Computer Architecture
CSCI 4350

Instruction Set Architecture

Kwangsung Oh

kwangsungoh@unomaha.edu

http://faculty.ist.unomaha.edu/kwangsungoh

University of Nebraska Omaha

Today

- High-level language to Instructions
- ISA - MIPS Machine Instructions
- Function calls
- Examples

Programming Languages

- High-level language (Human readable)
  - Procedural languages: C, Pascal …
  - Object-oriented languages: C++, Objective-C, Java
- Low-level language (Still human readable)
  - Symbolic machine languages
- Machine language
  - Binary codes

Compiler & Interpreter

- Compiler
  - Translating a source program into a target program
  - Source program – High-level language
  - Target program – Machine language
- Interpreter
  - Translating and executing programs directly (e.g., JVM)
  - Bytecode (e.g., JVM virtual machine language)
  - Translating and executing bytecode

From High-Level to Instruction

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly language program (for MIPS)

```
swap:
      multi $2, $5,4
      add   $2, $4,$2
      lw    $15, 0($2)
      lw    $16, 4($2)
      sw    $16, 0($2)
      sw    $15, 4($2)
      jr    $31
```

Assembler

Binary machine language program (for MIPS)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

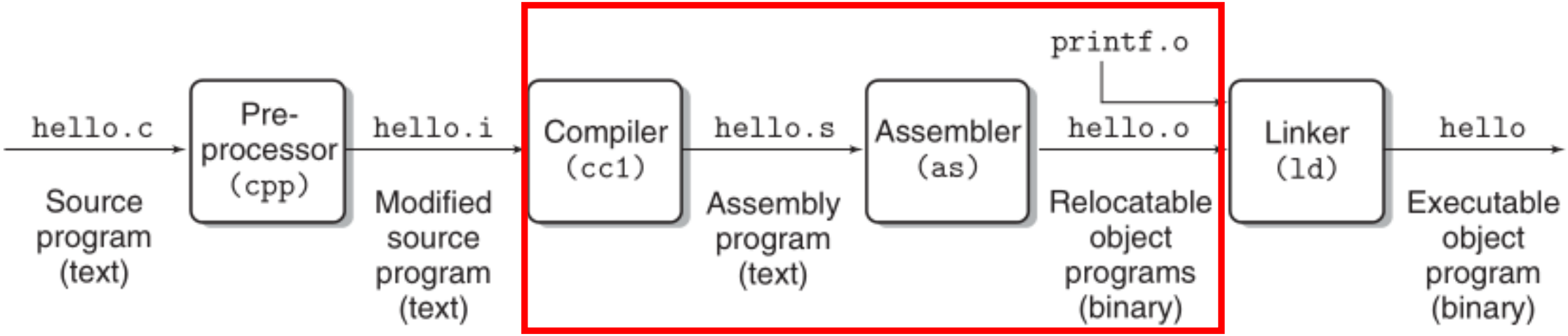
Compilation Process

code/intro/hello.c

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```

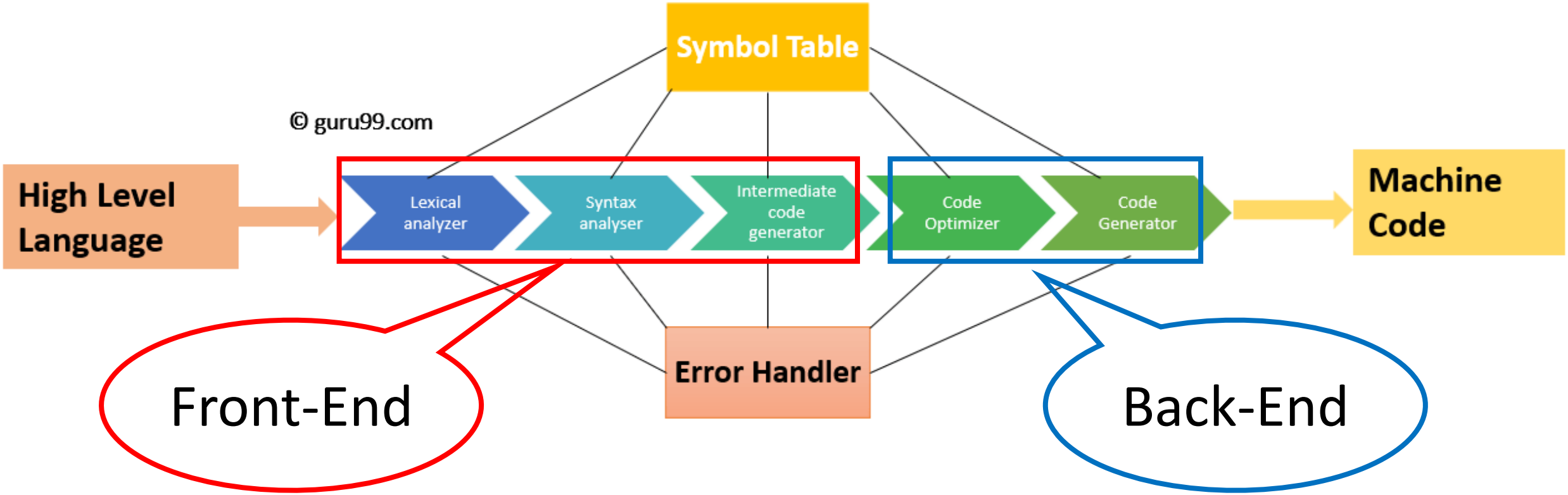
code/intro/hello.c



Good Compiler

- Correctness
- Optimal target language
- Performance
- Separate compilation
- Good diagnostic information

Compiler Phases



Today

- High-level language to Instructions
- ISA - MIPS Machine Instructions
- Function calls
- Examples

Machine State

- ISA (machine states and instructions)
- Registers
  - CPU Internal storage
  - 32, 32-bit (word size) registers in MIPS
  - **Register** -> Cache -> **Memory** ---- (OS) ----> **Hard (SSD) disk**
- Memory
  - A large single array (string at address 0)
