JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self-Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8491 – COMPUTER ARCHITECTURE

(Regulation 2017)

Year/Semester: II/IV CSE

2020 – 2021

Prepared by

Ms. R. Revathi

Assistant Professor/CSE

UNIT I BASIC STRUCTURE OF A COMPUTER SYSTEM 9

Functional Units – Basic Operational Concepts – Performance – Instructions: Language of the Computer – Operations, Operands – Instruction representation – Logical operations – decision making – MIPS Addressing.

ALU or DATAPATH UNIT

- It performs the arithmetic operations.
- It is also called as brawn of the processor.
- The entire operation can be done with the help of registers.
- Registers are faster than memory.
- Registers can hold variables and intermediate results.
- Memory traffic is reduced, so program runs faster.

CONTROL UNIT

- It is also called as a brain of the processor.
- It fetches and analyses the instructions one-by-one and issue control signals to all other units to perform various operations.
- For a given instruction, the exact set of operations required is indicated by the control signals. The results of instructions are stored in memory.
- The component of the processor that commands the datapath, memory, and I/O devices according to the instructions of the program.

MEMORY UNIT

- Memory is nothing but a storage device. It stores the program and data.
- It is divided into 'n' number of cells. Each cell is capable of storing one bit information at a time.
- There are 2 classes of memory.1.Primary 2.Secondary.

Primary Memory

- It is made up of semiconductor material. So it is called Semiconductor memory
- Data storage capacity is less than secondary memory.
- Cost is too expensive than secondary memory.
- CPU can access data directly. Because it is an internal memory.
- Data accessing speed is very fast than secondary memory.
- Ex.RAM & ROM

RAM	ROM
Random Access Memory	Read Only Memory
Volatile memory	Non volatile memory
Data lost when the power turns off and that is used to hold data and program while they are running.	It retains data even in the absence of a power source and that is store programs between runs.
Temporary storage medium	Permanent storage medium
User perform both read and write operation	User can perform only read operation

RAM: There are two types of memory available namely, 1. SRAM 2. DRAM

SRAM	DRAM
Static RAM	Dynamic RAM
Information is stored in 1 bit cell called Flip Flop.	Information is represented as charge across a capacitor

Information will be available as long as power is available	It retains data for few ms based on the charge of capacitor even in the absence of a power
No refreshing is needed	Refreshing is needed
ess packaging density	High packaging density
More complex Hardware	Less complex hardware
More expensive	Less expensive
No random access	Random access is possible
Access time 10 ns	Access time 50 ns

Cache Memory: A small, fast memory that acts as a buffer for a slower, larger memory.

Secondary memory

- Secondary memory (Nonvolatile storage) is a form of storage that retains data
- even in the absence of a power source and that is used to store the programs between runs.
- It is made up of magnetic material. So it is called magnetic memory.
- Data storage capacity is high than primary memory.
- Cost is too low than primary memory.
- CPU cannot access data directly. Because it is an external memory.
- Data accessing speed is very slow than primary memory.
- Ex. Magnetic disk, Hard Disk, CD, DVD, Floppy Disk

Magnetic Disk

- It consists of a collection of platters, which rotate on a spindle at 5400 revolution/min.
- The metal platters are covered with magnetic recording material on both sides.
- Also called hard disk. A form of nonvolatile secondary memory composed ofrotating platters coated with a magnetic recording material.
- Because they are rotating mechanical devices, access times are about 5 to 20 milliseconds and cost per gigabyte in 2012 was $0.05 to $0.10.

Optical Disk: Include both Compact Disk (CD) and Digital Video Disk(DVD).

Read-Only CD/DVD

- Data is recorded in a spiral fashion, with individual bits being recorded by burning small pit.
- The disk is read by shining a laser at the CD surface and determining by examining the reflected light whether there is a pit or flat surface.

Rewritable CD/DVD

- Use different recording surface that as a crystal line, reflective materials, pits are formed that are not reflective.

Erase CD/DVD

- The surface is heated and cooled slowly, allowing an annealing process to restore the surface recording layer to its crystalline structure.

Flash Memory

- A nonvolatile semiconductor memory. It is cheaper and slower than DRAM but more expensive per bit and faster than magnetic disks. Access times are about 5 to 50 microseconds and cost per gigabyte in 2012 was $0.75 to $1.00.

OUTPUT UNIT

- A mechanism that conveys the result of a computation to a user, such as a display, or to another computer.

