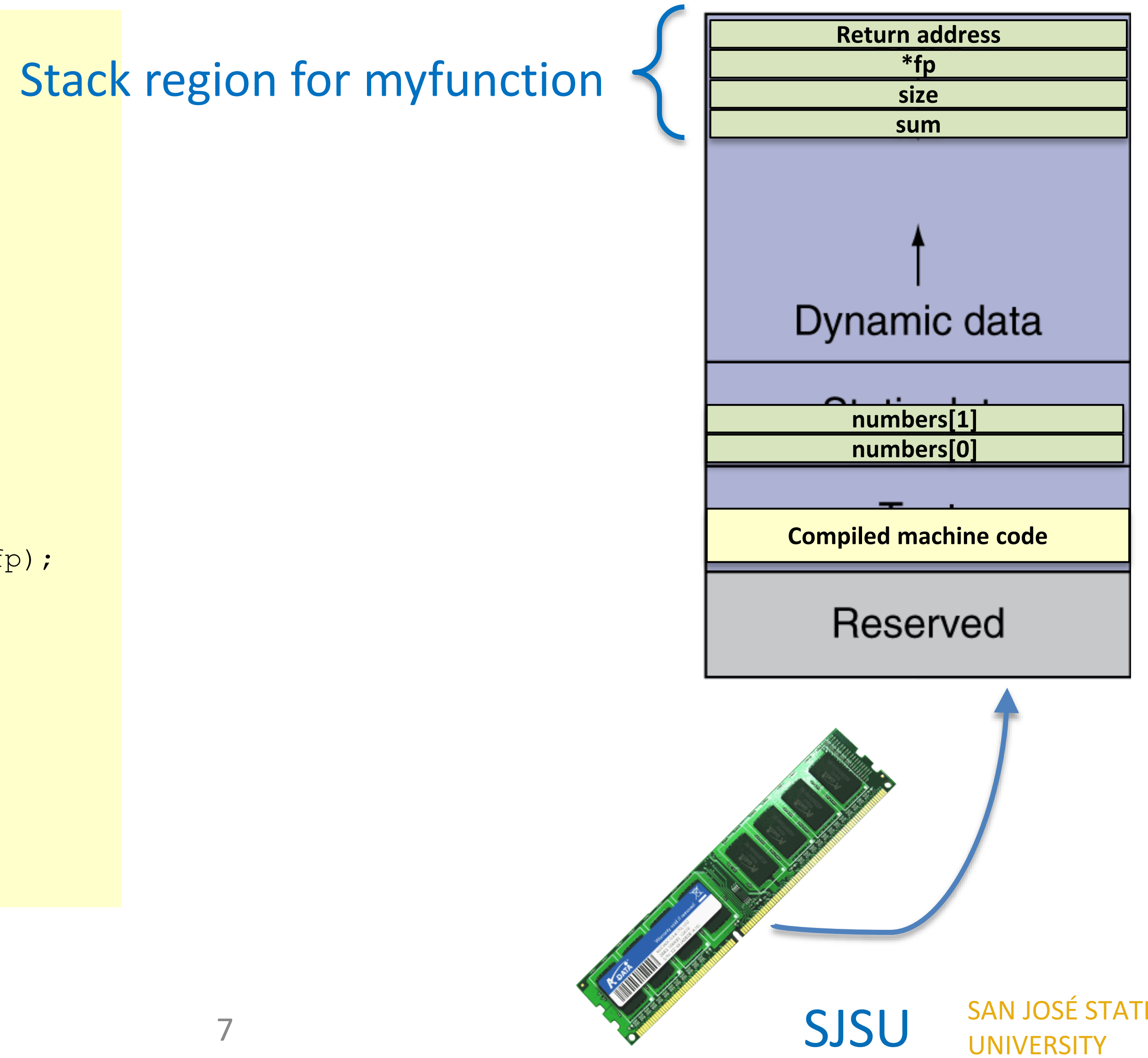




SJSU SAN JOSÉ STATE UNIVERSITY

Memory Layout

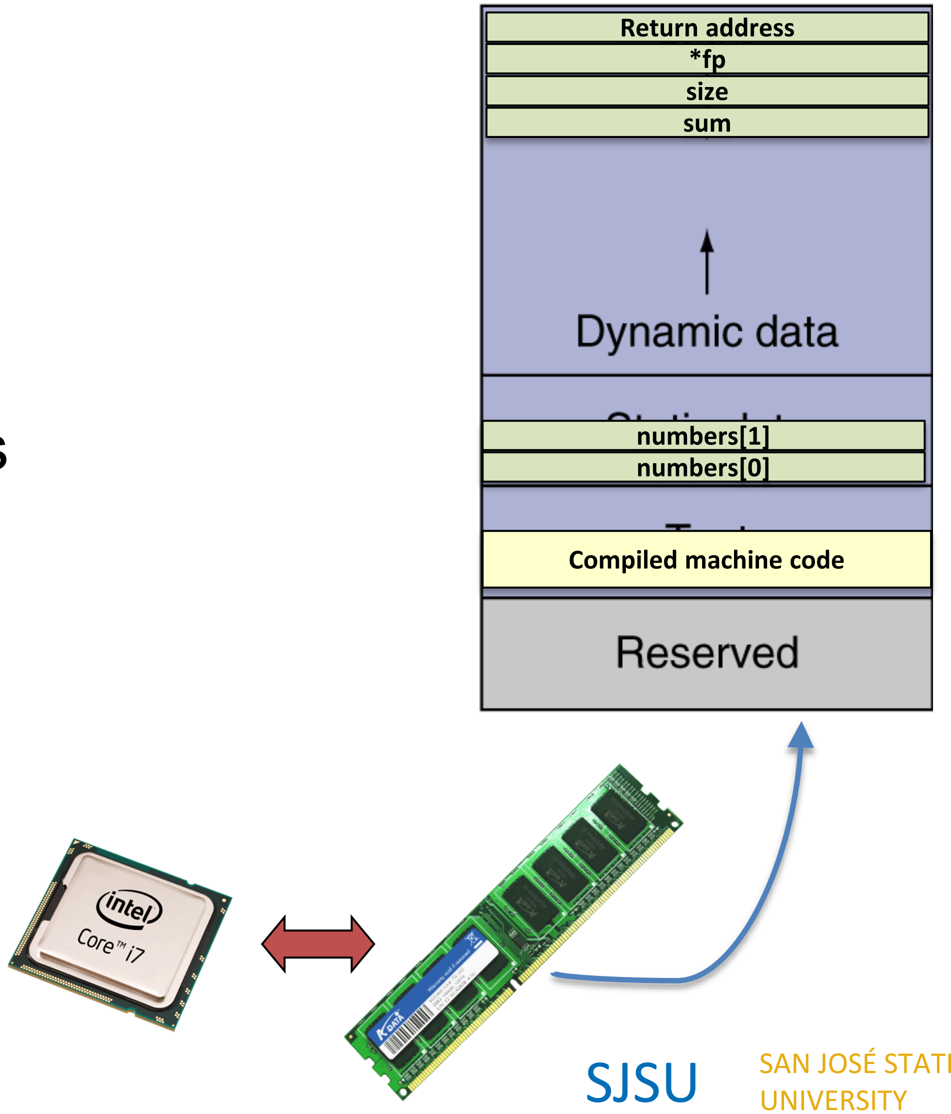
```c
#include <stdio.h>
#include <string.h>

int numbers[2];

void myfunction(void)
{
   FILE *fp;
   int size = 2;
   int sum = 0;

   /* Open file for reading */
   fp = fopen(“mynumbers.txt", “r");

   /* Read and display data */
   fread(numbers, sizeof(int), size, fp);
   fclose(fp);
   sum = numbers[0] + numbers[1];
   printf(“Sum = %d\n”, sum);
}

int main (void) {
   myfunction();
   return(0);
}
```



