Next: Many Examples: **Up:** Chapter 4: Instruction Set **Previous:** An Example: the M68000

Instruction Set of MIPS Processor

- **Register file (RF):** 32 registers ($0 through $31), each for a word of 32 bits (4 bytes);
  - $0 always holds zero
  - $sp (29) is the stack pointer (SP) which always points to the top item of a stack in the memory;
  - $ra (31) always holds the return address from a subroutine

  The table below shows the conventional usage of all 32 registers.

Register Number	Mnemonic Name	Conventional Use	Register Number	Mnemonic Name	Conventional Use
$0	zero	Permanently 0	$24, $25	$t8, $t9	Temporary
$1	$at	Assembler Temporary (reserved)	$26, $27	$k0, $k1	Kernel (reserved for OS)
$2, $3	$v0, $v1	Value returned by a subroutine	$28	$gp	Global Pointer
$4–$7	$a0–$a3	Arguments to a subroutine	$29	$sp	Stack Pointer
$8–$15	$t0–$t7	Temporary (not preserved across a function call)	$30	$fp	Frame Pointer
$16–$23	$s0–$s7	Saved registers (preserved across a function call)	$31	$ra	Return Address

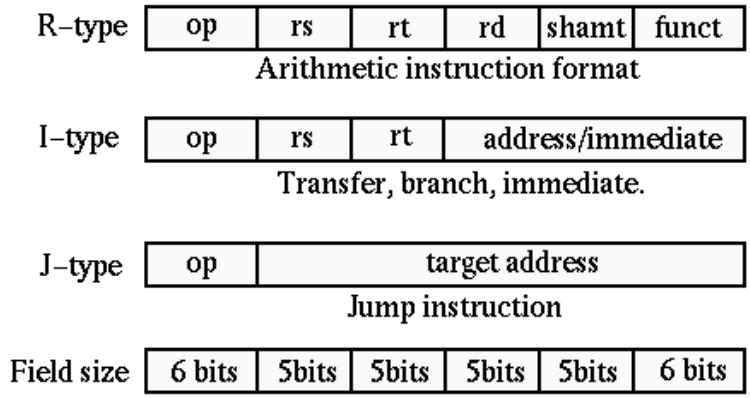
MIPS registers

- **Main memory (MM):** 2^{32} addressable bytes ($0, 1, 2, 3, \cdots, 2^{32}-1 = 4,294,967,295$, 4 Gb) or 2^{30} words ($0, 4, 8, 12, \cdots, 2^{32}-4 = 4,294,967,292$).

$2^{32}-1$	$2^{32}-2$	$2^{32}-3$	$2^{32}-4$
....	....	....	....
19	18	17	16
15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

MIPS virtual memory

- **Instruction set**: each instruction in the instruction set describes one particular CUP operation. Each instruction is represented in both **assembly language** by the *mnimonics* and **machine language** (binary) by a word of 32 bits subdivided into several fields.



There are different types of instructions:

1. Computational Instructions

   These instructions are for arithmetic or logic manipulations. In general they operate on two operands and store the result.

op $rd, $rs, $rt

   $rd ← $rs * $rt

   where

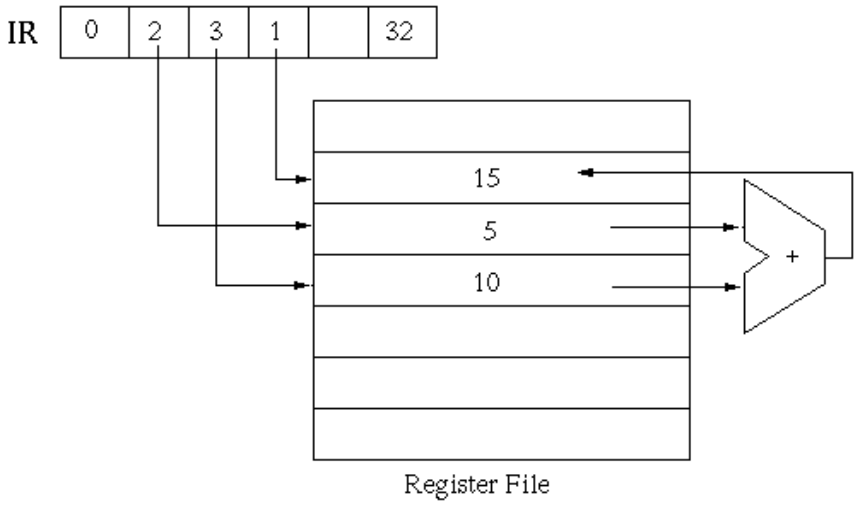
   - op: opcode specifying the arithmetic/logic operation to be performed;
   - $rd: destination register in which the result is to be stored;
   - $rs: source register containing the 1st operand;
   - $rt: source register containing the 2nd operand.