  - Storing programs (instructions and data)
  - Load (memory to register) and Store (register to memory)

Data in Memory

- Word
  - Basic size (unit) between memory and registers (32-bit MIPS)
  - Double word (64-bit), half word (16-bit), byte (8-bit)
  - Load instructions: ldw, lddw, ldhw, ldb
- Byte addressability: Each byte has an address
- Address alignment
  - Addresses of objects start at multiple of their size
  - Byte – 0, 1, 2, 3, 4, 5, 6, 7 Half word – 0, 2, 4, 6
  - Word – 0, 4 Double word – 0

Machine Instruction

- Opcode
  - Operation to be performed
  - Ex) ADD, MULT, LOAD, STORE, JUMP …
- Operand
  - Data location
  - Source operands (input) and destination operands (output)
  - Register number – R0 ~ R31
  - Memory address – 100 (R2), x1001F

Instruction types

- Arithmetic and logic instructions
  - Actual computation
  - Ex) ADD, SUB, MULT, AND, DIV, SHIFT, FDIVID, FADD …
- Data transfer (memory) instructions
  - Data transfer between registers and memory
  - Ex) LOAD and STORE – mapped IO (from/to IO ports)
- Control transfer (branch) instructions
  - Change the control flow (unconditional or conditional)
  - Ex) JUMP, CALL, RETURN, BEQ …

Instruction Format

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

R-type – Arithmetic and logic

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

I-type – Data transfer, conditional branch, immediate format

op	address
6 bits	26 bits

J-type – Jump (and Jump and Link)

R-type (Register)

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- R-type (Arithmetic and logic Instructions)
  - op – Opcode, operation of the instruction
  - rs – 1st source register
  - rt – 2nd source register
  - rd – destination register
  - shamt – shift amount
  - funct – Function code, variant of the opcode

R-type Encoding Example

- ADD R8, R9, R10

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
0 (R-type)	**R9**	**R10**	**R8**	**0**	**100000 (add)**
000000	**01001**	**01010**	**01000**	**00000**	**100000**

