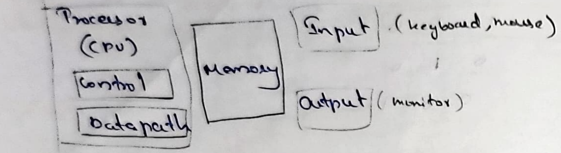
28/10/2023
Saturday

★ 5 classic components of a computer

info
Gmail
Essay
Ques.



- Input - To input data
- Output - To output the result.
- Memory - Main memory. — RAM (volatile memory) (in GB) — ROM (volatile ~~non~~ permanent)
  Large programs are saved on hard disk (secondary memory) (in TB)

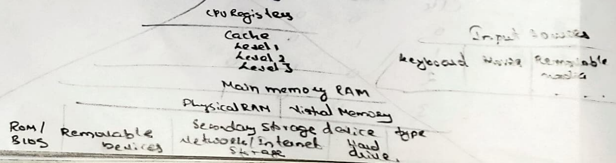
1. Input - Accept data or instruction as input. Eg: keyboard, Mouse
2. Output - This gives result in the form of output. Eg: Monitor, Speaker.
3. Memory
4. Data path
5. Control.

- Memory

=> The storage area in which programs are kept when they are running and that contains the data needed by the running programs.

=> Volatile memory devices: DRAM, SRAM.

=> Permanent storage : Magnetic disk, Optical disk etc..



- Datapath - The component of the processor that performs arithmetic operations.
- Control - The component of the processor that commands the datapath, memory and I/O devices according to the instruction of the program.

★ Performance of a Computer

2/11/2023
Thursday

- Response time : Also called execution time.
  The time between the start and the completion of a task.
- Throughput : Another measure of performance, Bandwidth
  It is the number of tasks completed per unit time.

★ Do the following changes to a computer system increase throughput, decrease response time, or both?

1. Replacing the processor in a computer with a faster version :
   - Throughput increases.
   - Response time decreases.
2. Adding additional processor to a system that uses multiple processor for separate tasks - For example searching the web.
   Throughput increases,
   Response time remains the same.

=> Decreasing response time (time taken for executing a task) almost always improves throughput.

* Performance

To maximize performance, we want to minimize response time / execution time.

Performance is inversely proportional to execution time

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

- This means, for 2 computers X and Y. If the performance of X is greater than of Y, then the execution time on Y is longer than X. ie; X is faster than Y.

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

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

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

"X is n times faster than Y" -

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

* Measuring performance.

- Program execution time is measured in seconds per Program.
- CPU execution time | Simply CPU time, the time the CPU spends to compute for this task and does not include time spent to waiting for I/O or running other programs.

=> The time CPU spends for computing a task (without including the waiting time for I/O).

* CPU time.

Two types:

- Uses CPU time – The CPU time spent in a program itself.
- System CPU time - The CPU time spent in the OS Performing tasks on behalf of the program

=> CPU performance and its Factors.

$$\boxed{\text{CPU execution time for a program} = \text{CPU clock cycle for a program} \times \text{clock cycle time.}}$$

=> Clock rate and clock cycle time are inverse,

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

* **Instruction performance.**

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

response time, elapsed time – total time to complete task, including disk access, memory access, I/O activities, OS overhead.

- clock cycle per instruction (CPI) – Average number of clock cycles each instruction takes to execute.

(Since different instruction may take different amounts of time, CPI is an average of all the instruction executed).

* **The classic CPU performance Equation.**

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

=> Since clock rate is inverse of clock cycle time:

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

Q. 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.

	A	B	C
CPI	1	2	3

CPI for each instruction class.

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

Code sequence	A	B	C
1	2	1	2
2	4	1	1

which code sequence executes the most instruction? which will be faster, CPI for each sequence.

ans): Sequence 1 executes 2+1+2 = 5 instruction

sequence 2 executes 4+1+1 = 6 instruction.

