CPE 431/531

Chapter 2 – Instructions: Language of the Computer

Dr. Rhonda Kay Gaede

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

2.1 Introduction

- The words of a computer's language are called __________ and its vocabulary is called an _________ ___ __________________.
- Instruction sets are more similar than they are different, however there are two camps:
  - RISC – _____________
  - CISC – ________

2.2 Basics of MIPS Arithmetic

- We need arithmetic

```
add a, b, c
```

- From high level

```
a = b + c + d + e;
```

- _________ the number of operands keeps the hardware ________.
- Design Principle 1: Simplicity favors regularity

2.2 Compiling C into MIPS

Compilation is the process of creating MIPS assembly language from a high level language.

Examples:

```
a = b + c;
d = a - e;
f = (g + h) - (i + j);
```

2.3 MIPS Basics

- In high level languages, variables live in __________
- In MIPS assembly, operands live only in ____________
- ______________ instructions move variables from memory/registers to registers/memory.
- MIPS has ___ registers and an address space of ____ memory bytes.
- Design Principle 2: Smaller is faster.

2.2 Compiling C into MIPS (Registers)

Reconsider

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

The variables **`f`**, **`g`**, **`h`**, **`i`**, and **`j`** are assigned to registers **`$s0`**, **`$s1`**, **`$s2`**, **`$s3`**, and **`$s4`**, respectively.

2.3 Memory Operands: First Pass

- Data transfer instructions

  Load

  Store

- Compiling an Assignment When an Operand is in Memory

  The compiler has associated g with **`$s1`** and h with **`$s2`** and A is an array of 100 words, base pointer **`$s3`**

  g = h + A[8];

- Hardware/Software Interface
  - A compiler translates, associates variables with registers, allocates memory to data structures.

2.3 Memory Operands: Second Pass

- Bytes/Words
  - 32-bit words consist of 4 8-bit bytes
  - Computers are bigendian or little endian depending on whether the _____ _____ is _____ or _____ significant
  - MIPS is _____ addressable
- Compiling Using Load and Store

  h is associated with **`$s2`** and the base address of **`A`** is in **`$s3`**

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

- . Hardware/Software Interface
  - The compiler keeps frequently used items in registers, spills other variables to memory.

2.3 Constant or Immediate Operands

- More than half of the MIPS arithmetic instructions have a _________ as an operand when running the SPEC CPU 2006 benchmarks.
- With the instructions we've seen so far, constants must be put in _________ when the program was loaded and then we would have to load them into a ________ to use.
- The alternative is to add a different kind of instruction.

```
addi $s3, $s3,4
```

- Design Principle 3: Make the common case fast.

2.4 Signed and Unsigned Numbers

- Unsigned

  1011_2 =

  Range for n bits:

- Signed

  1011_2 =

  Range for n bits:

- Finding the 2's complement

  00001010

  00101000

- Sign Extension

  00001010

  10110111

2.5 R-type Instruction Format

- Translating a MIPS assembly instruction into a machine instruction:
  `add $t0, $s1, $s2` (R-type)

Op(6)	rs(5)	rt(5)	rd(5)	shamt(5)	funct(6)

- MIPS Fields
  - op: opcode
  - rd: register destination
  - rs: first register source
  - rt: second register source/destination register for lw
  - shamt: shift amount
  - funct: function code

2.5 I-Type Instruction Format

- One size doesn't fit all. `lw` and `sw` have different requirements than `add`.
- Design Principle 4: Good design demands good compromises.

```
lw $t0, 32($s3)
```

op(6) rs(5) rt(5) constant or address(16)

- For data transfer, address offset is limited to ________, __________.
- Another Translation Example: A[300] = h + A[300];

2.5 Instructions for Making Decisions

- Two conditional ones for now:

```
beq register1, register2, L1
```

```
bne register1, register2, L1
```

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

2.7 Adding less than or greater than

- Less than is useful, i.e., for (i = 0; i < 10; i++)

```
slti $t0, $s1, 10
```

```
bne $t0, $zero, offset
```

```
slt $t0, $s0, $s1
bne $t0, $zero, offset
```

2.7 Compiling a while loop

Consider

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

where is associated with `$s3` and `k` with `$s5` and the base of array `save` is `$s6`.