SJSU SAN JOSÉ STATE UNIVERSITY

Memory Access Is Slow

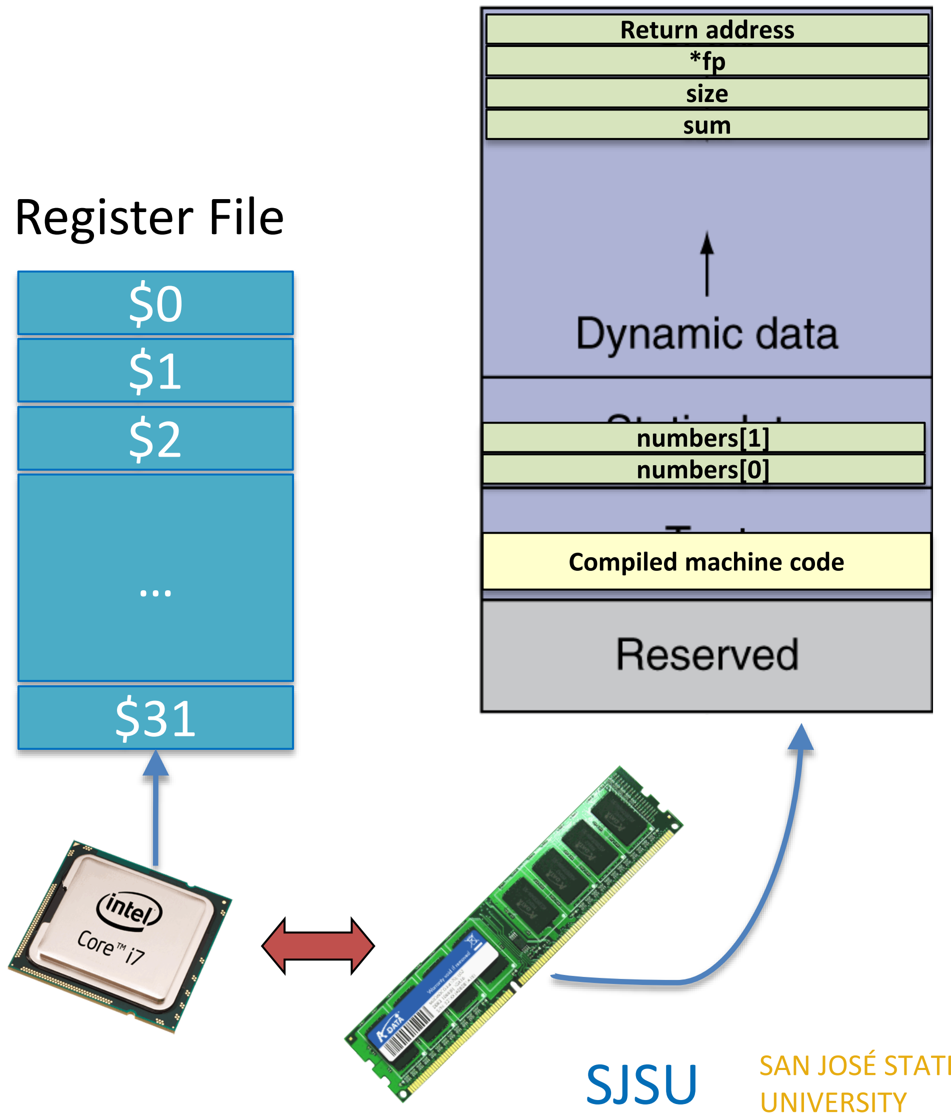
- Variables are all stored in Memory, which is **outside of CPU**
  - Slow..
- How can we execute the operations faster?
  - Use **on-chip memory**



Register File

- **Register File**
  - On-chip memory that stores "Registers"

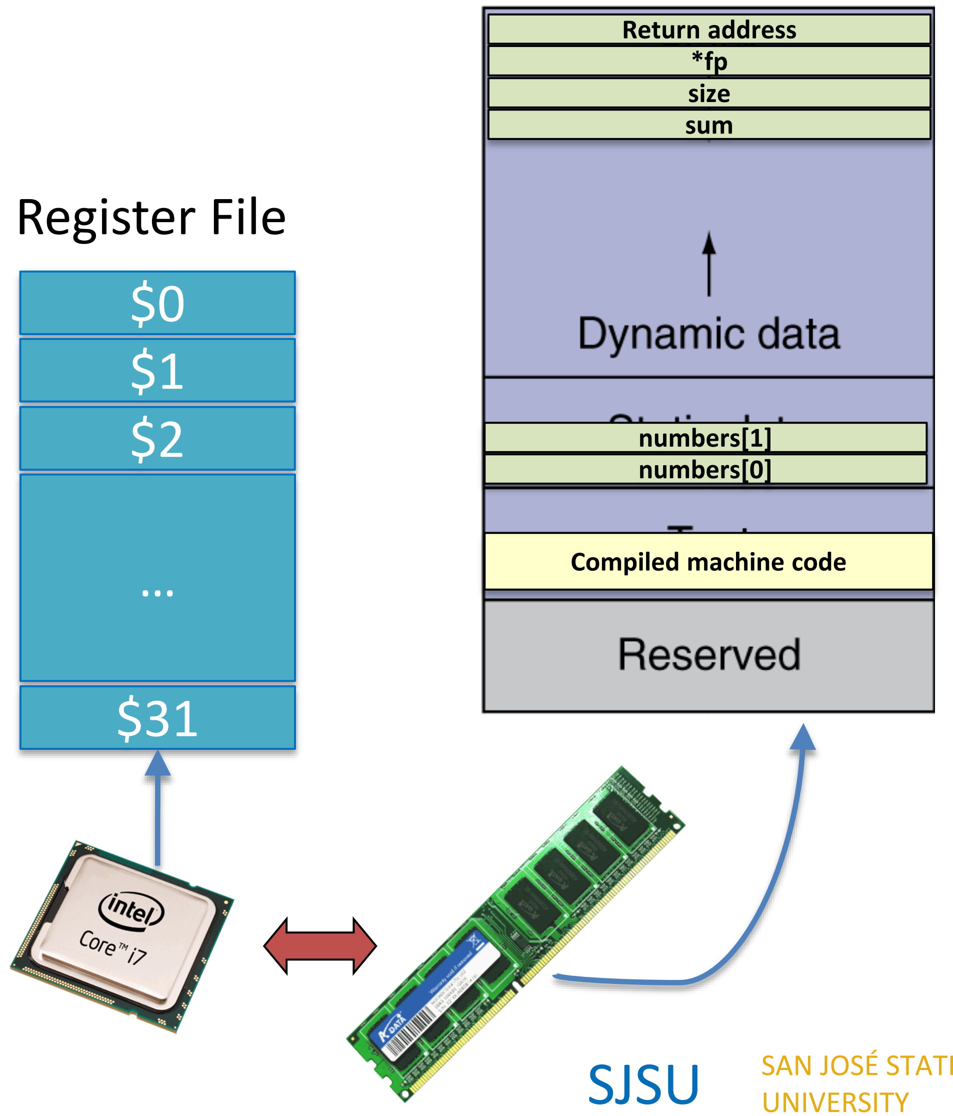
- **Registers**
  - Temporal space to maintain operand values and calculation results before storing back to memory
  - Presented with "$" + "register id" in MIPS processor
    - i.e. $0 : 0th register,
      $1 : 1st register