   Opcode can be: add, sub, mult, div, and, or, etc.

   $rd, $rs, $rt can be any of the 32 registers.

   The assembly instruction: add $1 $2 $3

   The machine instruction:

   Note:

   - The destination register is specified in the first field following the opcode field in the assembly language instruction, but the last 5-bit field in the binary machine language instruction.
   - In all R-type data manipulation instructions (arithmetic, logical, shift), the operations are specified by the function field (6 least significant bits) in the binary instruction, with the opcode field (6 most significant bits) all equal to zero.

Note:

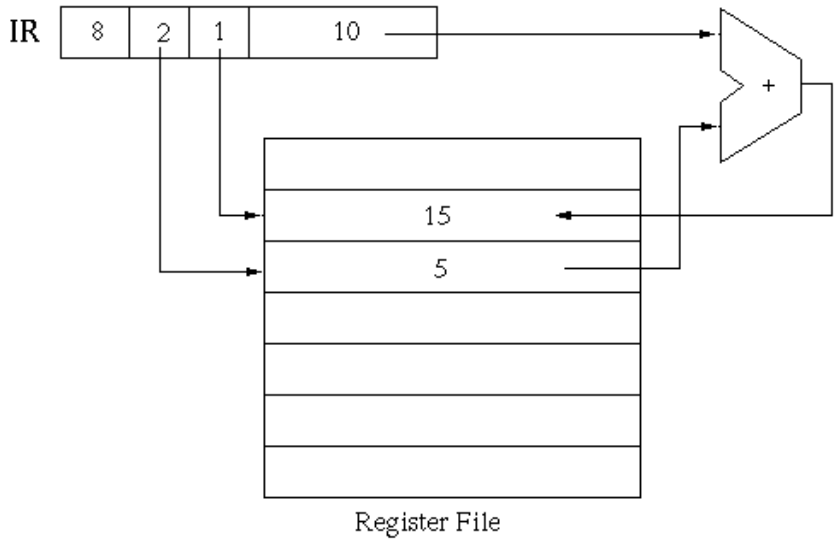
Immediate mode - replacing $rt by a constant:

op $rd, $rs, constant

where op can only be: add, and, or.

The assembly instruction: addi $1 $2 10

The machine instruction:



2. Shift Instructions

op $rd, $rs, shamt

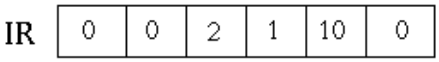
where op can be sll (shift left logical) srl (shift right logical) or sra (shift right arithmetic), and $shamt$ specifies the number of bits to shift. The shift amount can also be specified by a variable in a register, such as the example below ($t1 holds the shift amount):

srlv $rd, $rs, $t1

Examples:

The assembly instruction: sll $1 $2 10
($1 ← $2 << 10)

The machine instruction:



The assembly instruction: srl $1 $2 10
($1 ← $2 >> 10)

The machine instruction:

3. Data Transfer Instructions

These instructions transfer data back and forth between the MM and the CPU.

- Load word from MM to register:

```
lw   $rd, offset($rs)
```

$rd ← Memory[offset + $rs]

where

- lw: the opcode for *load word*;
- $rd: destination register into which the word is to be loaded;
- $rs: source register (e.g., an index number of an array);
- offset: (e.g., the beginning address of the array in memory).
- effective MM address: offset + $rs