2.8 Supporting Procedures in Computer Hardware

- Steps involved in calling a procedure (function)
  1) Make ____________ available to the ______ procedure
  2) Transfer _________ to the procedure
  3) __________ the needed ________ for the procedure.
  4) Perform the _________ ____
  5) Make _______ _________ to the calling procedure
  6) Transfer _________ back to ________ procedure
- Support comes in registers and instructions
  - Registers
    - **`$a0-$a3`** –
    - **`$v0-$v1`** –
    - **`$ra`**
  - Instructions
    - **`jal`**,
    - **`jr`**

2.8 Compiling a Leaf Procedure

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

leaf_example:  sub    $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)
               add    $sp, $sp, 12
               jr     $ra
```

2.8 Leaf Example Stack

- In the previous example, what happens if we change the procedure to have one more argument? _Spill them to the stack_

2.8 Nested Procedures

- Calling Procedure
  - Pushes its argument registers onto the stack so it can put arguments there for the callee
  - Pushes any temporary registers it needs after the call onto the stack
  - Pushes **`$ra`** onto the stack
- Called Procedure
  - Pushes saved registers it plans to use onto the stack

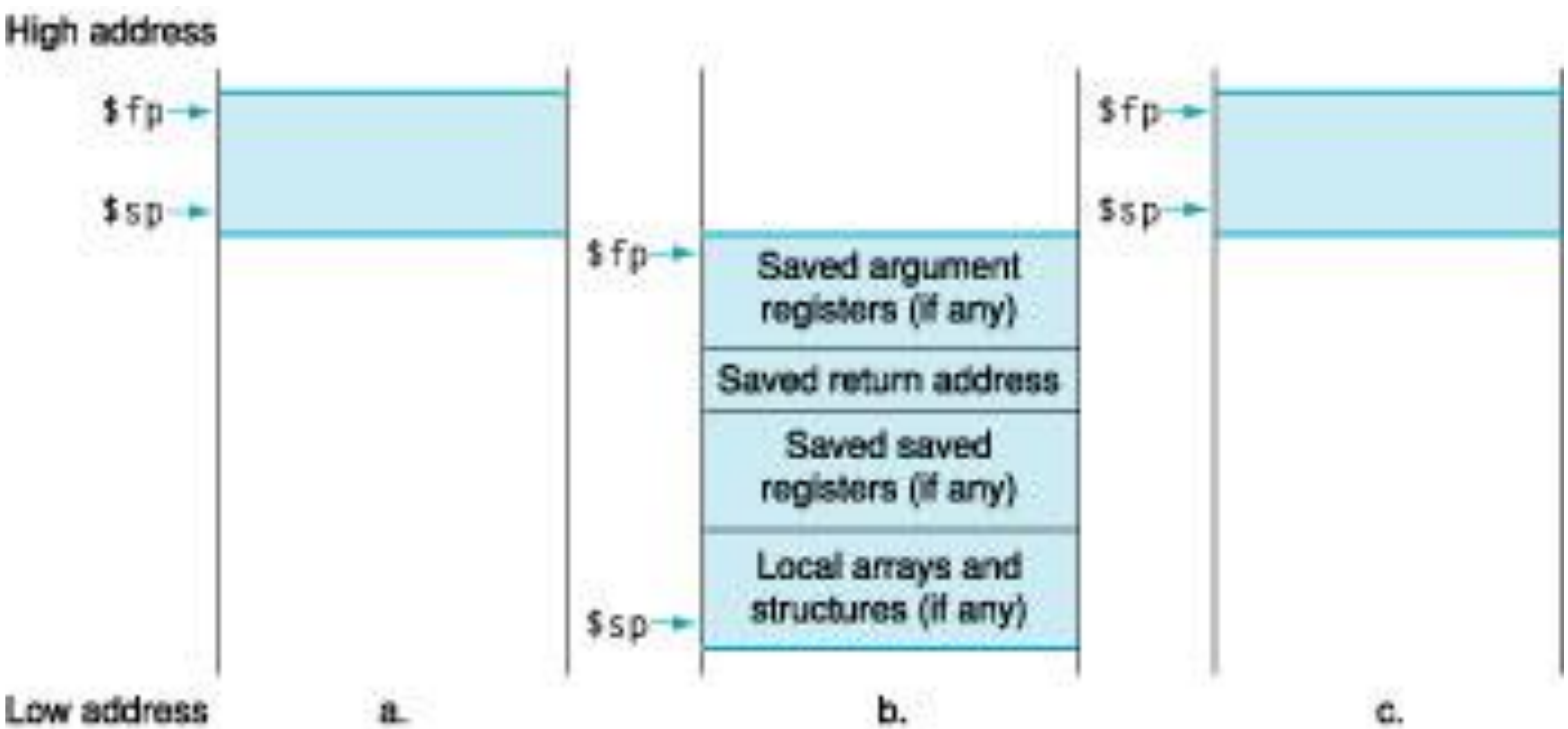
2.8 Nested Procedure Compilation

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

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
```