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

$$\text{CPU clock cycles}_1 = (2\times1)+(1\times2)+(2\times3) = 2+2+6 = \underline{\underline{10 \text{ cycles}}}$$

$$\text{CPU clock cycles}_2 = (4\times1)+(1\times2)+(1\times3) = 4+2+3 = 9 \text{ cycles.}$$

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

$$\text{CPI}_2 = \frac{9}{6} = \underline{1.5}$$

4/11/2023 Saturday

1 Ist (essay Ques. Short Ques.)

★ Instruction set architecture. (ISA)

- (language (instruction)
  - to execute → compiled (Assembly level)

Assembler → Machine code

→ Machine dependent → Assembler

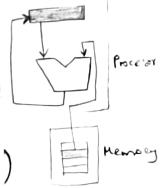
=> Interface between hardware & Software.

=> Load - copy ... operand from memory to register (Accumulator)

Very imp. ★ Classification : (based on the internal storage).

1. Single accumulator Organization (figure, explanation, eg: A+B Load (Accu) Store → to store in memory)
   - Load A
   - Add B
   - Store C.
2. General register organization.
   - Register - register (load Store). ← 2 operands are in registers. (No memory fetch)
     - Load R_1, A
     - Load R_2, B
     - Add R_3, R_1, R_2 (add → Store)
     - Store R_3, C (Stored to memory)
   - Register - memory → One operand in register one in memory
     - Load R_1 A (Loaded operand from memory to register) (register operand)
     - Add R_3, R_1, B (R_3 (stores)) (B (memory) operand) (No need to load operand as, it is Register - memory)
   - Memory - memory.
   - Accumulator - General purpose register.
3. Stack organization.

1) Single accumulator Organization



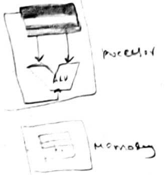
- one operand from accumulator
- one operand from memory.

Load A (Load data from memory)

Add B (add data from accumulator & data from memory) and stored in Register

Store C (The result is stored from accumulator to memory)

2) General register organization



=> Register - Register organization.

- two operands are in register

Load R_1, A (data/operand A is loaded from memory to Register R_1).

Load R_2, B

Add R_3, R_1, R_2 (Values in R_1 and R_2 is added and Stored in Register R_3).

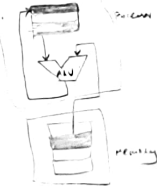
Store R_3, C. (R_3 value is stored in memory)

=> Register - Memory.

Load R_1, A (one operand from register, one from memory)

Add R_3, R_1, B

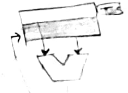
Store R_3, C (value/result from R_3 (register) is stored to memory C)



=> Memory - memory.

2 operands are taken from memory.

3) Stack organization



2 operands are in stack

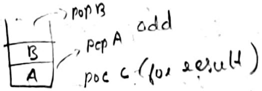
TOS (Top of stack)

Push A

Push B

Add

Pop C.



6/11/2023
Monday

* Different features that need to be Considered: (ISA)

i) Types of instructions.

- Data transfer (load, store)
- (arithmetic operation) ← Data manipulation / addition & subtraction, mul, div, incre, decre) XOR, OR, AND (bitwise operations) clear, carry rotate, shift)
- Jump instruction ← Program sequence & Control instructions.
- Input and output. Instructions.

Memory → Register (processor)

Register → memory (store)

⇒ clear R_0 mean $R_0 = 0$

⇒ interrupt - transfer of program Control from a currently running program

- Maskable interrupt
- Non maskable interrupt.

ii) Type and size of operands.

character (8 bits)
Half word (16 bits)
word (32 bits)
single precision floating point
double precision floating point.

(R_1) ⇒ 1004 ⇒ Content in 1004 memory location.

iii) Addressing mode. (where the operands are taken for calculation).

Imp Ques 6 mark.

1) Register - (operands are in Registers)

Add R_4, R_3, Regs $(R_4) \leftarrow$ Regs $(R_4]$ + Regs (R_3)

2) immediate addressing mode

Add R_4, #3

Regs $[R_4] \leftarrow$ Regs $[4]$ + 3.

3) Displacement.