0000	**0001**	**0010**	**1010**	**0100**	**0000**	**0010**	**0000**

= $012A4020_{hex}$

= $19{,}546{,}114_{ten}$

Funct Table (R-type)

op(31:26)=000000 (R-format), funct(5:0)								
2–0 5–3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump register	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)			set l.t.	set l.t. unsigned				
6(110)								
7(111)								

MIPS Register Convention

Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

Decoding Example

- $00af8020_{hex}$

0000	0000	1010	1111	1000	0000	0010	0000

R-type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
000000	00101	01111	10000	00000	100000

= ADD R16, R5, R15

= ADD $s0, $a1, $t7

I-type (immediate)

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- I-type (Loads/stores and conditional branches)
  - op – Opcode, operation of the instruction
  - rs – Base register
  - rt – 2nd source register
  - Address: +/- 2 15 bytes offset

Opcode Table 1

op(31:26)								
28–26 31–29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	set less than imm. unsigned	andi	ori	xori	load upper immediate
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	load byte unsigned	load half unsigned	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	load linked word	lwc1						
7(111)	store cond. word	swc1						

I-type Encoding Example

- LOAD $s0, 100 ($s1)

op	rs	rt	Constant or address
6 bits	5 bits	5 bits	16 bits

LOAD	**$s1**	**$s0**	**100**

100011	**10001**	**10000**	**0000000001100100**

1000	**1110**	**0011**	**0000**	**0000**	**0000**	**0110**	**0100**

= $8E300064_{hex}$

I-type Encoding Example

- STORE $s1, 32 ($s0)

op	rs	rt	Constant or address
6 bits	5 bits	5 bits	16 bits

STORE	$s0	$s1	32

101011	10000	10001	0000000000100000

1010	1110	0001	0001	0000	0000	0010	0000

= $AE110020_{hex}$

J-type Encoding Example

- JAL LOOP (Assume LOOP: 8000)

op	Address
6 bits	26 bits

JAL	8000

000011	00000000000001111101000000

0000	1100	0000	0000	0001	1111	0100	0000

= $0C001F40_{hex}$

MIPS Addressing Mode

- Base (or displacement) addressing
  - Sum of register and a constant – LOAD R1, 100 (R2)
- Register addressing
  - Address in a register – JR $ra (return address register)
- PC-relative addressing
  - Sum of PC and a constant – BEQ R1, R2, Loop
- Immediate addressing
  - For small constant operand – ADDI R1, R2, 3
- Pseudodirect addressing
  - 26bit offset with the upper bits of PC – J L1