SJSU SAN JOSÉ STATE UNIVERSITY

Typical Execution Steps

1. Load operand values from memory to registers
2. Do computation on registers
3. Move the results from register to memory



Operations in Hardware

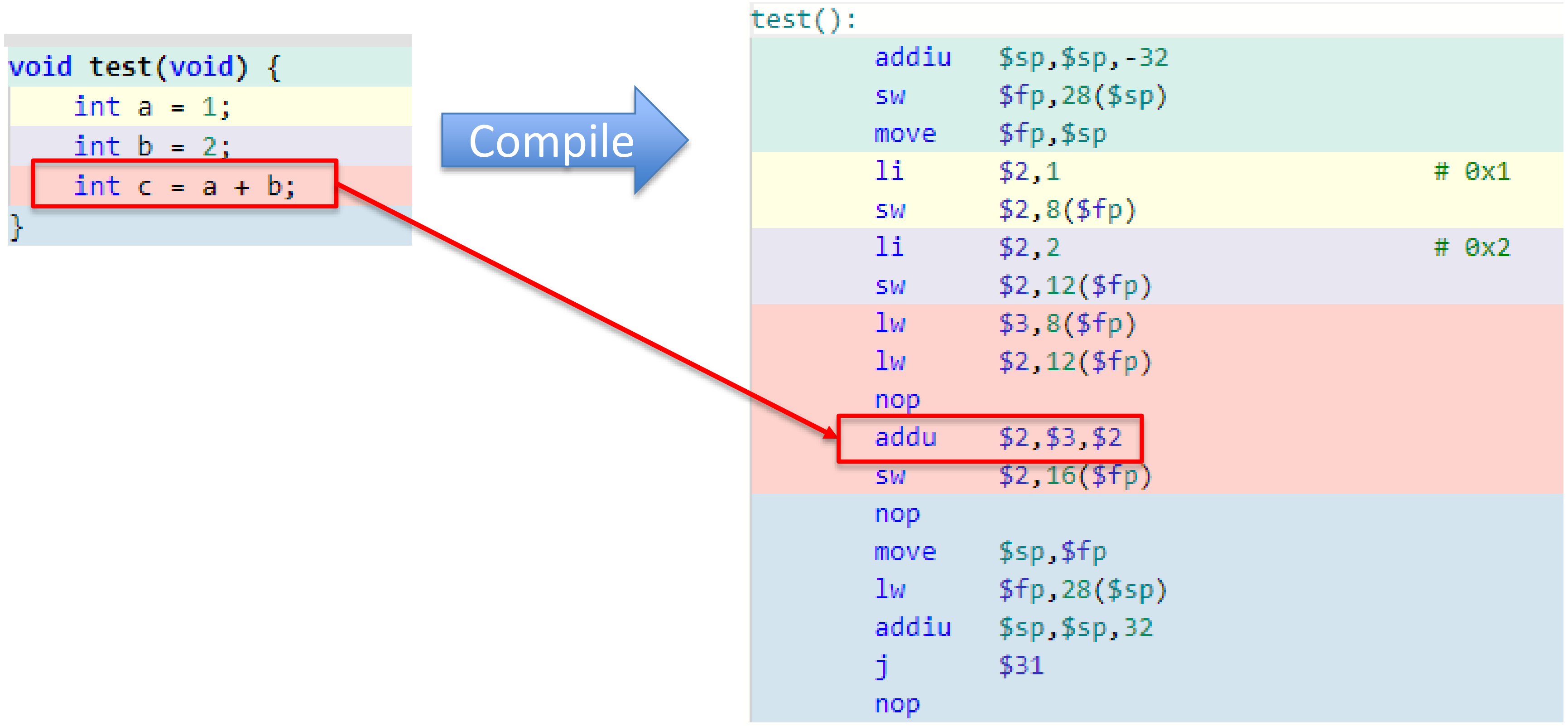
- **Hardware can do one operation at a time**
  - Arithmetic operations
    - add, sub, mult, div, …
  - Data movement
    - move, load data, store data, …
  - Logical operations
    - shift, and, or, xor, …
  - Conditional operations
    - jump, branch on condition, …

- Format of assembly instructions that uses register operands
  - ***Command Result, Operand 1, Operand 2***
  - E.g. **C = A + B → Add C, A, B** (A, B, C should be replaced by register id)

HLL vs. Hardware Operations

- HLL are designed to be understood and programmed easily by programmers

- A HLL line may be a combination of multiple assembly instructions



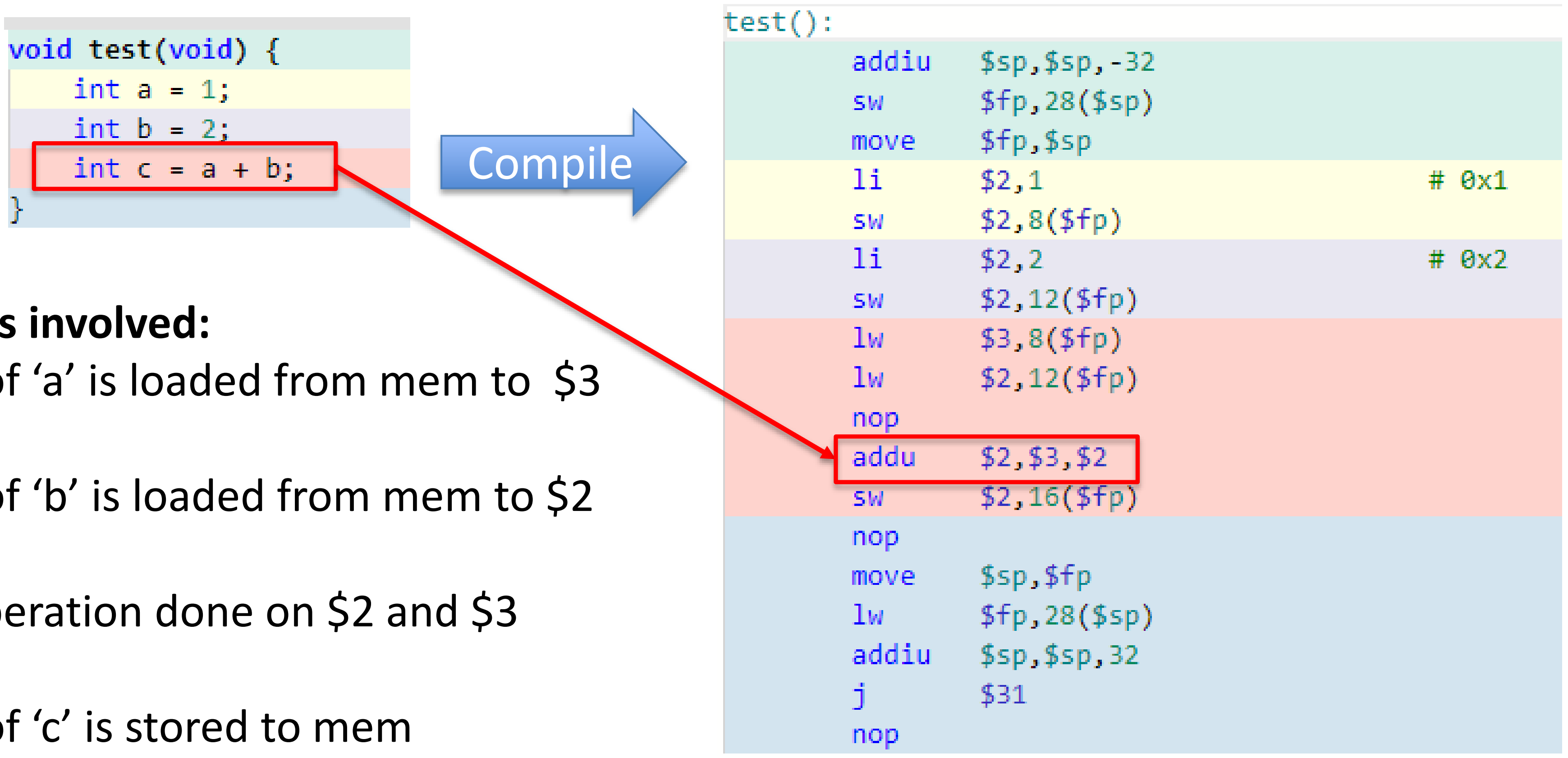
```
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                    # 0x1
        sw      $2,8($fp)
        li      $2,2                    # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

Code Example: Memory & Registers

C code

MIPS Assembly code