Add R_4, $100(R_1)$ → fetch content from R_1

R_4 5 R_1 904

$R_4 = 5+7=12$

$$\begin{array}{r} 904+ \\ 100 \\ \hline 1004 \end{array}$$ (memory location)

Regs$[R_4] \leftarrow$ Regs$[R_4]$ + Mem$[100 +$ Regs$[R_1]]$

4) Register indirect.

R_4 5 R_1 1000

Add $R_4, (R_1)$

1000 fetch data from memory

Regs$[R_4] \leftarrow$ Regs$[R_4]$ + memory$[$Regs$[R_1]$ + memory$]$

5) Indexed Add R_3, $(R_1 + R_2)$

Regs$[R_3] \leftarrow$ Regs$[R_3]$ + Mem$[$Regs$[R_1]$ + Regs$[R_2]]$

6) Direct or absolute.

Add R_1, (1001) → memory

Add R_1 value to value of memory (1000)

Regs1 ← Regs R_1 + mem [1001]

7) Memory indirect.

Add R_1, @(R_3) , Regs$[R_1] \leftarrow$ Regs$[R_1]$ + Mem[mem[Regs R3]]

8) Autoincrement.

Add R_1, R_2+
R_2 value is incremented

Regs$[R_1] \leftarrow$ Regs$[R_1]$ + Mem[Regs$[R_2]]$
Regs$[R_2] \leftarrow$ Regs[[illegible]
+d

9) Autodecrement.

Add R_1-(R_2)

, Regs$[R_2] \leftarrow$ Regs$[R_2]$ -
Regs$[R_1] \leftarrow$ Regs$[R_1]$ +
Mem [Regs $[R_2]$]

10) Scaled

Add R_1, 100 $(R_2)[R_3]$

Regs $[R_1] \leftarrow$ Regs$[R_1]$ + Mem [100 + Regs[[illegible]] + Regs$[R_3]$ * d]

8/11/2023
Wednesday

Q. If computer A runs a program in 10 Seconds and computer B runs the same program in 15 second. How much faster is A than B.

ans). $\frac{\text{Execution time of B}}{\text{Execution time of A}} = \frac{15}{10} = \underline{\underline{1.5}}$

=> Computer A is 1.5 time faster than Computer B.

Q. Computer C's performance is 4 times as fastest the performance of computer B, which runs a given application in 28 Seconds. How long will computer C take to run that application.

ans): $\frac{\text{Execution time of B}}{\text{Execution time of C}} = 4$

$\frac{28}{C} = 4$ $C = 28/4 = \underline{\underline{7 \text{ Seconds}}}.$

Q. Our favourite programs runs in 10 seconds on Computer A which has a 2 Gigahertz clock we are trying to help a computer designer build a computer B, which will run this program in ~~10~~ 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 be tell the designer to target.

ans): $\text{CPU time} = \frac{\text{CPU clock cycles}}{\text{clock rate.}}$

2 Giga hertz
$= 2 \times 10^9$ hertz

$\text{clock rate A} = 2 \times 10^9 \text{ Hz}.$

$\text{CPU time}_A = 10 \text{ seconds}.$

$\text{CPU time} = \frac{\text{CPU clock cycles}}{\text{clock rate}}$, $\text{CPU clock cycle}_A = 10 \times 2 \times 10^9$
$= 20 \times 10^9$

$\text{CPU clock cycle}_B = 1.2 \times \text{clock cycles}_A$
$= 1.2 \times 20 \times 10^9 = \underline{\underline{24 \times 10^9}}$

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

$\text{clock rate}_B = \frac{24 \times 10^9}{6} = 4 \times 10^9 \text{ Hz}$
$= \underline{\underline{4 \text{ GHz}}}$

9/11/2023
Thursday

Q. Suppose we have two implementations of the same instruction set architecture, computer A has a clock cycle time of 250 PS and CPI of 2.0 for ~~Sub~~ Some programs and computer B has a clock cycle time of 500 PS (picosec) and CPI of 1.2 for the same program, Which computer is faster for the program and by how much.