Liquid Crystal Display

- A display technology using a thin layer of liquid polymers that can be used to transmit or block light according to whether a charge is applied.
- All laptops and desktop computers use Liquid Crystal Display (LCD) to get a thin, low-power display. A tiny transistor switch at each pixel to control current and make sharper images.
- The image is composed of a matrix of picture elements or pixels, which can be represented as a matrix of bits called **bitmap.**
- Depending on the size of the screen and the resolution, the display matrix ranges in size from 640*480 to 2560*1600.
- A red, green, blue (RGB) associated with each dot on the display determines the intensity of the three color components in the final image.
- **Pixel:** The smallest individual picture element. Screens are composed of hundreds of thousands to millions of pixels, organized in a matrix.

BASIC OPERATIONAL CONCEPTS

To perform a given task an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be stored are also stored in the memory.

Examples: Add LOCA, R0

This instruction adds the operand at memory location LOCA, to operand in register R0 & places the sum into register. This instruction requires the performance of several steps,

1. First the instruction is fetched from the memory into the processor
2. The operand at LOCA is fetched and added to the contents of R0
3. Finally the resulting sum is stored in the register R0

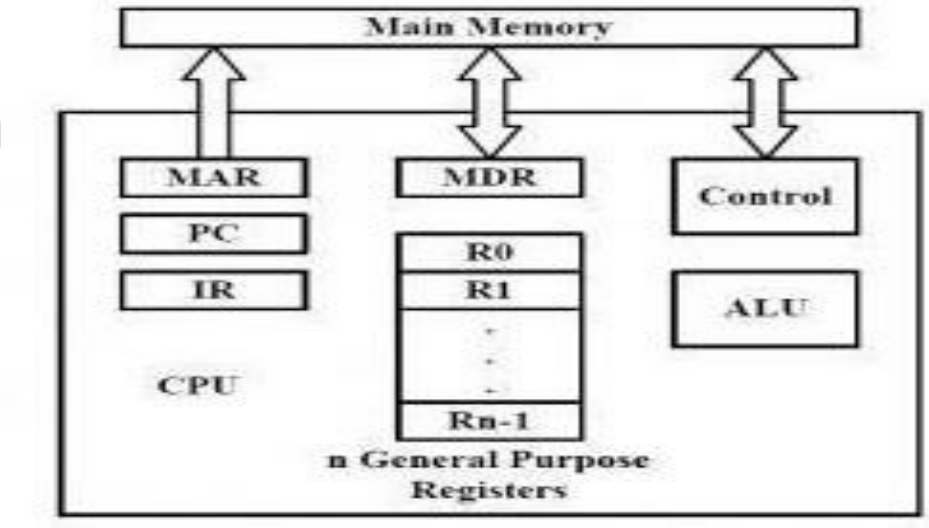


Fig: Connections between the processor and the memory

The fig shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains a number of registers used for several different purposes.

Instruction Register (IR)

- Holds the instruction that is currently being executed.
- Its output is available for the control circuits which generates the timing signals that

control the various processing elements in one execution of instruction.

Program Counter PC

- This is another specialized register that keeps track of execution of a program.
- It contains the memory address of the next instruction to be fetched and executed. Besides IR and PC, there are n-general purpose registers R0 through Rn-1. MAR PC IR MEMORY MDR R0 R1 …

The other two registers which facilitate communication with memory are:

1. MAR – (Memory Address Register):- It holds the address of the location to be accessed.

2. MDR – (Memory Data Register):- It contains the data to be written into or read out of the address location.

Operating steps are

1. Programs reside in the memory & usually get these through the I/P unit.
2. Execution of the program starts when the PC is set to point at the first instruction of the program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiating a read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.
9. After one or two such repeated cycles, the ALU can perform the desired operation.
10. If the result of this operation is to be stored in the memory, the result is sent to MDR.
11. Address of location where the result is stored is sent to MAR & a write cycle is initiated.
12. The contents of PC are incremented so that PC points to the next instruction that is to be executed.

PERFORMANCE

The machine (or CPU) is said to be faster or has better performance running this program if the total execution time is shorter. When trying to choose among different computers, performance is an important attribute.

1. Throughput and Response Time

Response time

- Response time also called execution time. The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.

Throughput or bandwidth

- The total amount of work done in a given time. Decreasing response time almost always improves throughput.
- To maximize performance, we want to minimize response time or execution time for some task. Thus, we can relate performance and execution time for a computer X:

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

This means that for two computers X and Y, if the performance of X is greater than the performance of Y, we have

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$

$$\text{Execution time}_Y > \text{Execution time}_X$$

That is, the execution time on Y is longer than that on X, if X is faster than Y.
Y"—or equivalently "X is n times as fast as Y"—to mean

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

If X is n times as fast as Y, then the execution time on Y is n times as long as it is on X:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

Relative Performance

If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times as fast as B.