```c
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                    # 0x1
        sw      $2,8($fp)
        li      $2,2                    # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

Operations involved:

- Value of ‘a’ is loaded from mem to $3
- Value of ‘b’ is loaded from mem to $2
- **Add** operation done on $2 and $3
- Value of ‘c’ is stored to mem

SJSU SAN JOSÉ STATE UNIVERSITY

Code Example: Memory & Registers

C code

```c
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

Compile

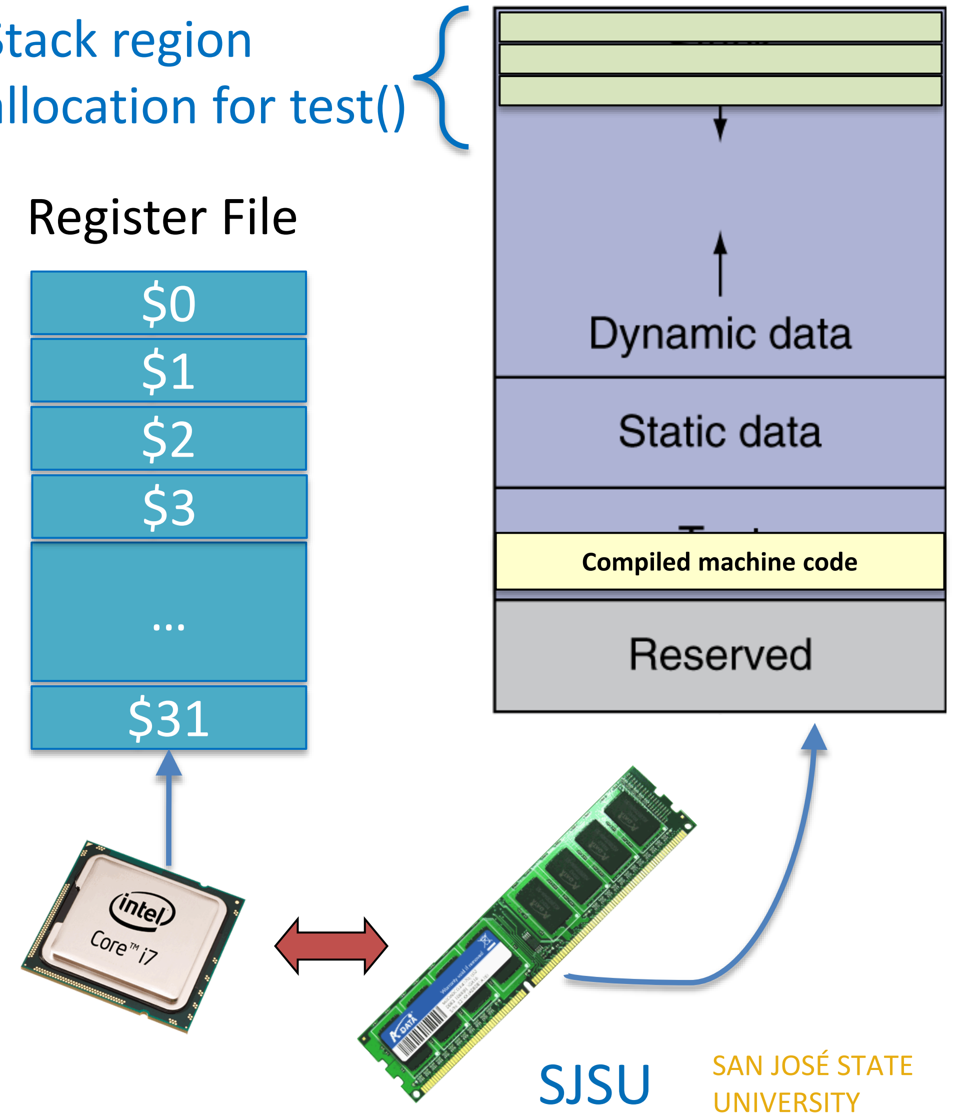
MIPS Assembly code

executing line

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                    # 0x1
        sw      $2,8($fp)
        li      $2,2                    # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

Stack region allocation for test()

Register File



SJSU SAN JOSÉ STATE UNIVERSITY

Code Example: Memory & Registers

C code