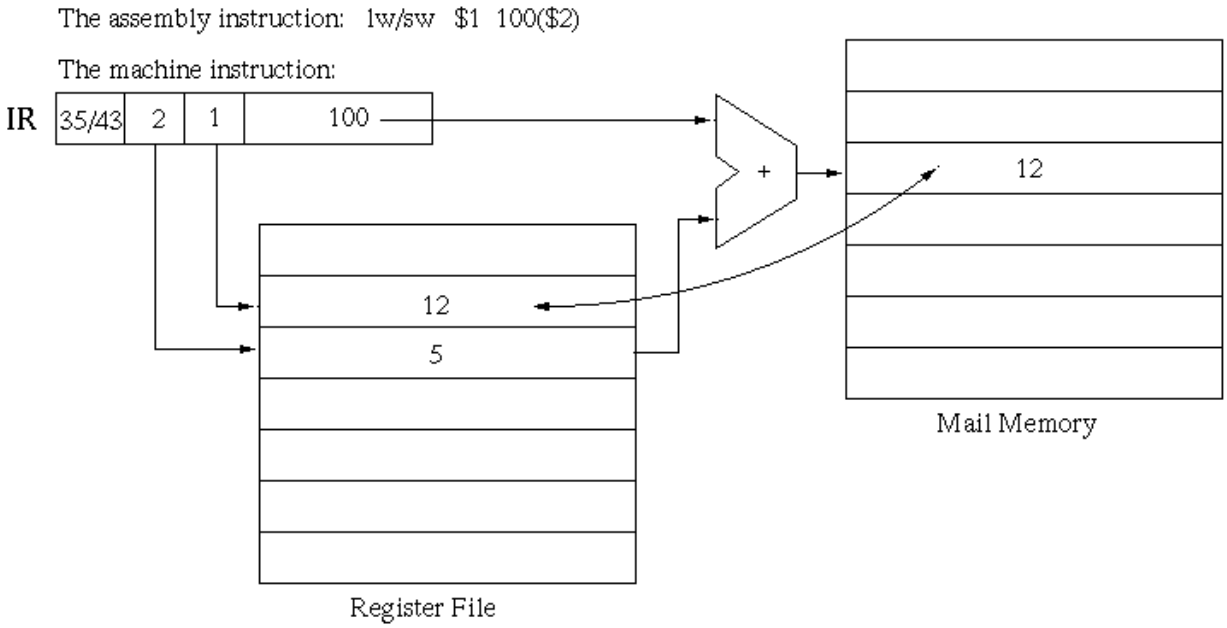
- Store word from register to memory:

```
sw   $rs, offset($rd)
```

$rs → Memory[offset + $rd]

where

- sw: the opcode for *save word*;
- $rs: source register whose content is to be stored;
- $rd: destination register, e.g., an index number of an array;
- offset: e.g., the beginning address of the array in memory.
- MM address: offset + $rd



4. Program Control

Usually the program is executed in the straight line fashion, i.e., the next instruction to be executed is the one that follows the previous one currently being executed. But some time it is needed to conditionally or unconditionally jump to some other part of the program (e.g., functions, loops, etc.) by the program control instructions.

- Branch to a labeled statement if two variables are equal:

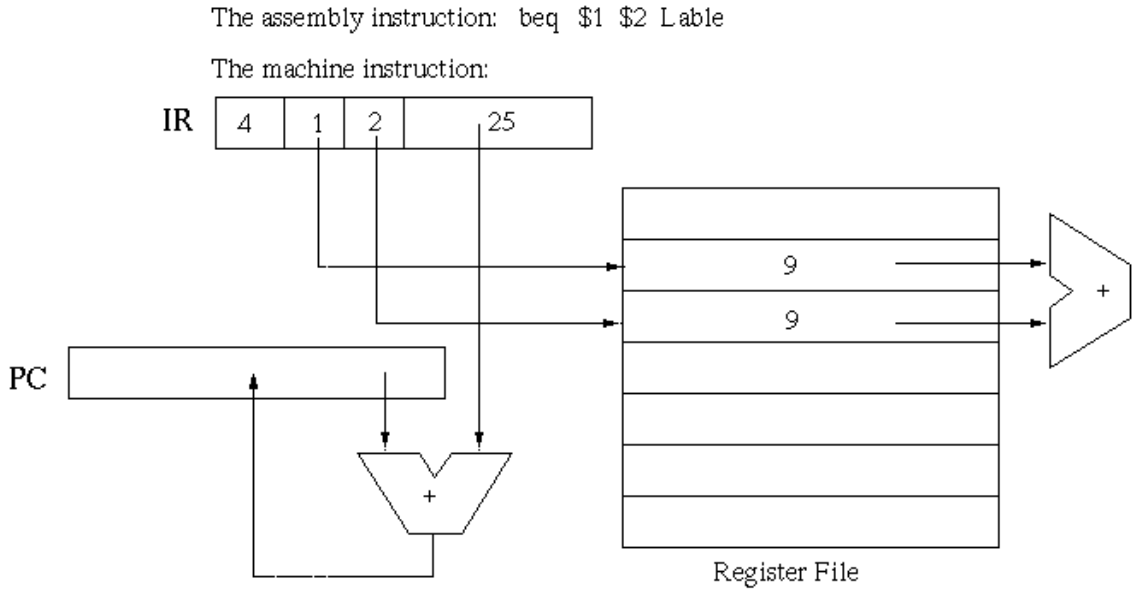
```
beq   $rd, $rs, L1
```

or not equal:

```
bne   $rd, $rs, L1
```

where

- beq (bne): branch if equal (not equal);
- $rd, $rs: two registers holding the variables;
- L1: label for the target statement.



- Set a register to 1 if first variable is smaller than the second, set the register to 0 otherwise.

```
slt   $rd, $rs, $rt
```

where

- slt: set if less than
- $rs, $rt: registers containing two variables to be compared;
- $rd: register to be set to 1 if $rs $<$ $rt, or 0 otherwise.

Immediate mode -- replacing $rt by a constant:

```
slti   $rd, $rs, constant
```

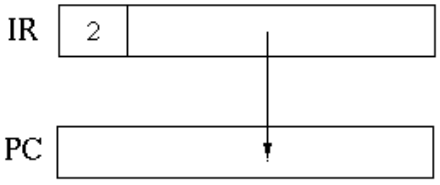
set $rd to 1 if $rs $<$ constant.

- Jump unconditionally to a certain labeled statement:

```
j   label
```

The assmebly instruction: j Label

The machine instruction:



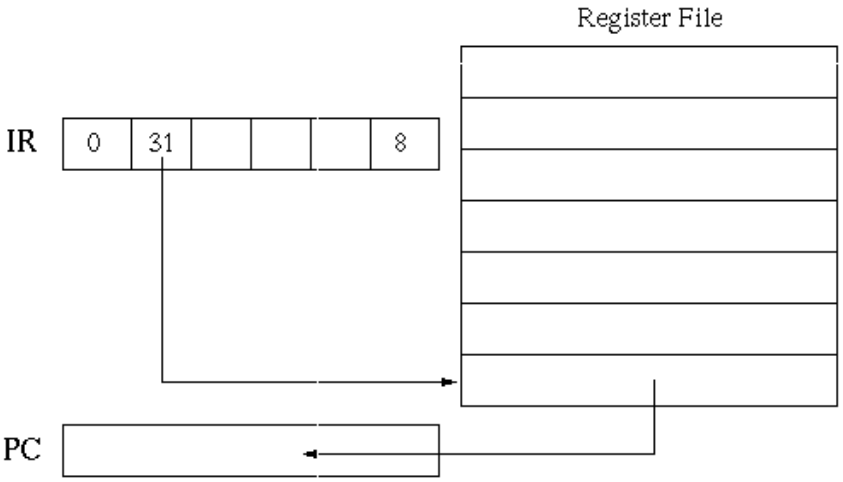
- Jump unconditionally to a certain statement whose address is given in a register:

```
jr   $rd
```

- Jump unconditionally to a subroutine (or function, procedure, etc.) whose address is given as a symbol SubroutineAddress:

The assmebly instruction: jr $31

The machine instruction:




```
jal   SubroutineAddress
```

Operations related to function calls:

- Branch to a subroutine by jal (jump and link)
  1. update PC ($PC \leftarrow PC + 4$) to point to the next instruction;
  2. save content of PC into register $ra ($31) as the return address;
  3. load starting address of subroutine (symbolized as SubroutineAddress) to PC.
- Return to calling routine:

  To return from subroutine to the calling routine, the last statement of the subroutine must be:

```
jr   $ra
```

- Recursion: For recursive subroutine calls, previous content of $ra ($31) needs to be stored in a *stack* in the memory.

MIPS R3000 Instruction Set

MIPS Machine Language

Next: Many Examples: **Up:** Chapter 4: Instruction Set **Previous:** An Example: the M68000

Ruye Wang 2003-10-30