2.8 More About the Stack

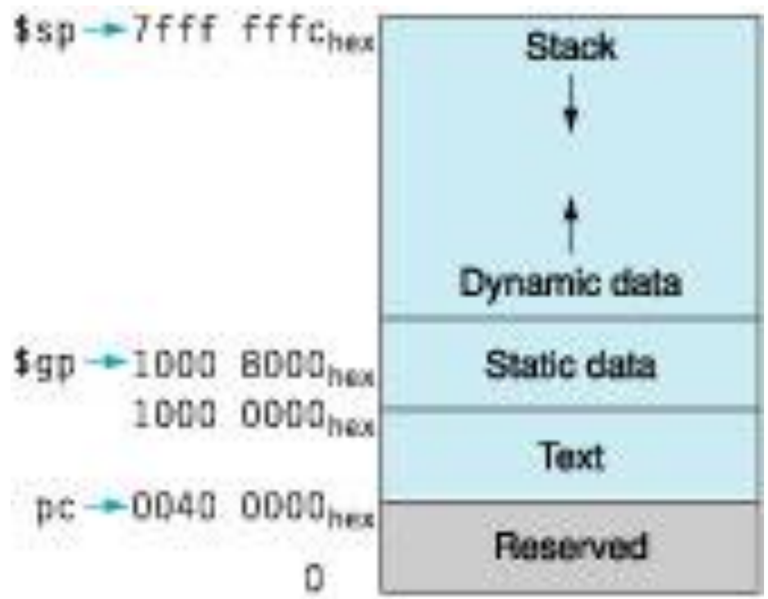
- Allocating Space for Automatic Variables
  - In addition to storing saved registers, the stack holds local variables that don't fit into registers, e.g., arrays, structs.
  - Saved registers + Local Variables = Procedure Frame



2.8 The Heap

- Space is needed for _______ variables and ________ data structures
  - Space is reserved and freed on the heap using ________ ______ ______.
- Register Usage

$zero	0
$v0-$v1	2-3
$a0-$a3	4-7
$t0-$t7	8-15
$s0-$s7	16-23
$t8-$t9	24-25
$gp	28
$sp	29
$fp	30
$ra	31



2.10 32-bit Immediate Operands

- 32-Bit Immediate Operands
  - Upper 16 Bits - **`lui`**
  - Lower 16 Bits - **`ori`**

- Loading `0x003D 0900`

2.10 32-Bit Addresses

- Addresses in Branches and Jumps

```
j 10000
```

```
bne $s0, $s1, Exit
```

  – Elaboration: For jumps, we give only 28 bits, from whence springeth the other 4?