In the above example, we could also say that computer B is 1.5 times slower than computer A

2. Measuring Performance

CPU execution time: Also called **CPU time**. The actual time the CPU spends computing for a specific task.

User CPU time: The CPU time spent in a program itself.

System CPU time: The CPU time spent in the operating system performing tasks on behalf of the program.

Clock cycle: Also called **tick**, **clock tick**, **clock period**, **clock**, or **cycle**. The time for one clock period, usually of the processor clock, which runs at a constant rate.

Clock period: This is the length of each clock cycle.

Clock rate*:* This is the inverse of the clock period.

3. CPU Performance and Its Factors

- A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time.
- This formula makes it clear that the hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Example

Improving Performance

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

Let's first find the number of clock cycles required for the program on A:

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A}$$

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \frac{\text{cycles}}{\text{second}}}$$

$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 2 \times 10^9 \frac{\text{cycles}}{\text{second}} = 20 \times 10^9 \text{ cycles}$$

CPU time for B can be found using this equation:

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = \frac{0.2 \times 20 \times 10^9 \text{ cycles}}{\text{second}} = \frac{4 \times 10^9 \text{ cycles}}{\text{second}} = 4 \text{ GHz}$$

To run the program in 6 seconds, B must have twice the clock rate of A.

4. Instruction Performance

- Execution time is that it equals the number of instructions executed multiplied by the average time per instruction.
- Therefore, the number of clock cycles required for a program can be written as

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \frac{\text{Average clock cycles}}{\text{per instruction}}$$

Clock cycles per Instruction

- The term clock cycles per instruction, which is the average number of clock cycles each instruction takes to execute, is often abbreviated as CPI.
- Since different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in the program.

Example

Suppose we have two implementations of the same instruction set architecture.Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program,and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the sameprogram. Which computer is faster for this program and by how much?

We know that each computer executes the same number of instructions for the program; let's call this number I. First, find the number of processor clock cycles for each computer:

$$\text{CPU clock cycles}_A = I \times 2.0$$
$$\text{CPU clock cycles}_B = I \times 1.2$$

Now we can compute the CPU time for each computer:

$$\text{CPU time}_A = \text{CPU clock cycles}_A \times \text{Clock cycle time}$$
$$= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}$$

Likewise, for B:

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Clearly, computer A is faster. The amount faster is given by the ratio of the execution times:

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

We can conclude that computer A is 1.2 times as fast as computer B for this program.

5. The Classic CPU Performance Equation

We can now write this basic performance equation in terms of instruction count (the number of instructions executed by the program), CPI, and clock cycle time:

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

or, since the clock rate is the inverse of clock cycle time:

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

$$\text{Time} = \text{Seconds/Program} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

Example

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a particular computer. The hardware designers have supplied the following facts:

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code sequence	Instruction counts for each instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Sequence 1 executes 2 + 1 + 2 = 5 instructions. Sequence 2 executes 4 + 1 + 1 = 6 instructions. Therefore, sequence 1 executes fewer instructions.

We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

This yields

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$

$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

Amdahl's Law

A rule stating that the performance enhancement possible with a given improvement is

limited by the amount that the improved feature is used.

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

Million Instructions per Second (MIPS)

- A measurement of program execution speed based on the number of millions of instructions.
- MIPS are computed as the instruction count divided by the product of the execution time. MIPS is easy to understand, and faster computers.

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

- There are three problems with using MIPS as a measure for comparing computers.
- First, we cannot compare computers with different instruction sets using MIPS, since the instruction counts will certainly differ.
- Second, MIPS varies between programs on the same computer; thus, a computer cannot have a single MIPS rating.
- For example, by substituting for execution time, we see the relationship between MIPS, clock rate, and CPI:

$$\text{MIPS} = \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- Finally, and most importantly, if a new program executes more instructions but each instruction is faster, MIPS can vary independently from performance.

INSTRUCTION

The words of a computer‘s language are called instructions, and its vocabulary is called an instruction set. Instruction performs one of the following operations

- Data transfer between register and memory.
- ALU operation.
- Program control and sequencing.
- I/O transfer.

Stored program concept

The idea that instructions and data of many types can be stored in memory as numbers, leading to the stored program computer.

OPERATIONS OF THE COMPUTER HARDWARE

1. Arithmetic Operations

Add and Subtract instruction use three Operands - Two Sources and one Destination.