$\text{No. of clock cycles} = \text{CPI} \times \text{Total no. of instructions}$

$= \text{CPI} \times I$

$\text{CPU clock cycle}_A = 2.0 \times I$

CPU clock cycle B = 1.2 * I.

$$\text{CPU time} = \frac{\text{CPU clock cycle A}}{\text{clock rate A.}}$$ [Clock rate not given]

CPU time$_A$ = CPU clock cycle A × clock cycle time A

= 2 * I & 250 PS

= <u>500 IPS</u>

CPU time$_B$ = CPU clock cycle$_B$ × Clock cycle time$_B$

= 1.2 * I × 500

=

= <u>600 IPS</u>

A is faster by $\cancel{\frac{500}{600}}$ $\frac{600}{500} = \frac{6}{5} = \underline{1.2 IPS}$

that computer B.

★ Addressing Memory

- Little Endian "Ssexdda" (address) MSB LSB
- Big Endian

imp short ans Ques 3 marks

=> Aligned

=> Missaligned

★ Module - 4

10/11/2023 Friday

* A basic MIPS Implementation

- Memory reference instruction.
- Arithmetic - logic instruction/R type
- Branch equal and Jump (without condition)
  Control instruction.

only if true. (condition based jump operation)

Load - Memory to register

Store - register to memory

R-type => fetch instruction from, memory

take address from program counter

R$_1$, R$_2$, R$_3$ | R$_1$ ← R$_2$ + R$_3$

(write) (Read)

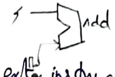
Read R$_2$

Read R$_3$

write R$_1$

write Data

★ Data Path



* State element -> Memory / registers.

i) PC address incremented by 4 (to move to next instruction). (First Add operation).

ii) Branch instruction (Second add operation)

- Combinational element - Operational element eg: ALU or gate
- State element - Memory element.
- edge triggered clocking - A clocking scheme (positive, negative)
- Control signal - selecting multiplexer, to control all operation
- Asserted - logically high or true
- Deasserted - logically low or false

[2 state elements and 1 adder]

Eg: 5 bit

blue line - control signals

=> ALU control signals. (blue line)

0000 AND

0001 OR

⋮

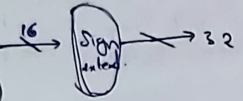
[based on the control signal ALU decides which operation is to be performed]

- branch target address – if

address + offset → PC. (updated address of PC).

⇒ Convert 16 bit to 32 bit.

(R type + Memory reference)



/ classroom Textbook 2 – first The processor chapter-4 Chapter-3 performance of computer

[Online class]

16/11/2023 Thursday

* ALU control lines (blue lines). (Arithmetic & Logic operations)

0000	AND
0001	OR
0010	add
0110	subtract

⇒ page 262

* R-type instruction

source register (write register) destination register. shift operation

32 bit *

0	rs	rt	rd	shamt	func
31:26	25:21 1st operand	20:16 2nd operand	15:11 5bits	10:6 10←6	5:0 ←5←0 [what operation to be performed]

⇒ 32 - general purpose register // 5 bits for 1 register

Load data stored in rt from memory.

32 bit *

Load-35 Store-43

(opcode)			
35 or 43	rs	rt	address
load / store 31:26	25:21	20:16 (destination register)	15:0

→ Load / Store instruction

→ Branch instruction

32 bit. *

4	rs	rt	address
31:26	25:21	20:16	15:0

compare value of rs & rt (AND operation]

if ALU = 0 (difference) equal

→ with explanation.

MIPS Datapath of < R-type, Load/Store, Branch

* Jump instruction

32 bit

000010	address.
31:26	25:0

// Jump Instruction

- upper 4 bits of the current PC+4
- 26 - bit immediate 25:0
- the bits 00

26+4+2 = 32

Jump = 2.

shift left 2 → 00

- FIGURES -

⇒ 4.3 Building a Datapath. Figure 4.9

⇒ 256 page branch control logic how to calculate branch.

⇒ 259 page Datapath for R-type & memory instruction ⇒ 262 page.