```c
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

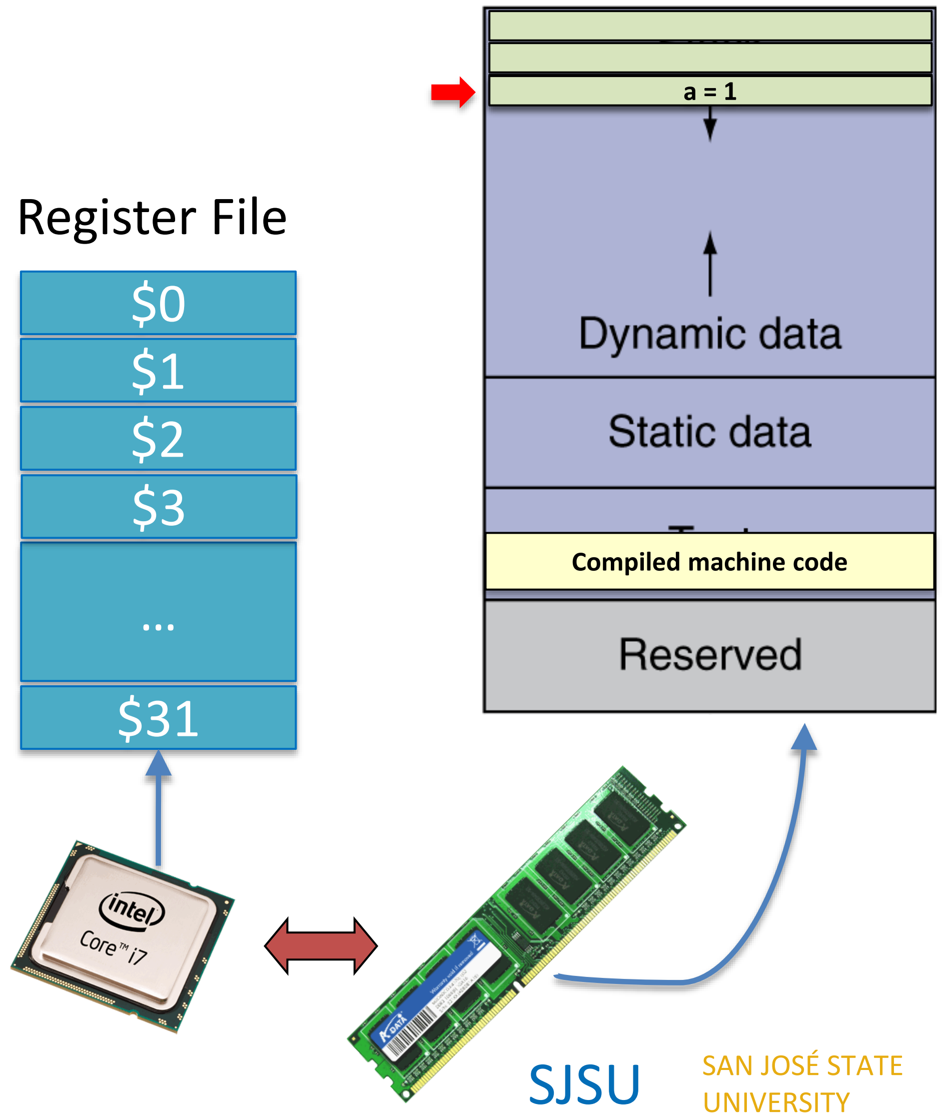
Compile

MIPS Assembly code

executing line

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                    # 0x1
        sw      $2,8($fp)
        li      $2,2                    # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

Register File



Code Example: Memory & Registers

C code

```c
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

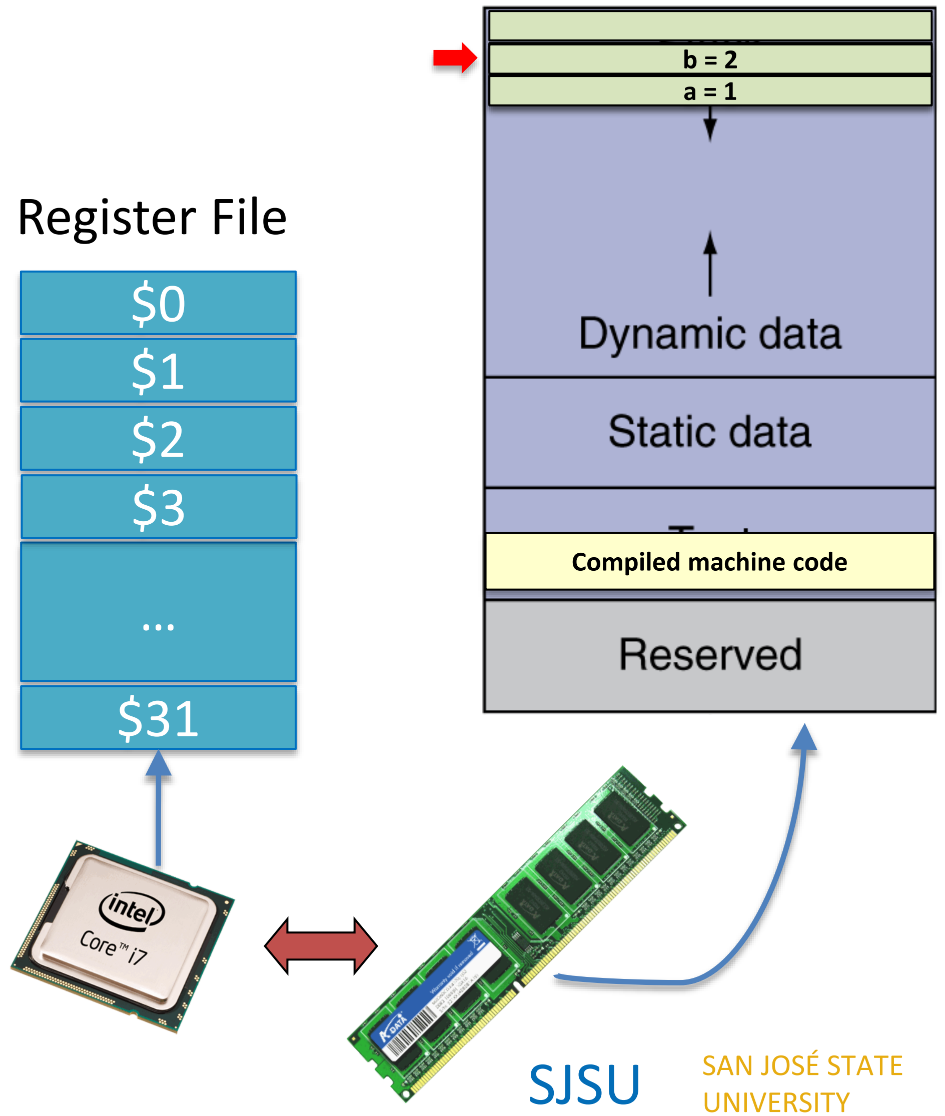
Compile

MIPS Assembly code

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                        # 0x1
        sw      $2,8($fp)
        li      $2,2                        # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

executing line

Register File



Code Example: Memory & Registers

C code