Example add $t0, $s1, $s2

2. Data Transfer Operations

These operations help in moving the data between memory and registers.

xample: 1

C code:

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

- g in $s1, h in $s2, base address of A in $s3

Compiled MIPS code:

- Index 8 requires offset of 32
  - 4 bytes per word

```
lw  $t0, 32($s3)     # load word
add $s1, $s2, $t0
```

offset — base register

Example: 2

C code:

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

- h in $s2, base address of A in $s3

Compiled MIPS code:

- Index 8 requires offset of 32

```
lw  $t0, 32($s3)     # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)     # store word
```

3. Logical Operations

Logical Instructions are used for bitwise manipulation. It is useful for extracting and inserting groups of bits in a word

Operation	C	MIPS
Shift left	<<	sll
Shift right	>>	srl
Bitwise AND	&	and, andi
Bitwise OR	|	or, ori
Bitwise NOT	~	nor

4. Conditional Operations

1. Branch to a labeled instruction if a condition is true. Otherwise, continue sequentially.

   beq rs, rt, L1
   if (rs == rt) branch to instruction labeled L1;
   bne rs, rt, L1
   if (rs != rt) branch to instruction labeled L1;
   j L1

2. Unconditional jump to instruction labeled L1. Set result to 1 if a condition is true. Otherwise, set to 0.

   slt rd, rs, rt
   if (rs < rt) rd = 1; else rd = 0;
   slti rt, rs, constant
   if (rs < constant) rt = 1; else rt = 0;

3. Use in combination with beq, bne

   slt $t0, $s1, $s2 # if ($s1 < $s2)
   bne $t0, $zero, L # branch to L
   Jump Instructions

Jump Address Table: Also called **jump table**. A table of addresses of alternative instruction sequences.

4. Procedure call: jump and link.

Jump-and-Link Instruction: An instruction that jumps to an address and simultaneously saves the address of the following instruction in a register($rain MIPS).

   jal ProcedureLabel
   Address of following instruction put in $ra.
   Jumps to target address.

5. Procedure return: jump register

Return address: A link to the calling site that allows a procedure to return to the proper address in MIPS it is stored in register $ra.

jr $ra
Copies $ra to program counter.
Can also be used for
computed jumps. e.g.,
for case/switch
statements.

Procedure: A stored subroutine that performs a specific task based on the parameters with which it is provided.

Caller: The program that instigates a procedure and provides the necessary parameter values.

Callee: A procedure that executes a series of stored instructions based on parameters provided by the caller and then returns control to the caller.

Program counter (PC): The register containing the address of the instruction in the program being executed.

Stack: A data structure for spilling registers organized as a last-in first-out queue.

Stack pointer: A valued noting the most recently allocated address in a stack that shows where registers should be spilled or where old register values can be found. In MIPS, it is register $sp.

Push: Add element to stack.

Pop: Remove element from stack.

Global pointer: The register that is reserved to point to the static area.

OPERANDS OF THE COMPUTER HARDWARE

1. Register Operands

- Arithmetic instructions use register operands and MIPS has a 32 × 32-bit register file
- Mainly used for frequently accessed data
- Register numbered 0 to 31 and 32-bit data called a ―word
- $t0, $t1, …, $t9 for temporary values
- $s0, $s1, …, $s7 for saved variables

Word: The natural unit of access in a computer, usually a group of 32 bits; corresponds to the size of a register in the MIPS architecture.

Compiling a C Assignment Using Registers

Example: f = (g + h) – (i + j); The variables f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $s4, respectively. What is the compiled MIPS code?

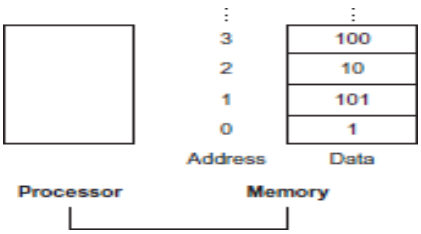
MIPS code

```
add $t0,$s1,$s2 # register $t0 contains g + h
add $t1,$s3,$s4 # register $t1 contains i + j
sub $s0,$t0,$t1 # f gets $t0 – $t1, which is (g + h)–(i + j)
```

2. Memory Operands

Data transfer instruction: A command that moves data between memory and registers.

Address: A value used to delineate the location of a specific data element within a memory array. For example, in the following figure, the address of the third data element is 2, and the value of Memory [2] is 10.



2.1 Load:

- The data transfer instruction that copies data from memory to a register is traditionally called load.
- The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.
- The sum of the constant portion of the instruction and the contents of the second register forms the memory address.
- The actual MIPS name for this instruction is lw, standing for load word.