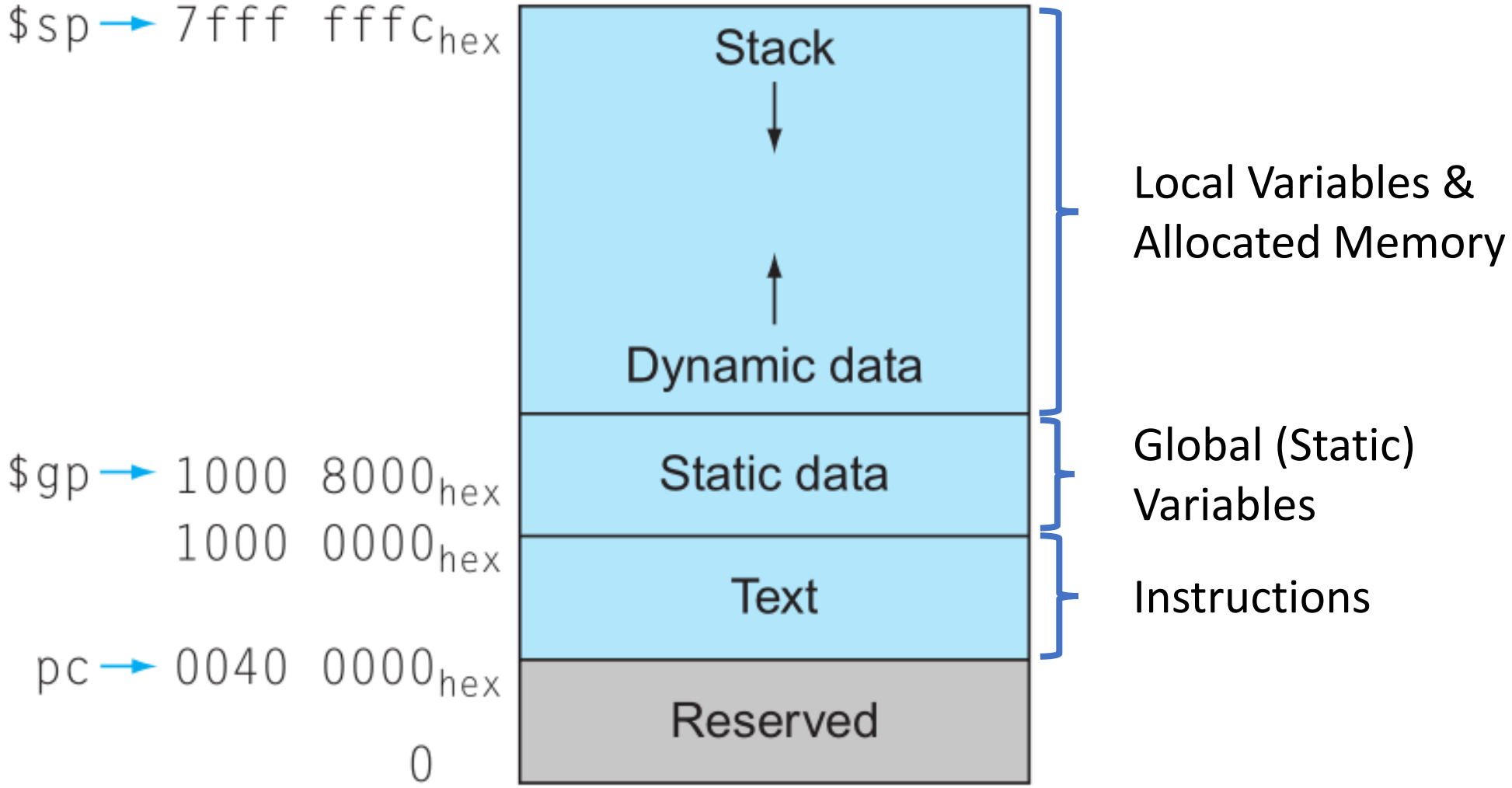
Today

- High-level language to Instructions
- ISA - MIPS Machine Instructions
- Function calls
- Examples