```
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

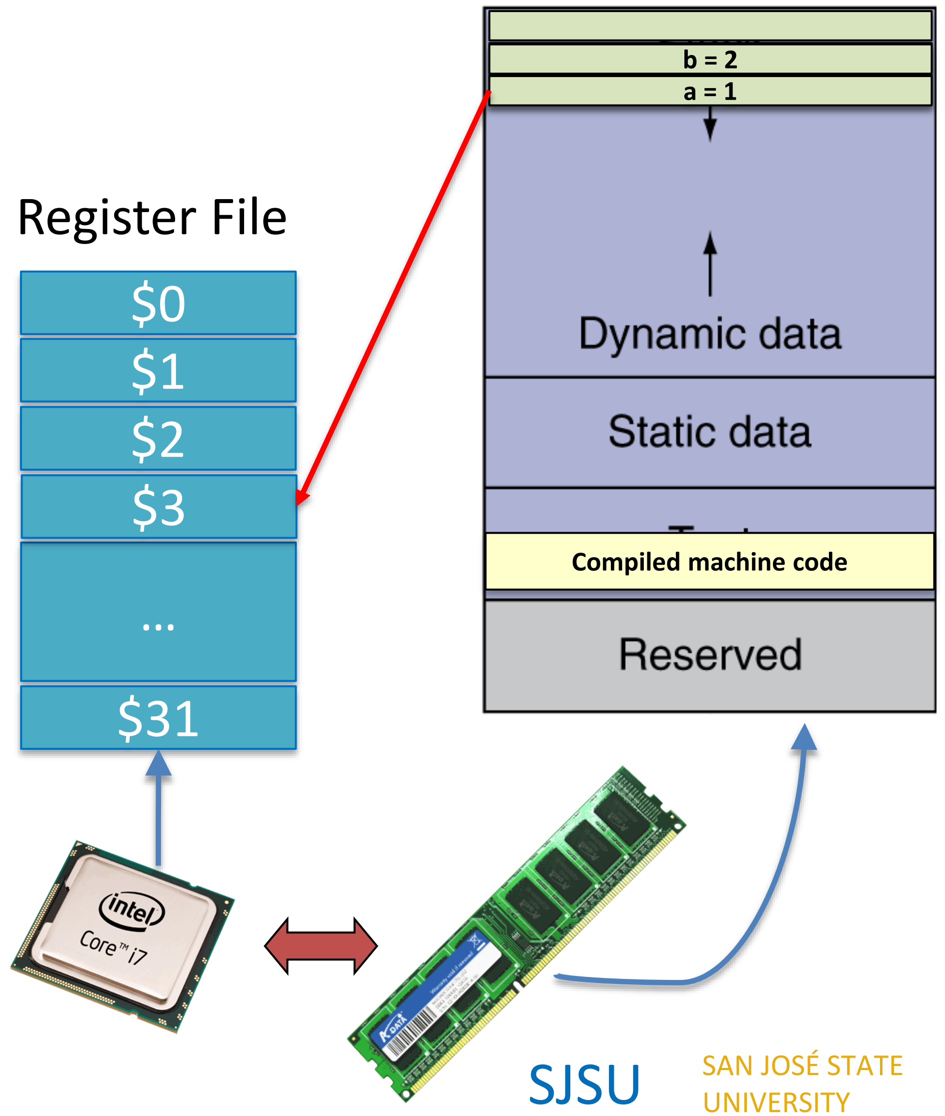
Compile

MIPS Assembly code

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                        # 0x1
        sw      $2,8($fp)
        li      $2,2                        # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

executing line

Register File



SJSU SAN JOSÉ STATE UNIVERSITY

Code Example: Memory & Registers

C code

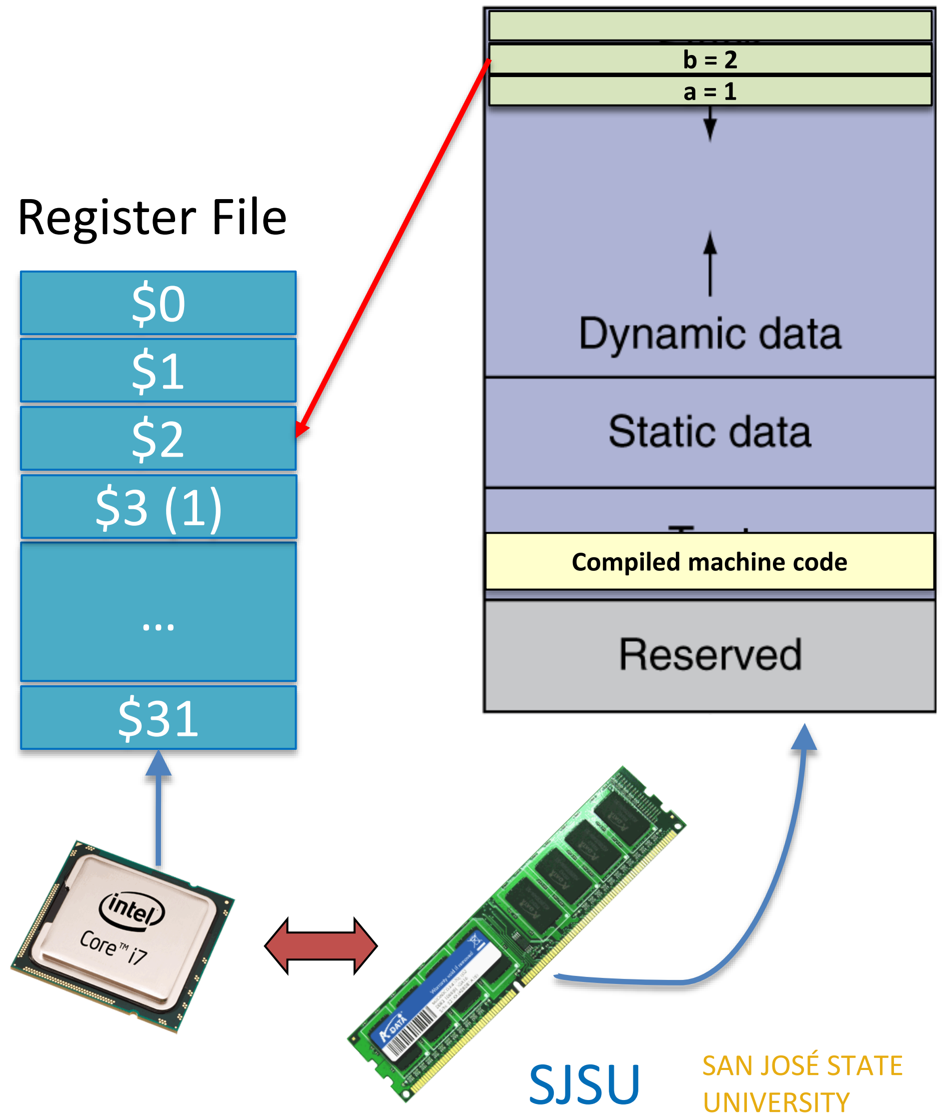
```c
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

Compile

MIPS Assembly code

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                        # 0x1
        sw      $2,8($fp)
        li      $2,2                        # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

executing line → `lw $2,12($fp)`



Code Example: Memory & Registers

C code