Compiling an Assignment When an Operand Is in Memory

Example: Let's assume that A is an array of 100 words and that the compiler has associated the variables g and h with the registers $s1 and $s2 as before. Let's also assume that the starting address, or base address, of the array is in$s3. Compile this C assignment statement:

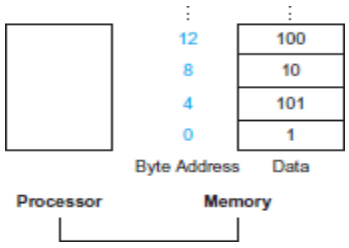
g = h + A[8];

Solution:

```
lw $t0,32($s3)      # Temporary reg $t0 gets A[8]
add $s1,$s2,$t0     # g = h + A[8]
```

The constant in a data transfer instruction (8) is called the off set, and the register added to form the address ($s3) is called the base register.

Alignment restriction: A requirement that data be aligned in memory on natural boundaries.



Little Endian Address: A lower byte address of the word is specified in least significant byte of the word is known as Little endian address.

Big Endian Address: A lower byte address of the word is specified in most significant byte of the word is known as big endian address.

2.2 Store:

- The instruction complementary to load is traditionally called store; it copies data from a register to memory. The format of a store is similar to that of a load.

- The actual MIPS name is sw, standing for store word.

Example:

Assume variable h is associated with register $s2 and the base address of the array A is in $s3. What is the MIPS assembly code for the C assignment statement below?

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

Solution:

```
lw $t0,32($s3)      # Temporary reg $t0 gets A[8]
add $t0,$s2,$t0     # Temporary reg $t0 gets h + A[8]
```

The final instruction stores the sum into A[12], using 48 (4×12) as the offset and register $s3 as the base register.

```
sw $t0,48($s3)      # Stores h + A[8] back into A[12]
```

3. Constant or Immediate Operands

- The constants would have been placed in memory when the program was loaded. For example, to add the constant 4 to register $s3.
- we could use the code

```
lw $t0, AddrConstant4($s1)      # $t0 = constant 4
add $s3,$s3,$t0                 # $s3 = $s3 + $t0 ($t0 == 4)
```

- This quick add instruction with one constant operand is called add immediate or addi. To add 4 to register $s3,We just write

```
addi $s3,$s3,4          # $s3 = $s3 + 4
```

- No subtract immediate instruction, Just use a negative constant

```
addi $s2,$s3,-1
```

REPRESENTING INSTRUCTIONS IN THE COMPUTER

- Since registers are referred to in instructions, there must be a convention to map register names into numbers.

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

Register numbers

$t0 – $t7 are reg‘s 8 – 15

$t8 – $t9 are reg‘s 24 – 25

$s0 – $s7 are reg‘s 16 – 23

Instruction format: A form of representation of an instruction composed of fields of binary numbers.

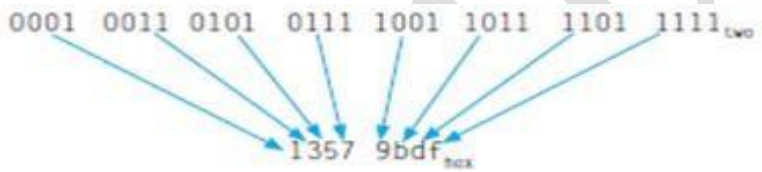
Machine language: Binary representation used for communication within a computer system.
Hexadecimal: Numbers in base 16. The following table shows Hexadecimal to binary conversion.

Example:

Convert the following hexadecimal and binary numbers into the other base:

1. eca8 6420$_{hex}$

2. 0001 0011 0101 0111 1001 1011 1101 1111$_{two}$



Format Types:

1. R-Format
2. I-Format
3. J-Format

1. R-Type Format:

Opcode: The field that denotes the operation and format of an instruction.

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

Instruction Fields:

op: operation code (opcode)
rs: first source register number
rt: second source register number
rd: destination register number
shamt: shift amount (00000 for now)
funct: function code (extends opcode)

Example:

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$$00000010001100100100000000100000_2 = 02324020_{16}$$

2. I-Type Format:

A second type of instruction format is called I-type (for immediate)or I-format and is used by the immediate and data transfer instructions. The fields of I-format are

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

the hardware knows whether to treat the last half of the instruction as three fields(R-type) or as a single field (I-type).

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

Example:

Translating MIPS Assembly Language into Machine Language

We can now take an example all the way from what the programmer writes to what the computer executes. If `$t1` has the base of the array `A` and `$s2` corresponds to `h`, the assignment statement

```
A[300] = h + A[300];
```

is compiled into

```
lw   $t0,1200($t1) # Temporary reg $t0 gets A[300]
add  $t0,$s2,$t0   # Temporary reg $t0 gets h + A[300]
sw   $t0,1200($t1) # Stores h + A[300] back into A[300]
```

What is the MIPS machine language code for these three instructions?

Solution:

Decimal Form:

Op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

Binary Form:

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

- The two MIPS instruction formats so far are R and I.
- The first 16 bits are the same: both contain an op field, giving the base operation; an rs field, giving one of the sources; and the rt field, which specifies the other source operand, except for load word, where it specifies the destination register.
- R-format divides the last 16 bits into an rd field, specifying the destination register; the shamt field, and the funct field, which specifies the specific operation of R-format instructions.
- I-format combines the last 16 bits into a single address field.

3. J-Format:

j target- J-type is short for "jump type". The format of an J-type instruction looks like:

Opcode	Address

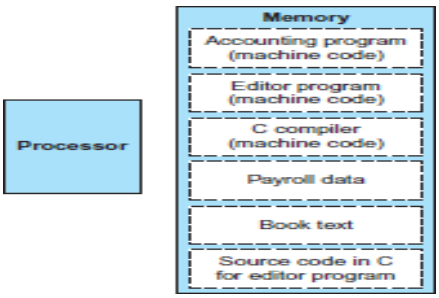
The semantics of the j instruction (j means jump) are:

PC <- PC31-28 IR25-0 00

where PC is the program counter, which stores the current address of the instruction being executed. You update the PC by using the upper 4 bits of the program counter, followed by the 26 bits of the target (which is the lower 26 bits of the instruction register), followed by two 0's, which creates a 32 bit address.

The stored-program concept:

- Stored programs allow a computer that performs by loading memory with programs and data and to begin executing at a given location in memory.



.

What MIPS instruction does this represent? Choose from one of the four options below.

op	rs	rt	rd	shamt	funct
0	8	9	10	0	34

1. sub $t0, $t1, $t2

2. add $t2, $t0, $t1

3. sub $t2, $t1, $t0

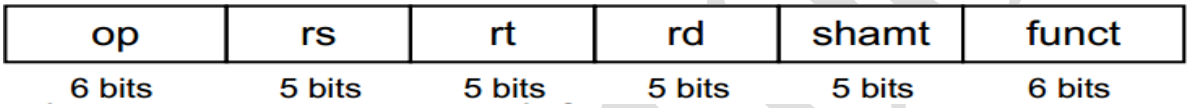
4. sub $t2, $t0, $t1

LOGICAL OPERATIONS

The packing and unpacking of bits into words. These instructions are called logical operations.

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

1. SHIFT:



shamt: how many positions to shift

Shift left logical

Shift left and fill with 0 bits
sll by i bits multiplies by 2i

Shift right logical

Shift right and fill with 0 bits
srl by i bits divides by 2i (unsigned only)

- For example, if register $s0contained

  $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_{two} = 9_{ten}$

- and the instruction to shift left by 4 was executed, the new value would be:

  $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_{two} = 144_{ten}$

- The dual of a shift left is a shift right. The actual name of the two MIPS shift instructions are called shift left logical (sll) and shift right logical (srl).
- The following instruction performs the operation above, assuming that the original value was in register $s0 and the result should go in register $t2:

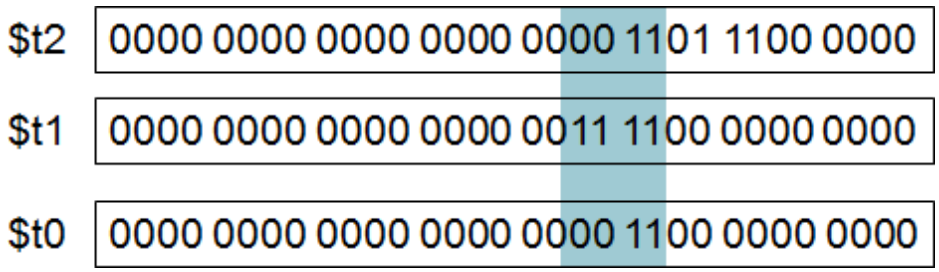
  sll $t2,$s0,4 # reg $t2 = reg $s0 << 4 bits

- The encoding of sll is 0 in both the op and funct fields, rd contains 10 (register $t2), rt contains 16 (register $s0), and shamt contains 4. The rs field is unused and thus is set to 0.Shifting left by i bits gives the same result as multiplying by 2i.

2. AND

- A logical bit by-bit operation with two operands that calculates a 1 only if there is a 1 in both operands.

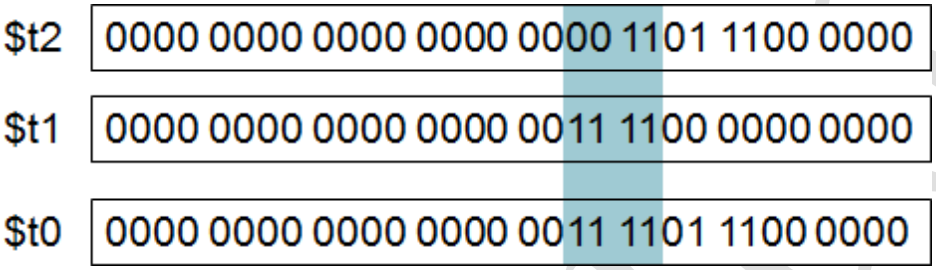
Ex: and $t0, $t1, $t2



3. OR

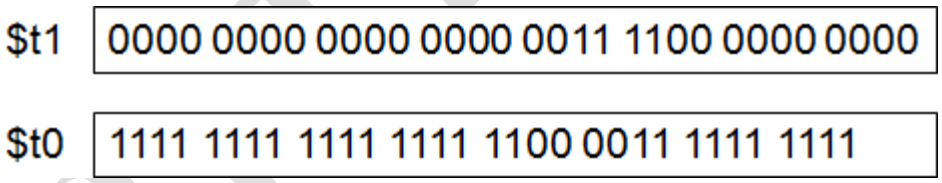
- A logical bit-by bit operation with two operands that calculates a 1 if there is a 1 in either operand.

  Ex: or $t0,$t1,$t2



4. NOT

- A logical bit-by bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.



5. NOR

- A logical bit-by bit operation with two operands that calculates the NOT of the OR of the two operands. That is, it calculates a 1 only if there is a 0 in both operands. If one operand is zero, then it is equivalent to NOT:

  A NOR 0= NOT (A OR 0) = NOT (A).

- If the register $t1 is unchanged from the preceding example and register $t3has the value 0, the result of the MIPS instruction \

  nor $t0,$t1,$t3 # reg $t0 = ~ (reg $t1 | reg $t3)

- register $t0

  $1111\ 1111\ 1111\ 1111\ 1100\ 0011\ 1111\ 1111_{two}$

INSTRUCTIONS FOR MAKING DECISIONS –CONTROL OPERATIONS

- It is ability to make decisions. Based on the input data and the values created during

computation.

- The first instruction is **beq register1, register2, L1**
- This instruction means go to the statement labeled L1 if the value in register1 equals the value in register2. The mnemonic beq stands for branch if equal.
- The second instruction is **bne register1, register2, L1**
- It means go to the statement labeled L1 if the value in register1 does not equal the value in register2. The mnemonic bne stands for branch if not equal. These two instructions are traditionally called conditional branches.

Compiling if-then-else into Conditional Branches

- In the following code segment, f, g, h, i, and j are variables. If the five variables f through j correspond to the five registers $s0 through $s4, what is the compiled MIPS code for this C if statement?

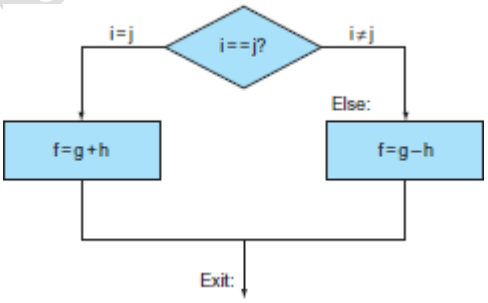
if (i == j) f = g + h; else f = g – h;

MIPS Code

```
bne $s3,$s4,else        # go to Else if i ≠ j
add $s0,$s1,$s2         # f = g + h (skipped if i ≠ j)
j Exit                  # go to Exit
else: sub $s0,$s1,$s2   # f = g – h (skipped if i = j)
Exit:
```

Conditional Branch

- n instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.



Loops

- Decisions are important both for choosing between two alternatives—found in if statements and for iterating a computation found in loops. The same assembly instructions are the building blocks for both cases.

Compiling a while Loop in C

- Here is a traditional loop in C:

while (save[i] == k)

```
i += 1;
```

- Assume that i and k correspond to registers $s3 and $s5 and the base of the array save is in $s6. What is the MIPS assembly code corresponding to this C segment?

MIPS Code:

```
Loop: sll $t1,$s3,2      # Temp reg $t1 = i * 4
add $t1,$t1,$s6          # $t1 = address of save[i]
lw $t0,0($t1)            # Temp reg $t0 = save[i]
bne $t0,$s5, Exit        # go to Exit if save[i] ≠ k
addi $s3,$s3,1           # i = i + 1
j Loop                   # go to Loop
Exit:
```

Basic Block

- A sequence of instructions without branches (except possibly at the end) and without branch targets or branch labels (except possibly at the beginning).

Set on Less Than Instruction

- Compares two registers and sets a third register to 1 if the first is less than the second; otherwise, it is set to 0.
- The MIPS instruction is called set on less than, or slt. For example,

  slt $t0, $s3, $s4 # $t0 = 1 if $s3 < $s4
- It means that register $t0 is set to 1 if the value in register $s3 is less than the value in register $s4; otherwise, register $t0 is set to 0.
- Constant operands are popular in comparisons, so there is an immediate version of the set on less than instruction. To test if register $s2 is less than the constant10, we can just write

  slti $t0,$s2,10 # $t0 = 1 if $s2 < 10

Signed versus Unsigned Comparison

- Suppose register $s0 has the binary number

  $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two}$ and that register $s1 has the binary number $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two}$

  What the values are of registers $t0 and $t1 after these two instructions?

Solution

```
slt $t0, $s0, $s1      # signed comparison
sltu $t1, $s0, $s1     # unsigned comparison
```

- The value in register $s0 represents -1_{ten} if it is an integer and $4,294,967,295_{ten}$ if it is an unsigned integer.
- The value in register $s1 represents 1_{ten} in either case. Then register $t0 has the value 1, since $-1_{ten} < 1_{ten}$, and register $t1 has the value 0, since $4,294,967,295_{ten} > 1_{ten}$.

MIPS ADDRESSING MODE SUMMARY

Multiple forms of addressing are generically called **addressing modes.** The following diagram shows how operands are identified for each addressing mode.

1. **Immediate addressing**, where the operand is a constant within the instruction itself
2. **Register addressing**, where the operand is a register
3. **Base or displacement addressing,** where the operand is at the memory location whose address is the sum of a register and a constant in the instruction
4. **PC-relative addressing**, where the branch address is the sum of the PC and a constant in the instruction
5. **Pseudo direct addressing**, where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC

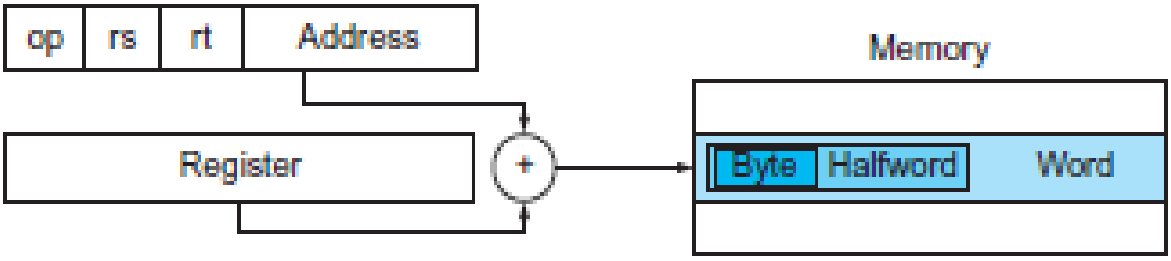
The following diagram shows all the MIPS instruction formats.

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/Immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

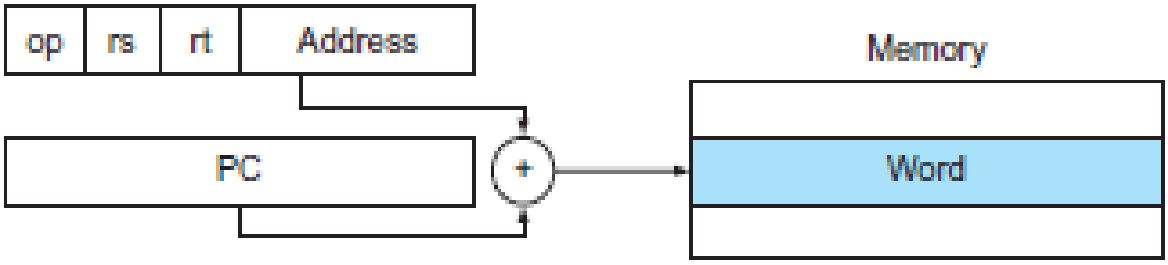
1. Immediate addressing

2. Register addressing

3. Base addressing



4. PC-relative addressing



5. Pseudodirect addressing