Function (Procedure) Call

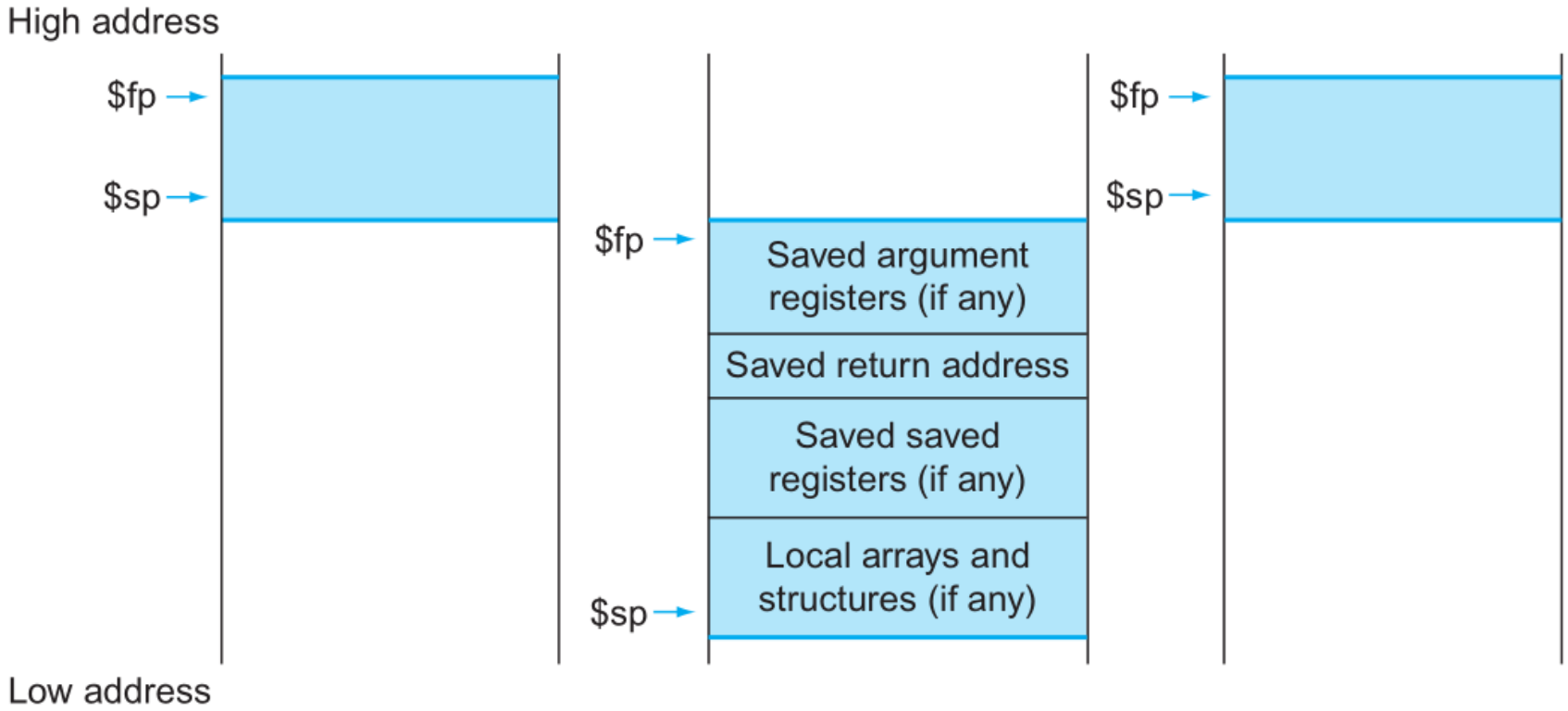
1. Setting parameters in registers by caller
   - **$a0** ~ **$a3** – four agreements registers
2. Transferring control to callee (Function code)
   - JAL (Jump and link) – Update **$ra** (PC+4) and Jump to callee
3. Creating a stack for callee and performing
4. Placing the result value by callee
   - **$v0** ~ **$v1** – two registers to return a value
5. Transferring control to caller ($ra)

MIPS Address Space (User Space)



Stack Pointer

- Stack frame (activation record)
  - Setting arguments, return address, saved registers, local variable
  - $fp – Frame Pointer, $sp – Stack Pointer



Today

- High-level language to Instructions
- ISA - MIPS Machine Instructions
- Function calls
- Examples

MIPS Procedure Example

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

```
leaf_example:  addi   $sp, $sp, -12
               sw     $t1, 8($sp)
               sw     $t0, 4($sp)
               sw     $s0, 0($sp)
               add    $t0,$a0,$a1
               add    $t1,$a2,$a3
               sub    $s0,$t0,$t1
               add    $v0,$s0,$zero
               lw     $s0, 0($sp)
               lw     $t0, 4($sp)
               lw     $t1, 8($sp)
               addi   $sp,$sp,12
               jr     $ra
```

Assembly code?

MIPS Recursion Example

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

Assembly code?

```
fact: addi  $sp, $sp, -8
      sw    $ra, 4($sp)
      sw    $a0, 0($sp)
      slti  $t0, $a0, 1
      beq   $t0, $zero, L1
      addi  $v0, $zero, 1
      addi  $sp, $sp, 8
      jr    $ra
L1:   addi  $a0, $a0, -1
      jal   fact
      lw    $a0, 0($sp)
      lw    $ra, 4($sp)
      addi  $sp,$sp, 8
      mul   $v0, $a0, $v0
      jr    $ra
```