  – Branching Far Away

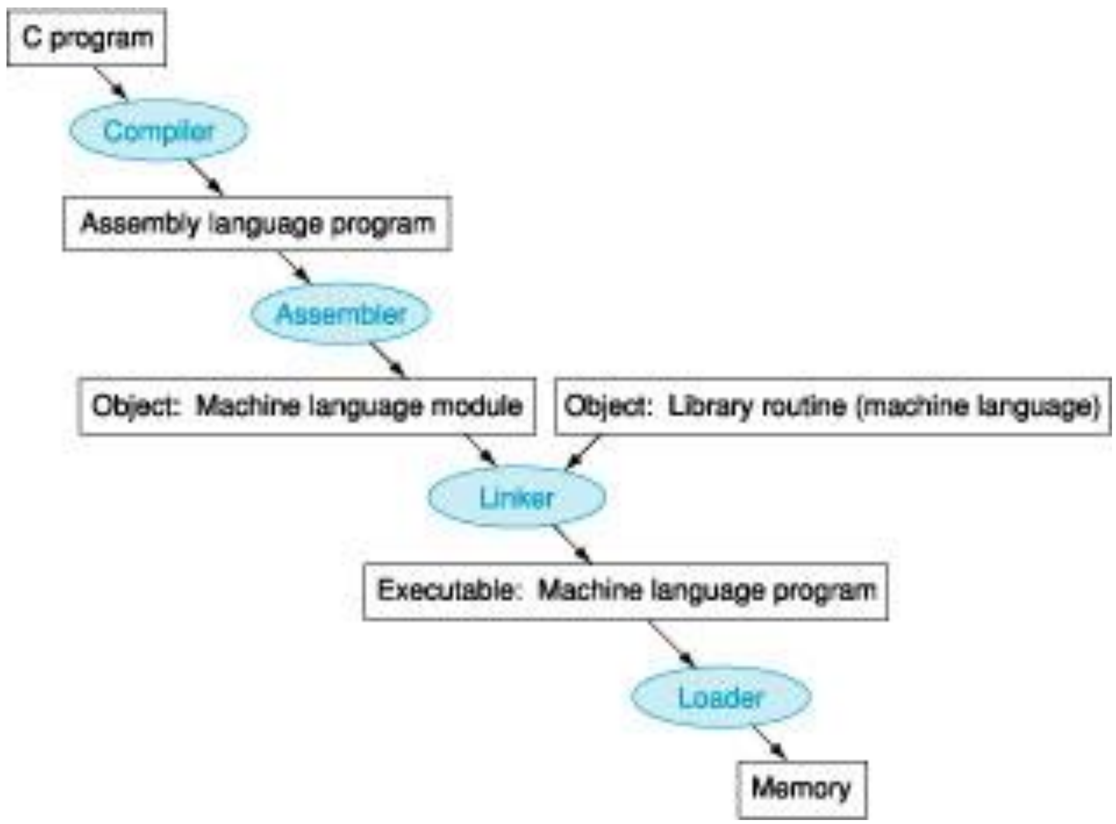
2.11 Parallelism and Instructions: Synchronization

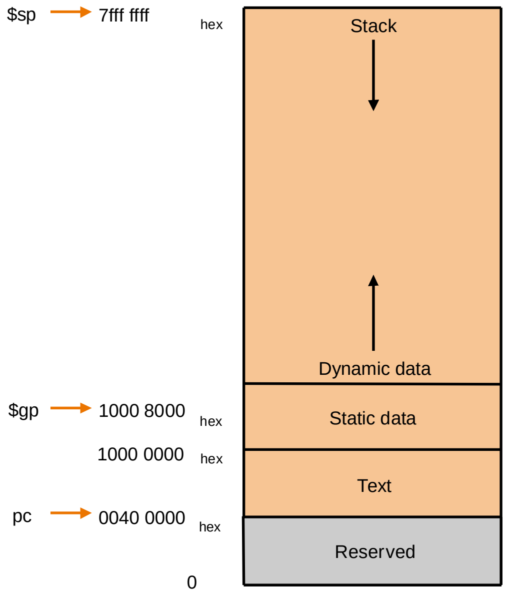
- Cooperation between tasks usually means some tasks are _________ new values that others must _____
- In computing, synchronization mechanisms are typically built with _____________ software routines that rely on __________-supplied synchronization instructions
- One hardware primitive will both read from and write to a location in one _________ operation
- The other approach is to have a ___ of instructions in which the _________ instruction ________ __ ________ showing whether the pair of instructions was executed as if the pair were atomic
- MIPS has ______ _______, ___, and ______ _______________, ___

2.11 Code Sequence for Atomic Exchange

```
again: addi $t0,$zero,1       ;copy locked value
       ll   $t1,0($s1)        ;load linked
       sc   $t0,0($s1)        ;store conditional
       beq  $t0,$zero,again ;branch if store fails
       add  $s4,$zero,$t1    ;put load value in $s4
```

2.12 Translating and Starting a Program





2.20 Fallacies and Pitfalls

- Fallacy: More powerful instructions mean higher performance.
- Fallacy: Write in assembly language to obtain the highest performance.
- Fallacy: The importance of commercial binary compatibility means successful instruction sets don't change.
- Pitfall: Forgetting that sequential word addresses in machines with byte addressing do not differ by one.
- Pitfall: Using a pointer to an automatic variable outside its defining procedure.