```c
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

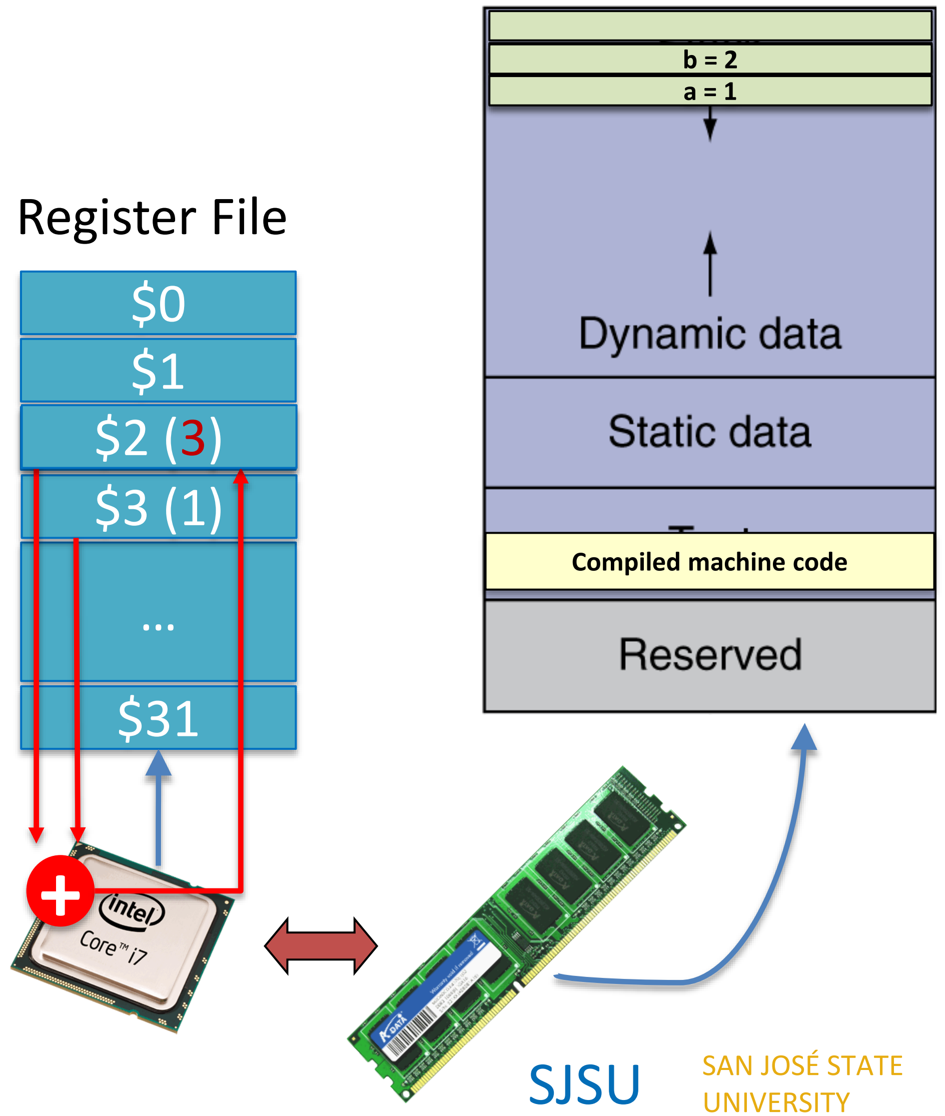
Compile

MIPS Assembly code

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                        # 0x1
        sw      $2,8($fp)
        li      $2,2                        # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

executing line ➡

Register File



Code Example: Memory & Registers

C code

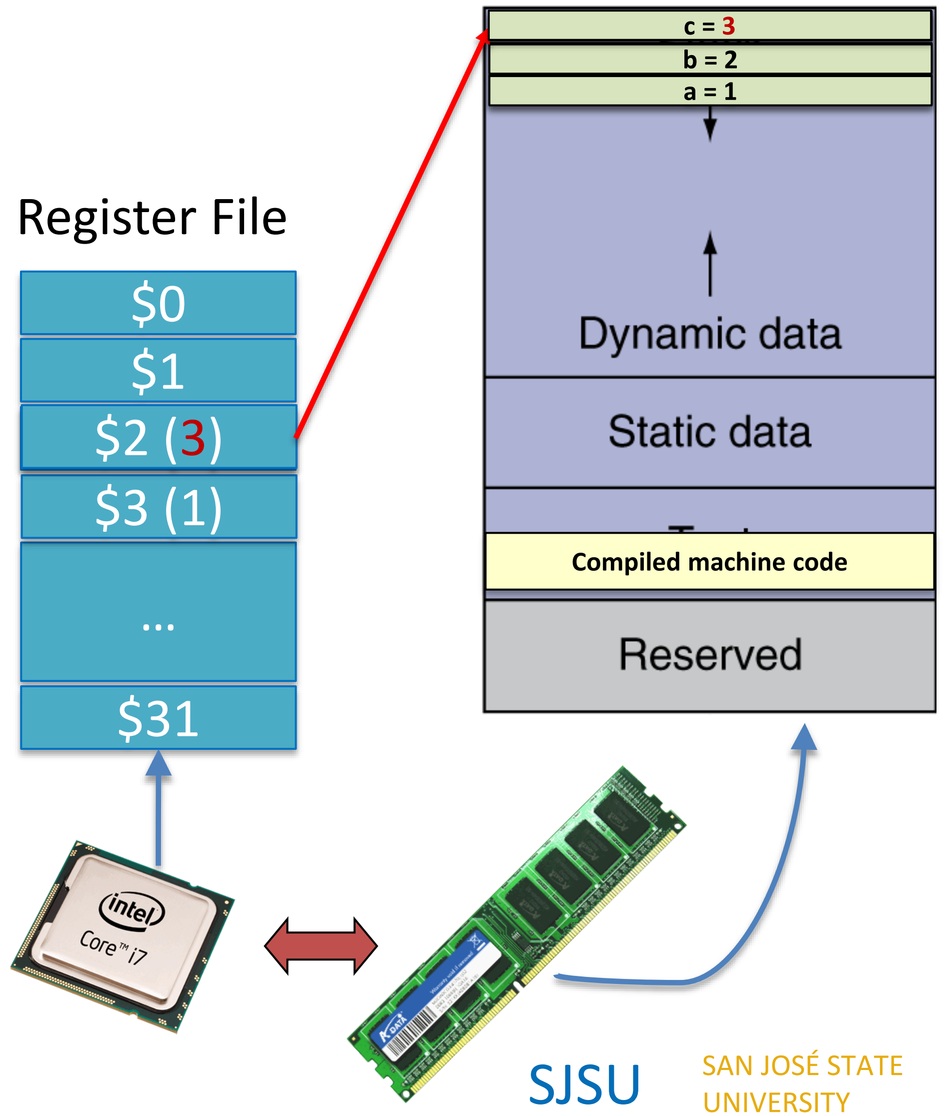
```c
void test(void) {
    int a = 1;
    int b = 2;
    int c = a + b;
}
```