⇒ Overall structure. Figure 4.10 page 265

* Notes:

A Basic MIPS Implementation → [Million Instruction Per Second]

Core MIPS instruction set:

- Load – Load data from memory to register.
- Store – register to memory

i) The memory-reference instruction load word (lw) and store word (sw):

ii) The arithmetic-logical instruction add, sub, AND, OR and slt. [R-type instruction]

iii) The instruction branch equal (beq) and jump (j), which we add last. [beq]

condition equal

↳ Branch instruction < condition based jump [if true, jump to that part] / without any condition [jump]

- Fetch instruction from memory.

PC [register] – Address of next instruction to be executed (address) so from PC fetch the instruction in that particular memory

1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.

Fetch the instruction [from memory, by using the memory address from PC]

2. Read one or two registers using fields of the instruction to select the registers to read.

For the load word (lw) instruction, we need to read only one register (memory - register), but most other instruction require reading two registers.

Eg. If instruction is to perform Arithmetic operation. Both operands may be in register, 1 in register & 1 in memory

=> These 2 steps are identical for all instructions.

Other steps are based on the instruction class. [ALU used for all instruction]

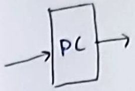
- For arithmetic operation - ALU required.
- Memory reference - ALU required because address calculation is needed.
- Branch instruction - ALU required => Jumpequal for comparison operation.

After using the ALU, the actions required to complete various instruction differs.

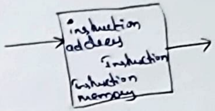
- A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
- For ALU instruction, the result based on the operation Performed (add, multiplication etc...) output from ALU ⟶ Store to register/ Store to memory } write.
- For Branch instruction ⟶ Decision for next instruction result of Comparison ⟶ if jump → address will be changed.

1 word = 32 bit
4 byte
(PC+4) => to move for next instruction

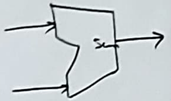
moves to next address (based on the word size) ⟵ if not jump ↵



- Program Counter - holds address of the next instruction to be executed. From that address, instruction is fetched.

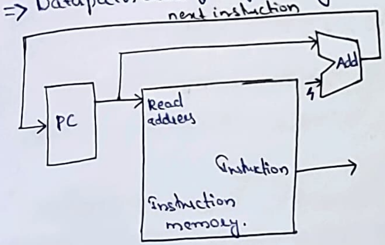


- Instruction memory - Memory where the instruction is stored (saved)



- Adder - used for addition for address calculation etc...

=> Datapath used for fetching instructions and incrementing the PC.

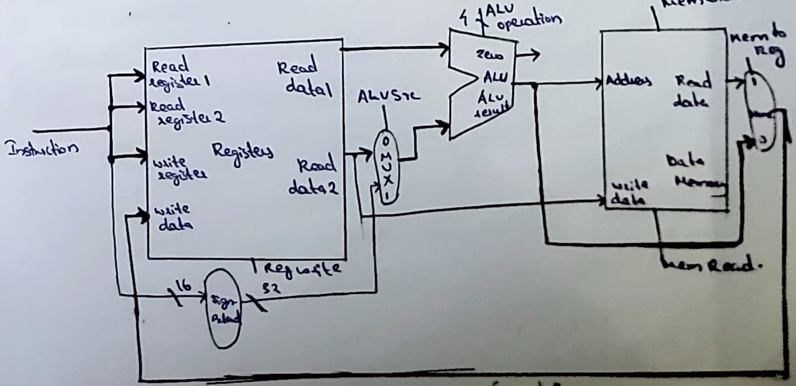


=> From PC the address of next instruction will be taken. from that address the instruction is taken.

=> That instruction is stored in the instruction memory.

=> For next instruction PC incremented as → the current value of PC+4

=> Datapath for the memory instruction (Load/store.) and the R-type instructions (Register based ALU operation)



R-type => add $R_1, $R_2, $R_3 {Read R2 Read R3 write R1 write data

stored in R1 ↖ R2 & R3 (added)

[illegible]