Compile

MIPS Assembly code

```
test():
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,1                    # 0x1
        sw      $2,8($fp)
        li      $2,2                    # 0x2
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        nop
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

executing line



SJSU SAN JOSÉ STATE UNIVERSITY

Registers of MIPS CPUs

Assembler Name	Register Number	Description
$zero	$0	Constant 0 value
$at	$1	Assembler temporary
$v0-$v1	$2-$3	Function return values
$a0-$a3	$4-$7	Function Arguments
$t0-$t7	$8-$15	Temporaries
$s0-$s7	$16-$23	Saved Temporaries
$t8-$t9	$24-$25	Temporaries
$k0-$k1	$26-$27	Reserved for OS kernel
$gp	$28	Global Pointer (Global and static variables/data)
$sp	$29	Stack Pointer
$fp	$30	Frame Pointer
$ra	$31	Return Address

Exercise: Register Reuse

- **Example:**
  - **Assume g, h, i and j are loaded to registers, \$t0, \$t1, \$t2, \$t3**

  - **HLL:**

    f = (g + h) – (i + j);

  - **Assembly (with add & sub operations)?**

    ```
    add $t0, $t0, $t1
    add $t2, $t2, $t3
    sub $t2, $t0, $t2
    ```

- **Register values are maintained unless overwritten**

SJSU SAN JOSÉ STATE UNIVERSITY

A Brief Review

- **One line of HLL can involve:**
  1. **Load operand values from memory to registers**
  2. **Do computation on registers**
  3. **Move the results from register to memory**

Data load operations
    move, load data, …

Arithmetic operations
    add, sub, mult, div, …
Logical operations
    shift, and, or, xor, …
Conditional operations
    jump, branch on condition, …

Data store operations
    move, store data, …

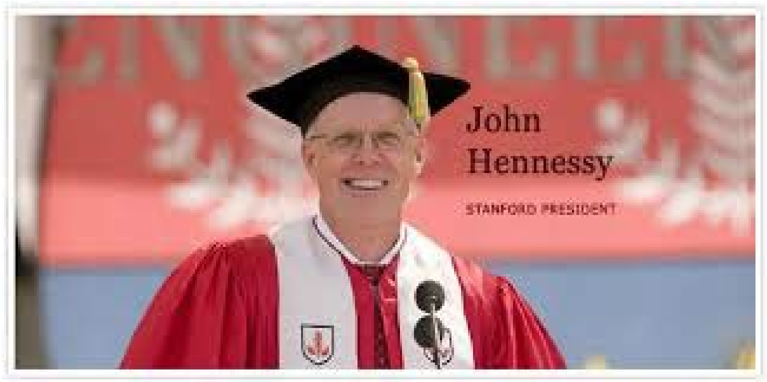
Operations of MIPS CPUs

C operator	Assembly Operations	Comments
C = A + B	**add** C, A, B	Add two values
C = A - B	**sub** C, A, B	Subtract one from another
C = A * B	**mul** C, A, B	Multiply two values
C = A & B	**and** C, A, B	Logical AND operation
C = A | B	**or** C, A, B	Logical OR operation
C = A ^ B	**xor** C, A, B	Logical XOR operation
C = A << shamt	**sll** C, A, shamt	Shift left by shamt
C = A >> shamt	**srl** C, A, shamt	Shift right by shamt
If (A < B) C = 1	**slt** C, A, B	Set if less than
C = Memory	**lw** C, Memory address	Load value from memory
Memory = C	**sw** C, Memory address	Store value to memory
If (A == B) go to Addr	**beq** A, B, address	Jump if A == B
	Many more …	

- Note: A, B, C in the table should be replaced by proper register ids

About MIPS Assembly

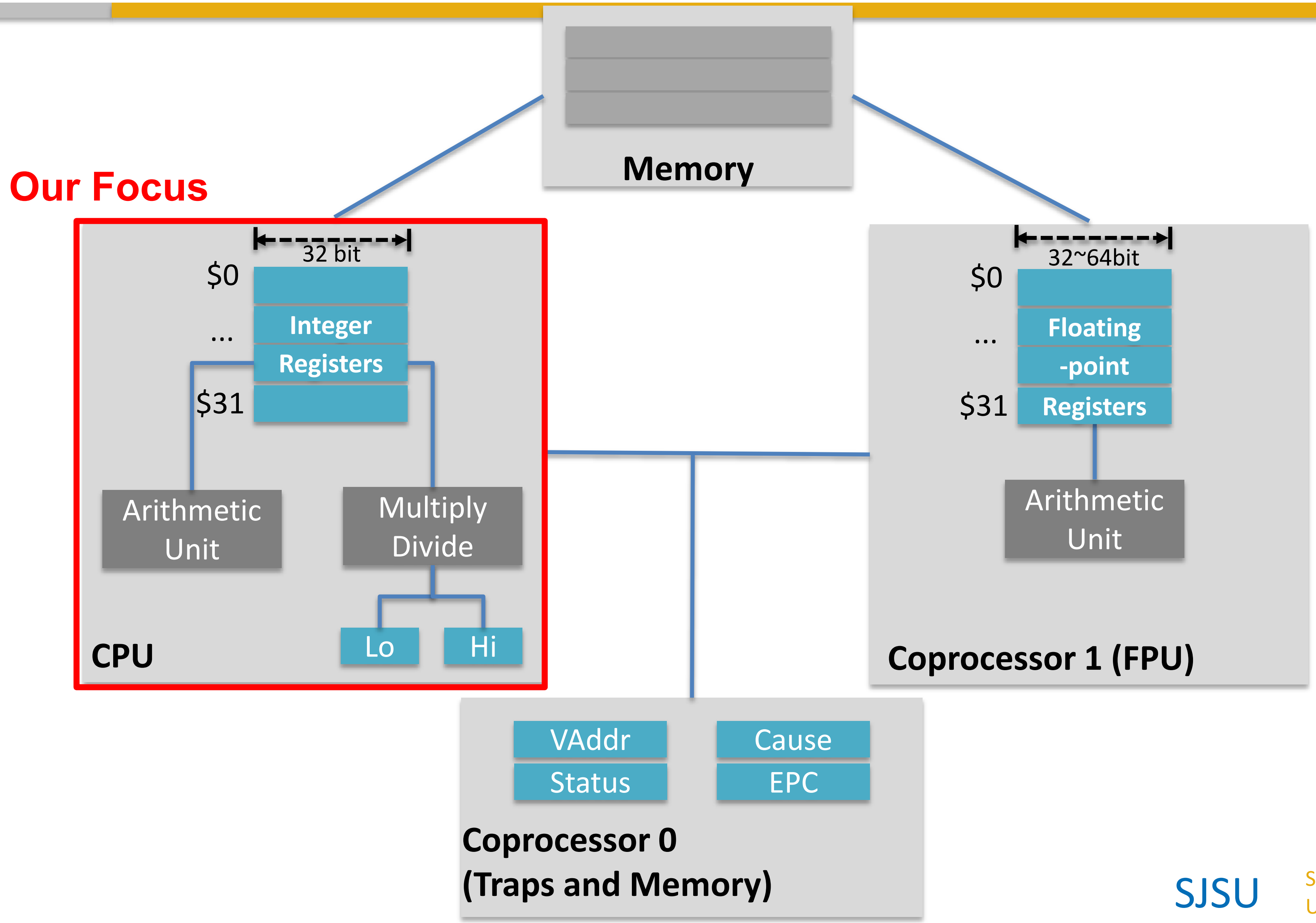
- **CPUs use their own assembly languages**
  - Assembly language of ARM, Pentium, Opteron… are all different
- **MIPS Assembly Features**
  - Very similar to ARM Assembly
  - One of the earliest RISC architectures that used pipelined instruction processing
- **Developed by MIPS Technologies**
  - Now maintained by Wave Computing
  - Various generations
    - MIPS I~V
    - MIPS32 (32-bit processor)
    - MIPS64 (64-bit processor)
    - microMIPS..



Two types of Computers

- **Reduced Instruction Set Computer (RISC):** MIPS, ARM, …
  - Operands are in registers
  - Each instruction can do only one operation: Simple but longer codes
- **Complex Instruction Set Computer (CISC):** Intel, AMD, …
  - Operands can be in registers, stacks, accumulators, in memories
  - Each instruction can do multiple operations: Complex but shorter codes
  - Preferred when memory was very expensive

MIPS Processor Organization



SJSU SAN JOSÉ STATE UNIVERSITY

About MIPS32 Processor

- Instructions are 32-bit wide
- Registers and Computing Logic use 32-bit data
- Memory bus is logically 32-bit wide
- 32 general purpose registers (GPRs) for integer and address values
    - A few special ones (i.e. $zero: constant 0, $fp: frame pointer, $sp: stack pointer..)
- 32 floating point registers for floating point operations (not our focus)

Conclusion Time

What is ISA?

Hardware operation definition

What is Microarchitecture?

ISA implementation

What is Register File?

On-chip memory

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY