von-Neumann architecture:

Data and instructions are both stored in the same memory

Link

Inside the Processor:

Also called Central Processing Unit (CPU),
The CPU is divided into two parts: **ALU** and the **Control Unit**, which also has registers.

ALU –

1. All calculations happen inside ALU
2. Contains several registers (***general-purpose*** and ***special-purpose***)
3. Contains ***circuits*** to carry out *logic* operations like AND, OR, NOT, shift, compare, etc.
4. Contains ***circuits*** to carry out *arithmetic* operations like addition, subtraction, multiplication, division, etc.

Control Unit –

1. The control unit generates a sequence of control signals to carry out all operations
    a. Timing Signals
    b. Control Signals
2. When an instruction is fetched from memory, the operation (called opcode) is decoded by the control unit, and the control signals are issued.

Inside the memory Unit:

There are two types of memory subsystems:

Primary or Main memory –

It stores the active instructions and data for the program being executed on the processor.

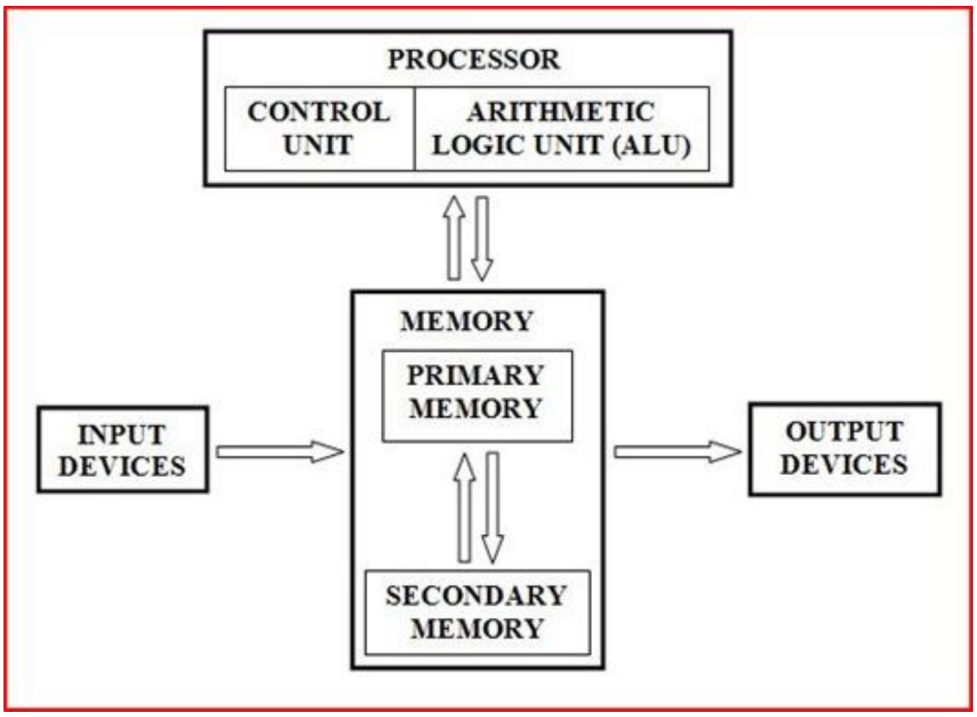
Secondary memory –

It is used as a backup and stores all active and inactive programs and data, typically as files.

The processor only has direct access to the primary memory.

In reality, the memory system is implemented as a hierarchy of several levels.

1. L1 cache, L2 cache, L3 cache, primary memory, and secondary memory.
2. The objective is to provide faster memory access at an affordable cost.



Interfacing with the Primary Memory:

Memory be 4096 x 16

2^12 locations
Each location contains one word, which is equal to 2 bytes which is 16 bits.
So total bits 2^12 * 16

Special Purpose Registers:

Memory Address Register (MAR): Holds the address of the memory location to be accessed. If it has n bits, then we can read from 2^n locations.
The MAR size needs to be 12 bits.

Memory Data Register (MDR): Holds the data that is being written into memory or will receive the data being read out from memory.
The MDR size needs to be 16 bits.

Accumulator:
Store the data fetched from memory.
The size needs to be 16 bits.

Program Counter (PC): Holds the memory address of the next instruction to be executed.

Automatically incremented to point to the next instruction when an instruction is being executed.

The size needs to be 12 bits

Instruction Register (IR): Temporarily holds an instruction that has been fetched from memory.

The first bit represents direct or indirect addressing. Followed by the opcode and operand.

The size needs to be 16 bits.

Types of Buses:

Address Bus:

Carries memory addresses from the processor to other components such as primary storage and input/output devices.
The address bit shows the number of slots in the memory
Uni-directional
The microprocessor generates the address.

Data Bus:

Carries the data between the processor and other components
Bi-directional

Control Bus:

Carries control signals from the processor to other components. The control bus also carries the clock's pulses
Uni-directional

- ***Bit****: A single binary digit (0 or 1).*
- ***Nibble****: A collection of 4 bits.*
- ***Byte****: A collection of 8 bits.*
- ***Word****: Does not have a unique definition.*
  • Varies from one computer to another; typically 32 or 64 bits.

- *Memory is byte or word organized.*

Unit	Bytes	In Decimal
8 bits (B)	1 or 2^0	10^0
Kilobyte (KB)	1024 or 2^{10}	10^3
Megabyte (MB)	1,048,576 or 2^{20}	10^6
Gigabyte (GB)	1,073,741,824 or 2^{30}	10^9
Terabyte (TB)	1,099,511,627,776 or 2^{40}	10^{12}
Petabyte (PB)	2^{50}	10^{15}
Exabyte (EB)	2^{60}	10^{18}
Zettabyte (ZB)	2^{70}	10^{21}

KMGTPEZ

Byte Ordering Conventions:

1. Little Endian:
   The least significant byte is stored at a lower address, followed by the most significant byte.
2. Big Endian:
   The most significant byte is stored at a lower address, followed by the least significant byte.

Von Neumann and Harvard Architecture:

Link

VON NEUMANN ARCHITECTURE	HARVARD ARCHITECTURE

It is ancient computer architecture based on stored program computer concept.	It is modern computer architecture based on Harvard Mark I relay based model.
Same physical memory address is used for instructions and data.	Separate physical memory address is used for instructions and data.
There is common bus for data and instruction transfer.	Separate buses are used for transferring data and instruction.
Two clock cycles are required to execute single instruction.	An instruction is executed in a single cycle.
It is cheaper in cost.	It is costly than Von Neumann Architecture.
CPU can not access instructions and read/write at the same time.	CPU can access instructions and read/write at the same time.

It is used in personal computers and small computers.	It is used in microcontrollers and signal processing.

Pipeline in Executing Instructions:1

Link

Design of a basic pipeline

- *In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages/segments such that the output of one stage is connected to the input of the next stage and each stage performs a specific operation.*
- *Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.*
- *All the stages in the pipeline along with the interface registers are controlled by a common clock.*

- Instruction Fetch (IF)
- Instruction Decode (ID)
- ALU operation (EX)
- Memory Access (MEM)
- Write Back result to register file (WB)

ETpipeline = k + n – 1 cycles = (k + n – 1) Tp

ETnon-pipeline = n * k * Tp

Instruction Set Architecture (ISA):

- Number of explicit operands:
    - 0, 1, 2, or 3.
- Location of the operands:
    - Registers, accumulators, and memory.
- Specification of operand locations:
    - Addressing modes: register, immediate, indirect, relative, etc.
- Sizes of operands supported:
    - Byte (8-bits), Half-word (16-bits), Word (32-bits), Double (64-bits), etc.
- Supported operations:
    - ADD, SUB, MUL, AND, OR, CMP, MOVE, JMP, etc.

Instructions:

Format:

Mode	Opcode	Operand

Single Accumulator Organization:
Only one operand.
Single address instructions only.

Accumulator
PC
IR

General Register Organization:
2 or 3 addresses

Register Stack Organization:
Zero address instruction.
Push, pop, and execute different instructions on the stack.

Instruction sets:

Stack Machine:
Push, pop and do different operations on stack

Accumulator based Machine:
All instructions assume that one of the operands (and also the result) is in a special register called accumulator

Register-Memory Machine:

One of the operands is assumed to be in register and another to be in memory

Register-Register Machine:
Also called **load-store** architecture, as only load and store instructions can access memory

Addressing Mode:
link

Implied Mode:
1. Operand is specified implicitly in the definition of instruction.
2. Used for zero address and one address instructions.

INCA → Increment Accumulator
For stack → Add

Immediate Mode:
1. Operand is directly provided as constant
2. No computation is required to calculate effective address
3. Fast but limited range (a limited number of bits are provided to specify the immediate data)

ADD R1,R2,42

Give direct data
<2^12
The maximum value is fixed

Register Mode:
1. Operand is present in the register
2. The register number is written in the instruction

Register Number is given in the instruction operand

The instruction size will be smaller
The address field will be 4 bits.
Register will give fast access

Register indirect mode:
1. Register contains the address of the operand rather than the operand itself.
2. Can access large address space
3. One fewer memory address as compared to indirect addressing.

Example: ADD R1,(R5)

Auto increment or auto decrement addressing mode:
A special case of register indirect addressing mode

Example: LD (R1)+

Direct Addressing Mode:

1. The actual address is given in instruction
2. Use to access variables
3. Single memory access is required to access the operand
4. Limited address space(limited by a number of 16 bits)

ADD R1,20A6H

Indirect Addressing Mode:

1. Used to implement pointers and pass parameters
2. 2 memory accesses are required
3. Slower but can access ample address space

ADD R1,(20A6H)

Relative Addressing: (PC Relative)

1. Effectively stored with respect to PC+1
2. Effective Address = PC (PC+1 because PC also increases) + offset
3. -2048 to +2047 if 12 bits offset

Base Register Addressing Mode:

1. Used in program relocation.
2. The processor has a special register called base register or segment register

Effective Address = Base Address + Displacement

Indexed Addressing Mode:

Either a special-purpose register or a general-purpose register is used as an index register in the addressing mode.
The instruction specifies an offset of displacement, which is added to the index register to get the effective address of the operand.

1. Use to access or implement arrays effectively.
2. Multiple registers are required to implement.
3. Any element can be accessed without changing instructions.

EA = Base Add + IR

Stack Addressing:

1. Operand is implicitly on top of the stack
2. Used in zero-address machines earlier
3. Many processors have a special register called the stack pointer (SP) that keeps track of the stack-top in memory.
4. PUSH, POP, CALL, RET instructions automatically modify SP

CISC vs RISC:

CISC	RISC (LOAD-STORE Architecture)
Complex Instruction Set Computer	Reduced Instruction Set Computer
A large number of instructions	Less number of instructions
Special-purpose registers and flags	More general purpose registers, and very few special purpose registers.
Variable length Instruction Format	Fixed Length Instruction Format
A large number of addressing modes	Few number of addressing modes
Cost high	Cost low
More powerful	Less powerful
Several Cycle Instructions	Single Cycle Instructions
Manipulation of data in memory	Only in registers
Microprogaram controlled unit	Hardware controlled unit
CISC based computers use hardware to translate into RISC instructions	RISC based computers use compilers to translate into RISC instructions

MIPS32 CPU:

The following CPU registers are visible to the machine/assembly language programmer:

1) 32, 32-bit general purpose registers (GPRs), R0 to R31
2) A special-purpose 32-bit program counter (PC)
    a) Points to next instruction in memory to be fetched and executed
    b) Not directly visible to the programmer
    c) Affected only indirectly by certain instructions (like branch,call,etc.)
3) A pair of 32-bit special purpose registers HI and LO, which are used to hold the results of multiply, divide and multiply-accumulate instructions.

Missing registers:

1) Stack Pointer
2) Index Register
3) Flag registers

Two of the GPRs have assigned functions:

1) R0 is hard-wired to a value of zero

a) It can be used as the target register for any instruction whose result is to be discarded.
b) It can also be used when a zero value is needed

2) R31 is used to store the return address when a function call is made
   a) Used by jump-and-link and branch-and-link like JAL
   b) It can also be used as a normal register.

Register name	Register number	Usage
$zero	R0	Constant zero
$at	R1	Reserved for assembler
$v0,$v1	R2, R3	Result of function, or for expression evaluation
$a0, $a1, $a2, $a3	R4, R5, R6, and R7	Argument 1,2,3,4 for functions
$t0 to $t9	R8 to R15 and R24, R25	Temporary (not preserved across calls)
$s0 to $s7	R16 to R23	Temporary (preserved across calls)
$gp	R28	Pointer to global area
$sp	R29	Stack Pointer
$fp	R30	Frame Pointer
$ra	R31	Return address (used by function call)
$k0, $k1	R26, R27	Reserved for OS kernel

Load and Store Instructions:

1. All operations are performed on operands held in processor registers
2. Main memory is accessed only through load and store instructions

Data sizes that can be accessed through LOAD and STORE

Data Size	Load Signed	Load Unsigned	Store
Byte	YES	YES	YES
Half-word	YES	YES	YES
Word	YES	Only for MIPS64	YES
Unaligned word	YES		YES
Linked word (atomic modify)	YES		YES

Type	Mnemonic	Function
Aligned	LB	Load Byte
	LBU	Load Byte Unsigned
	LH	Load Half-word
	LHU	Load Half-word Unsigned
	LW	Load Word
	SB	Store Byte
	SH	Store Half-word
	SW	Store Word

Type	Mnemonic	Function
Unaligned	LWL	Load Word Left
	LWR	Load Word Right
	SWL	Store Word Left
	SWR	Store Word Right
Atomic Update	LL	Load Linked Word
	SB	Store Conditional Word

LL, SC:

In the MIPS32 assembly language, "LL" and "SC" instructions are used in the context of implementing atomic operations for multi-threaded programming, particularly for synchronization in concurrent environments. They stand for "Load Linked" (LL) and "Store Conditional" (SC) respectively. Here's what each instruction means:

Load Linked (LL):

- The LL instruction loads a value from memory into a register while marking the memory location as "linked."
- The "linked" status indicates that you are starting an atomic sequence of instructions. If another thread modifies the memory location after the LL instruction but before the corresponding SC instruction (if any), the SC instruction will fail.
- The LL instruction helps implement a form of atomic read on the memory location.

Store Conditional (SC):

- The SC instruction stores a value from a register into a memory location only if the memory location is still "linked."

- If the memory location was modified by another thread after the LL instruction but before the SC instruction, the SC instruction will fail to store the value, and this failure can be detected.
- The SC instruction helps in implementing an atomic write to the memory location.

LWL, LWR:

In MIPS assembly language, LWL and LWR are instructions to load unaligned data from memory into registers. They are used when the loaded data is not aligned to a word boundary (i.e., not at an address that is a multiple of 4 bytes). These instructions are used to load data that spans multiple memory locations.

LWL (Load Word Left):

- The LWL instruction loads a 32-bit word into a register, performing a left-aligned load.
- It is used to load the leftmost (most significant) portion of the word from memory, and the rest of the word remains unchanged in the register.
- The loaded word occupies the leftmost bits of the destination register, and the rightmost bits are not modified.

LWR (Load Word Right):

- The LWR instruction loads a 32-bit word into a register, performing a right-aligned load.
- It is used to load the rightmost (least significant) portion of the word from memory, and the rest of the word remains unchanged in the register.
- The loaded word occupies the rightmost bits of the destination register, and the leftmost bits are not modified.

Arithmetic and Logic Instructions:

1. All arithmetic and logic instructions operate on registers.
2. Can be broadly classified into the following categories:
    a. ALU immediate
    b. ALU 3-operand
    c. ALU 2-operand
    d. Shift
    e. Multiply and Divide

Type	Mnemonic	Function
16-bit Immediate Operand	ADDI	Add Immediate Word
	ADDIU	Add Immediate Unsigned Word
	ANDI	AND Immediate
	LUI	Load Upper Immediate
	ORI	OR Immediate
	SLTI	Set on Less Than Immediate
	SLTIU	Set on Less Than Immediate Unsigned
	XORI	Exclusive-OR Immediate

Type	Mnemonic	Function
3-Operand	ADD	Add Word
	ADDU	Add Unsigned Word
	AND	Logical AND
	NOR	Logical NOR
	SLT	Set on Less Than
	SLTU	Set on Less Than Unsigned
	SUB	Subtract Word
	SUBU	Subtract Unsigned Word
	XOR	Logical XOR

Type	Mnemonic	Function
Shift	ROTR	Rotate Word Right
	ROTRV	Rotate Word Right Value (Register)
	SLL	Shift Word Left Logical
	SLLV	Shift Word Left Logical Value (Register)
	SRA	Shift Word Right Arithmetic
	SRAV	Shift Word Right Arithmetic Value (Register)
	SRL	Shift Word Right Logical
	SRLV	Shift Word Right Logical Value (Register)

SRA (Shift Right Arithmetic):

- SRA performs a right shift on the binary representation of the value stored in the source register.
- The leftmost vacant bit positions (those that are shifted out) are filled with the sign bit (the most significant bit) of the original value. This is done to maintain the sign of the value after the shift.
- It's useful when performing arithmetic operations on signed integers to preserve their signedness.

SRL (Shift Right Logical):

- SRL also performs a right shift on the binary representation of the value stored in the source register.
- The leftmost vacant bit positions are filled with zeros.
- It's commonly used for bit manipulation and extracting specific bits from a value, without consideration for signedness.

SRLV (Shift Right Logical Variable):

The value in a separate register determines the amount of the logical right shift that the SRLV instruction performs on a register's bits.

```
srlv $destination, $source, $shift_amount_register
```

Multiply and Divide Instructions:

1. For multiplication, the low half of the product is loaded into LO, while the higher half in HI.
2. Divide produces a quotient that is loaded into LO and a remainder that is loaded into HI

Type	Mnemonic	Function
Multiply and Divide	DIV	Divide Word
	DIVU	Divide Unsigned Word
	MADD	Multiply and Add Word
	MADDU	Multiply and Add Word Unsigned
	MFHI	Move from HI
	MFLO	Move from LO
	MSUB	Multiply and Subtract Word
	MSUBU	Multiply and Subtract Word Unsigned
	MTHI	Move to HI
	MTLO	Move to LO
	MUL	Multiply Word to Register
	MULT	Multiply Word
	MULTU	Multiply Unsigned Word

Jump and Branch Instructions:

1. Following types of jump and branch instructions are supported by MIPS32
    a. PC relative conditional branch
        i. 16-bit offset
    b. PC relative unconditional jump
        i. 26-bit offset
    c. Absolute unconditional jump
    d. Special jump instructions that link the return address in R31

Type	Mnemonic	Function
Unconditional Jump within a 256 MB Region	J	Jump
	JAL	Jump and Link
	JALX	Jump and Link Exchange

Type	Mnemonic	Function
Unconditional Jump using Absolute Address	JALR	Jump and Link Register
	JR	Jump Register

J (Jump):

The J instruction, also known as "Jump," is used to perform an unconditional jump to a target memory address. It's often used for implementing loops and branching to distant locations in the code.

```
j target_address
```

- target_address is the 26-bit absolute target memory address.

JAL (Jump and Link):

The JAL instruction, short for "Jump and Link," is used to perform a jump to a target address while saving the return address (address of the instruction following JAL) in the link register ($ra).

```
jal target_address
```

- target_address is the 26-bit absolute target memory address.

JALX (Jump and Link Exchange):

JALX is a variation of the JAL instruction. In MIPS32, it's used in a 64-bit context to jump and link to a target address while saving the return address in $ra.

JALR (Jump and Link Register):

The JALR instruction, "Jump and Link Register," is used to perform a jump to a target address specified by a register while saving the return address in the link register ($ra).

```
jalr $rd, $rs
```

- $rd is the destination register where the return address will be stored.
- $rs is the source register containing the target address.

JR (Jump Register):

The JR instruction, "Jump Register," is used to perform an unconditional jump to a target address specified by a register.

```
jr $rs
```

- $rs is the source register containing the target address.

Type	Mnemonic	Function
PC-Relative Conditional Branch Comparing Two Registers	BEQ	Branch on Equal
	BNE	Branch on Not Equal

Type	Mnemonic	Function
PC-Relative Conditional Branch Comparing With Zero	BGEZ	Branch on Greater Than or Equal to Zero
	BGEZAL	Branch on Greater Than or Equal to Zero and Link
	BGTZ	Branch on Greater than Zero
	BLEZ	Branch on Less Than or Equal to Zero

The MIPS architecture defines four coprocessors

1. Coprocessor 0 (CP0) is incorporated on the CPU chip and supports the virtual memory system and exception handling. CP0 is also referred to as the System Control Coprocessor.
2. Coprocessor 1 (CP1) is reserved for the floating point coprocessor.
3. Coprocessor 2 (CP2) is available for specific implementations.
4. Coprocessor 3 (CP3) is available for future extensions.

Pseudo Instructions:

```
blt $s1,$s2,Label
```

```
slt $at,$s1,$s2
bne $at,$zero,Label
```

Pseudo-Instruction	Translates to	Function
blt $1, $2, Label	slt $at, $1, $2 bne $at, $zero, Label	Branch if less than
bgt $1, $2, Label	sgt $at, $1, $2 bne $at, $zero, Label	Branch if greater than
ble $1, $2, Label	sle $at, $1, $2 bne $at, $zero, Label	Branch if less or equal
bge $1, $2, Label	sge $at, $1, $2 bne $at, $zero, Label	Branch if greater or equal
li $1, 0x23ABCD	lui $1, 0x0023 ori $1, $1, 0xABCD	Load immediate value into a register

Pseudo-Instruction	Translates to	Function
move $1, $2	add $1, $2, $zero	Move content of one register to another
la $a0, 0x2B09D5	lui $a0, 0x002B ori $a0, $a0, l0x09D5	Load address into a register

Working with value in registers:

1. Small constants, which can be specified in 16 bits can be used through immediate instructions
2. Large constants that require 32 bits
    a. Load through 2 instructions
        i. Load upper immediate
        ii. OR immediate
        iii. Example: load 0xAAAA3333 in register $s1
        lui $s1,0xAAAA
        ori $s1,$s1,0x3333

MIPS Instruction Encoding:

Link

R-type: register-register operations

1. Shift operations
2. Jump with register link
3. Move,store from and to hi and lo
4. Mult ,div…..
5. slt

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

Opcode = 000000

Function bits	**Binary representative**
ADD	100000
SUB	100010
AND	100100

OR	100101
SLT	101010

Tricky:

1. **break**
2. **jalr***
3. **jr***
4. **syscall**

For more, check here: Link

I-Type:

1. Branch
2. Load
3. Alu operations with immediate
4. slti

opcode (6)	rs (5)	rt (5)	Immediate/constant or offset (16)

Check here: Link

Offest is stored with respect to PC+4 in general.

Tricky:

1. **beq**
2. **bgez**
3. **bgtz**
4. **blez**
5. **bltz**
6. **bne**
7. **lb**
8. **lh**
9. **lui**

10. **lw**
11. **sb**
12. **sh**
13. **sw**

J-Type:

opcode (6)	Offset relative to PC (26)

Check Here: Link

Offest is multiplied by 4 before using. Basically, two 0's are padded on right and then made 28 bits.

Jump, Jump and Link, trap and return

Offest is stored with respect to PC+4 in general.

Addressing Modes in MIPS32

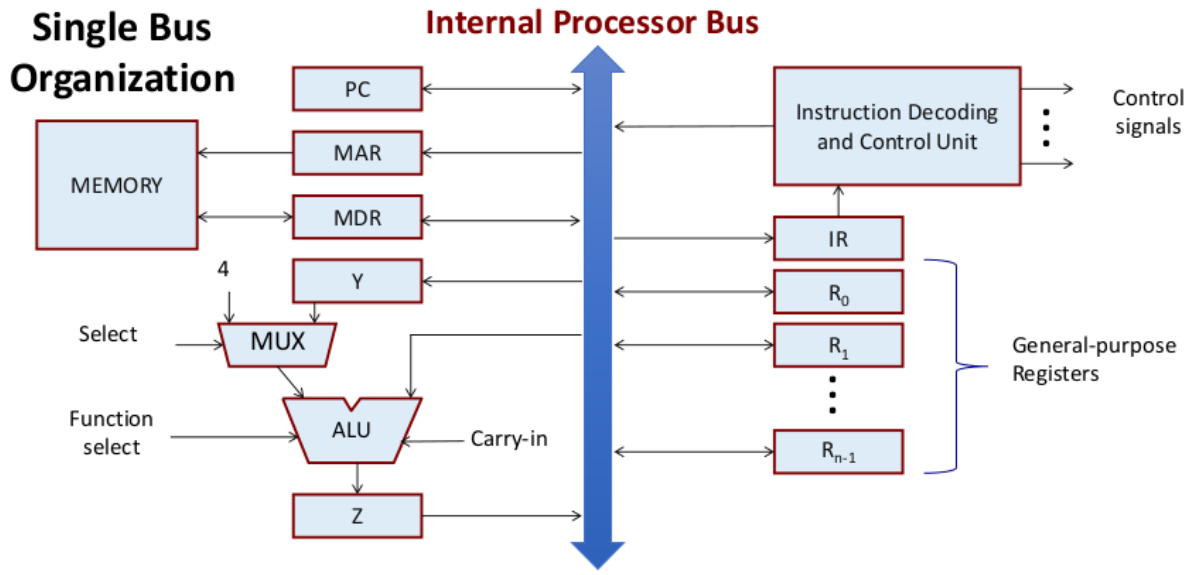
- Register addressing — *add $s1, $s2, $s3*
- Immediate addressing — *addi $s1, $s2, 200*
- Base addressing — *lw $s1, 150($s2)*
  - Content of a register is added to a "base" value to get the operand address.
- PC relative addressing — *beq $s1, $s2, Label*
  - 16-bit offset is added to PC to get the target address.
- Pseudo-direct addressing — *j Label*
  - 26-bit offset if shifted left by 2 bits and then added to PC to get the target address.

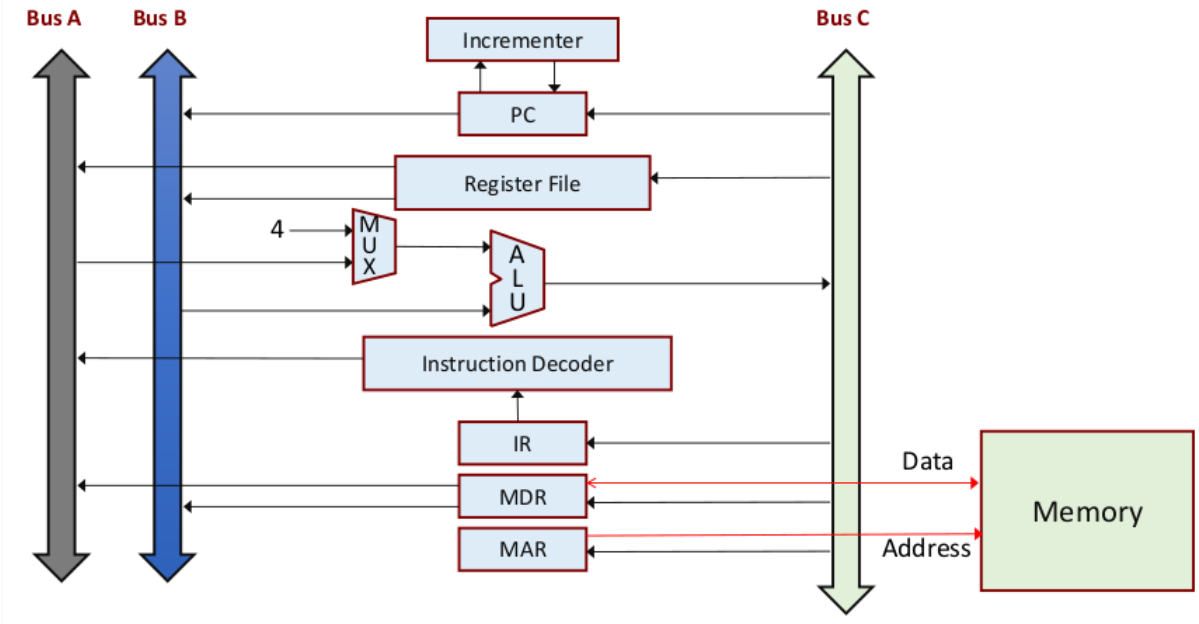
Amdahl's Law

It basically states that the performance improvement to be gained from using

some faster mode of execution is limited by the fraction of the time the faster mode can be used.

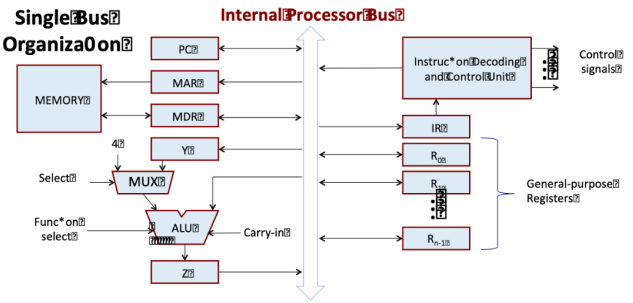
Design of Control Unit





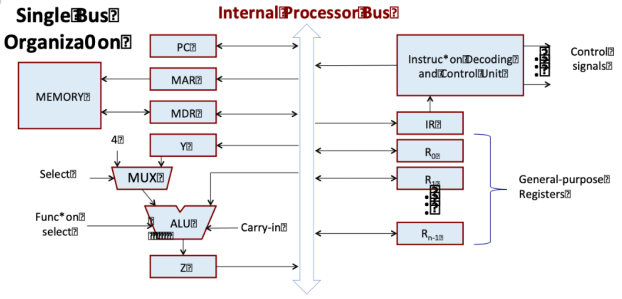
1. ADD R1, R2 (R1 = R1+R2)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Add, Z_{in}
6	Z_{out}, $R1_{in}$, End



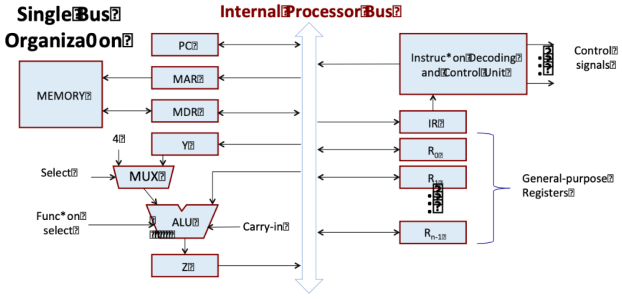
2. ADD R1, LOCA (R1 = R1 + Mem[LOCA])

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



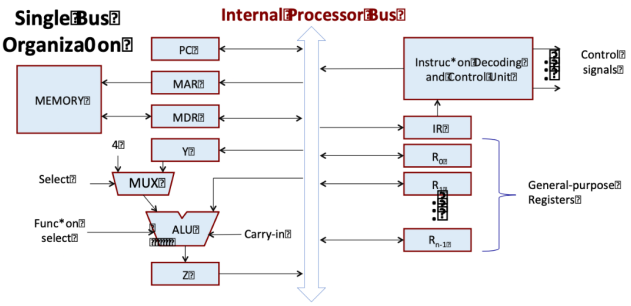
3. LOAD R1, LOCA (R1 = Mem[LOCA])

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	WMFC
6	MDR_{out}, $R1_{in}$, END



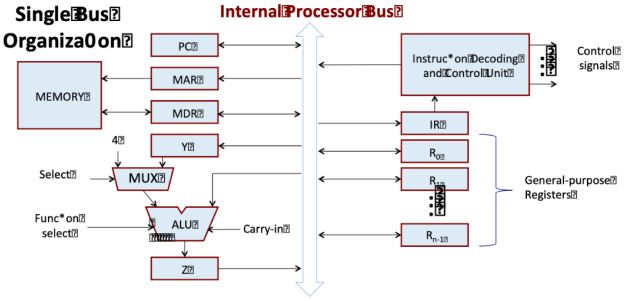
4. STORE LOCA, R1 (Mem[LOCA] = R1)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}
5	$R1_{out}$, MDR_{in}, Write
6	MDR_{outE}, WMFC, End



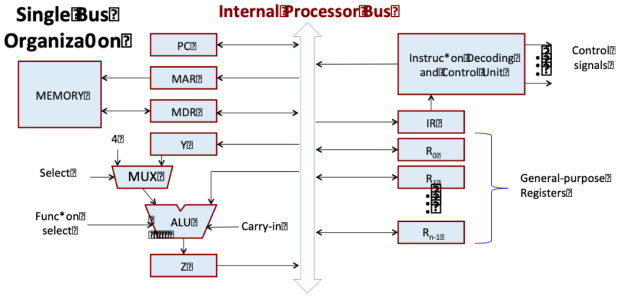
5. MOVE R1, R2 (R1 = R2)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R2_{out}$, $R1_{in}$, END



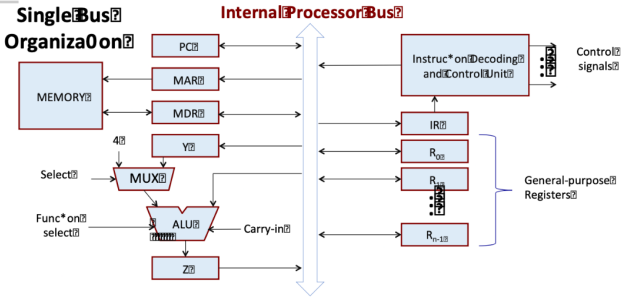
6. MOVE R1, #10 (R1 = 10)

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Immediate field of IR_{out}, $R1_{in}$, END



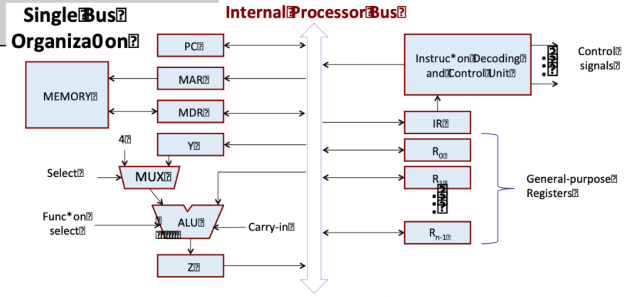
7. BRANCH Label (PC = PC + offset)

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
5	Z_{out}, PC_{in}, End



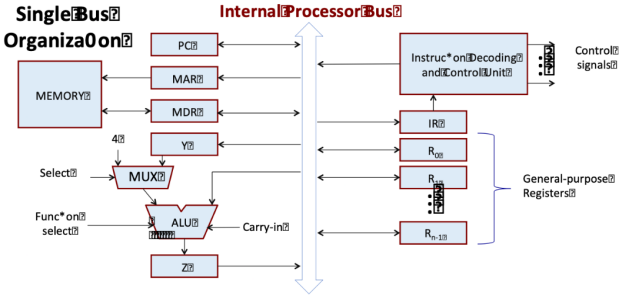
8. BNZ Label (if Z=1 PC = PC + offset)

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}, If Z=0 then End
5	Z_{out}, PC_{in}, End



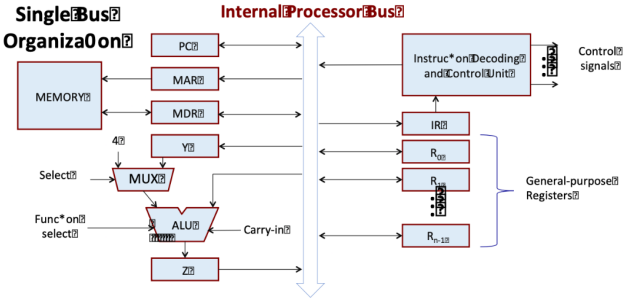
9. INC R1 (R1 = R1 + 4)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Select4, Add, Z_{in}
5	Z_{out}, $R1_{in}$, End



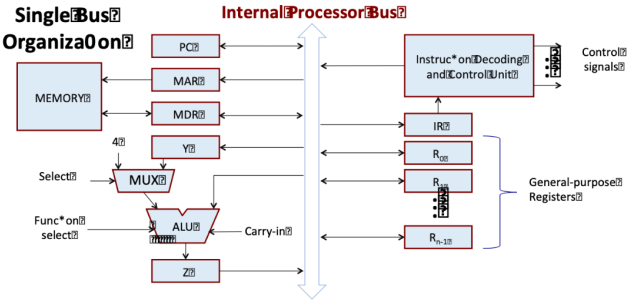
10. DEC R1 (R1 = R1 – 4)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Select4, SUB, Z_{in}
5	Z_{out}, $R1_{in}$, End



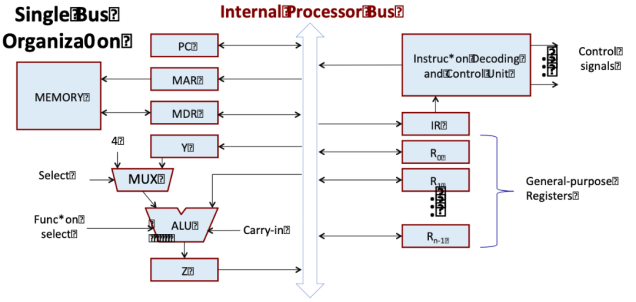
11. CMP R1, R2

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Sub, Z_{in}, End



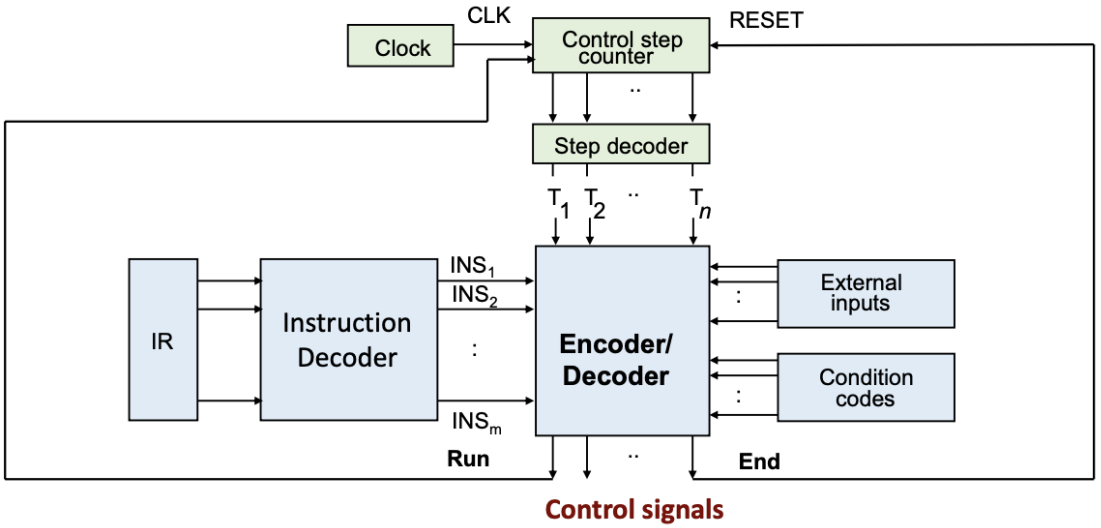
12. HALT

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	End



Hardwired and Microprogrammed Control Unit Design

Hardwired Control unit



The encoder/decoder circuit is a combinational circuit that generates control signals depending on the inputs provided.

The step decoder generates a separate signal line for each step in the control sequence (T1, T2, T3, etc.).

Depending on the maximum number of steps required for an instruction, the step decoder is designed.

• If a maximum of 10 steps are required, then a 4 x 16-step decoder is used.

Among the total set of instructions, the instruction decoder is used to select one of them. (That particular line will be 1, and the rest will be 0).

If a maximum of 100 instructions are present in the ISA then a 7 x 128 instruction decoder is used.

• At every clock cycle the RUN signal is used to increment the counter by one.
• When RUN is 0 the counter stops counting.
• This signal is needed when WMFC is issued.

• END signal starts a new instruction.
• It resets the control step counter to its starting value.

• The sequence of operations carried out by the control unit is determined by the wiring of the logic elements and hence it is named hardwired.

• This approach of control unit design is fast but limited to the complexity of instruction set that is implemented.

Generation of Control Signals

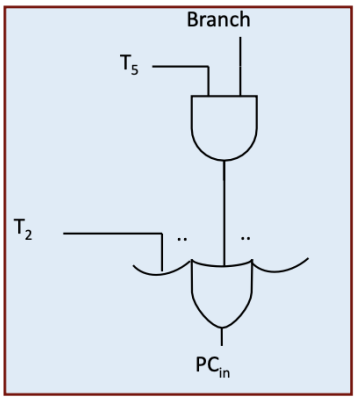
	ADD R1, R2
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Add, Z_{in}
6	Z_{out}, $R1_{in}$, End

	ADD R1, LOCA
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

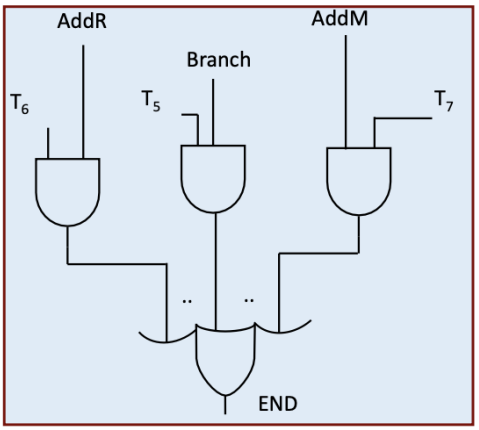
	BRANCH Label
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Generation of PC_{in} and END

$PC_{in} = T_2 + T_5.Branch$



$END = T_6.ADDR + T_5.Branch + T_7.AddM$



Microprogrammed Control Unit Design

1. Control signals are generated by a program similar to machine language program
2. A control store (CS) stores the microroutines for all instructions of an ISA
3. The sequence of steps corresponding to the control sequence of a machine instruction is the microroutine.
4. Each sequence of steps is a control word (CW) whose individual bits represent various control signals

5. Individual control words in a micro routine are called microinstructions
6. Control-unit generates the control signals for an instruction by sequentially reading CWs of the corresponding micro routine from CS
7. The μPC is used to read CWs sequentially from CS
8. Every time a new instruction is loaded into IR, the output of the starting address generator is loaded into μPC
9. Then μPC is automatically incremented by clock, causing successive microinstructions to be read from the Control store

Microinstructions are low-level instructions that control the operations of a microprocessor or microcontroller at a very granular level. These microinstructions are executed sequentially to perform various tasks, such as data manipulation, memory access, and control flow. In some microprogramming architectures, especially those used in microcontrollers and microprocessors with complex instruction sets, controlling the flow of microinstructions can be challenging. To manage this, a concept called "microinstructions with a Next-Address Field" can be employed, which aims to simplify control flow.

Here's an explanation of this approach, its pros, and cons:

Microinstructions with Next-Address Field:

Traditional Microinstructions with Branch Instructions: In traditional microprogramming, controlling the flow of microinstructions often involves the use of specific branch microinstructions. These branch microinstructions are responsible for altering the program counter (PC) or determining the next microinstruction to be fetched. However, these branch instructions typically don't perform any useful operation in the datapath. They exist solely for controlling the program flow.

Alternative Approach with Next-Address Field: The alternative approach involves adding an address field as part of every microinstruction. This address field indicates the location or

address of the next microinstruction to be fetched and executed. Instead of relying on separate branch microinstructions, the microinstructions themselves carry the information about where to go next.

Pros of Microinstructions with Next-Address Field:

- Reduced Dependency on Branch Microinstructions: This approach significantly reduces the need for separate branch microinstructions. In traditional microprogramming, a substantial portion of microinstructions may be dedicated solely to branching, leading to inefficient use of microcode memory and execution time. With the next-address field, branching can be integrated into regular microinstructions.
- Flexible Control Flow: There are fewer limitations in assigning addresses to microinstructions. This flexibility allows for more intricate control flow logic to be encoded directly within the microinstructions, making it easier to implement complex sequences of instructions.

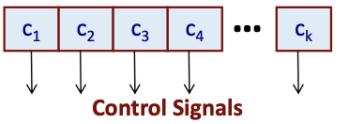
Cons of Microinstructions with Next-Address Field:

- Additional Bits Required: To incorporate the address field into every microinstruction, additional bits are needed. These bits consume extra microcode memory and may increase the complexity of the microinstruction format.
- Complexity and Cost: Implementing the hardware to manage the next-address field within microinstructions can add complexity to the microarchitecture and might increase the cost of the processor or microcontroller.

In summary, the use of microinstructions with a next-address field simplifies control flow in microprogramming by reducing the reliance on separate branch microinstructions. This approach offers greater flexibility and can lead to more efficient microcode, but it does come with the trade-off of requiring additional bits for the address field and potentially increasing hardware complexity. The choice between traditional branch microinstructions and microinstructions with next-address fields depends on the specific requirements of the microarchitecture and the trade-offs deemed acceptable for a given design.

Horizontal versus Vertical Microinstruction Encoding

1. Horizontal Micro-Instruction Encoding:



- Suppose that there are *k* control signals: $c_1, c_2, \ldots, c_k$.
- In horizontal encoding, every control word stored in control memory (CM) consists of *k* bits, one bit for every control signal.
- Several bits in a control word can be 1:
  - Parallel activation of several micro-operations in a single time step.

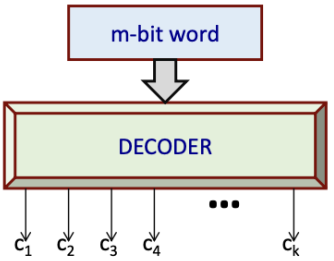
0	1	0	1	1	0

→ c_2, c_4 and c_5 are activated together

- Advantage:
  - Unlimited parallelism is possible in the activation of the micro-operations.
- Disadvantage:
  - Size of the control memory is large (word size is much longer).
  - Cost of implementation is higher.

2. Vertical Micro-Instruction Encoding:

- Again consider that there are k control signals: $c_1, c_2, \ldots, c_k$.
- We encode the control signals in an m-bit word in the control memory, where $k \leq 2^m$.
- Depending on the m-bit control word, exactly one control signal will be activated (= 1), while all others will remain de-activated (= 0).
  - At most one control signal can be activated in a time step.

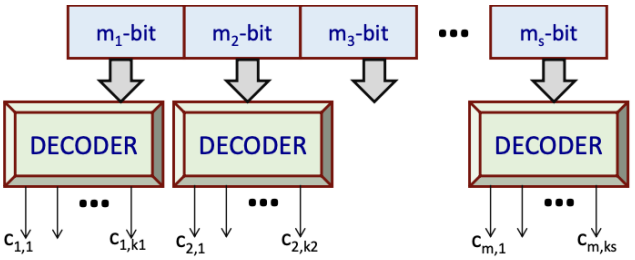


- Advantage:
  - Requires much smaller word size in control memory.
  - Low cost of implementation.
- Disadvantage:
  - More than one control signals cannot be activated at a time.
  - Requires sequential activation of the control signals, and hence more number of time steps.

S. No	Horizontal μ-programmed CU	Vertical μ-programmed CU
1.	It supports longer control word.	It supports shorter control word.
2.	It allows a higher degree of parallelism. If degree is n, then n Control Signals are enabled at a time.	It allows a low degree of parallelism i.e., the degree of parallelism is either 0 or 1.
3.	No additional hardware is required.	Additional hardware in the form of decoders is required to generate control signals.
4.	It is faster than a Vertical micro-programmed control unit.	it is slower than a Horizontal micro-programmed control unit.
5.	It is more flexible than a vertical micro-programmed control unit.	It is less flexible than horizontal but more flexible than that of a hardwired control unit.
6.	A horizontal micro-programmed control unit uses horizontal	A vertical micro-programmed control unit uses vertical micro-instruction, where a code is used for each action to be

	micro-instruction, where every bit in the control field attaches to a control line.	performed and the decoder translates this code into individual control signals.
7.	The horizontal micro-programmed control unit makes less use of ROM encoding than the vertical micro-programmed control unit.	The vertical micro-programmed control unit makes more use of ROM encoding to reduce the length of the control word.

3. Diagonal Micro-Instruction Encoding:



- Suppose we group the set of k control signals into s groups, containing $k_1, k_2, \ldots, k_s$ signals.
- We encode the control signals in groups as shown, where $k_i \leq 2^{mi}$.
  - Within a group, at most one control signal can be activated in a time step.
  - Parallelism across groups is allowed.

- Advantages:
  - Maximum parallelism as required by the micro-programs can be supported.
  - Word size of control memory is less than that for horizontal encoding.
  - Used in practice.
- Disadvantages:
  - Multiple decoders (though smaller in sizes) are required.

Control Unit Design for MIPS32

The instruction execution cycle is divided into 5 steps:

1. ***IF***: Instruction Fetch → **2**
2. ***ID***: Instruction Decode / Register Fetch → **4**
3. ***EX***: Execution / Effective Address Calculation → **4**
4. ***MEM***: Memory Access / Branch Completion → **4**
5. ***WB***: Register Write-back → **3**

(a) IF : Instruction Fetch

- Here the instruction pointed to by *PC* is fetched from memory, and also the next value of *PC* is computed.
  - Every MIPS32 instruction is of 32 bits (i.e. 4 bytes).
  - For a branch instruction, new value of the *PC* may be the target address. So *PC* is not updated in this stage; new value is stored in a register *NPC*.

IF:

```
IR  ← Mem [PC];
NPC ← PC + 4;
```

(b) ID : Instruction Decode

- The instruction already fetched in *IR* is decoded.
  - *Opcode* is 6-bits: bits 31..26, with optional *function specifier*: bits 5..0
  - First source operand *rs*: bits 25..21, second source operand *rt*: bits 20..16
  - 16-bit immediate data: bits 15..0
  - 26-bit immediate data: bits 25..0
- Decoding is done in parallel with reading the register operands *rs* and *rt*.
  - Possible because these fields are in a fixed location in the instruction format.
- In a similar way, the immediate data can be sign-extended.

ID:

```
A    ← Reg [rs];
B    ← Reg [rt];
Imm  ← (IR15)16 ## IR15..0          // sign extend 16-bit immediate field
Imm1 ← (IR26)6 ##IR25..0 ## 00      // pad 2 0's to 26-bit immediate field
```

A, B, Imm, Imm1 are temporary registers.

(c) EX: Execution / Effective Address Computation

- In this step, the ALU is used to perform some calculation.
  - The exact operation depends on the instruction that is already decoded.
  - The ALU operates on operands that have been already made ready in the previous cycle.
- We show the micro-instructions corresponding to the type of instruction.

Memory Reference:

```
ALUOut ← A + Imm;
```

Example: LW R3, 100(R8)

Register-Register ALU Instruction:

```
ALUOut ← A func B;
```

Example: SUB R2, R5, R12
[operation specified by func field (bits 5..0)]

Register-Immediate ALU Instruction:

```
ALUOut ← A func Imm;
```

Example: SUBI R2, R5, 524
[operation specified by func field (bits 5..0)]

Branch:

```
ALUOut ← NPC + Imm1;
cond   ← (A op 0);
```

Example: BEQZ R2, Label
[op is ==]

(d) MEM: Memory Access / Branch Completion

- The only instructions that make use of this step are loads, stores, and branches.
  - The load and store instructions access the memory.
  - The branch instruction updates *PC* depending upon the outcome of the branch condition.

Load instruction:

PC ← NPC;

LMD ← Mem [ALUOut];

Store instruction:

PC ← NPC;

Mem [ALUOut] ← B;

Other instructions:

PC ← NPC;

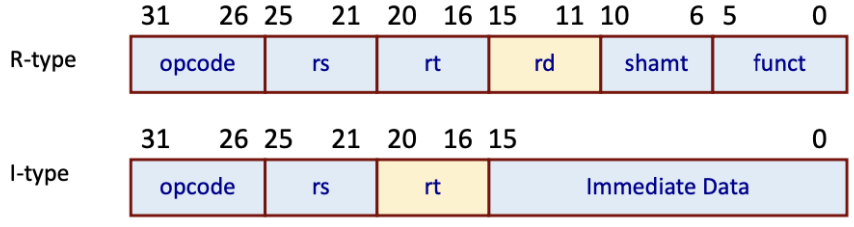
Branch instruction:

if (cond) PC ← ALUOut;

else PC ← NPC;

(e) WB: Register Write Back

- In this step, the result is written back into the register file.
  - Result may come from the ALU.
  - Result may come from the memory system (viz. a LOAD instruction).
- The position of the destination register in the instruction word depends on the instruction → *already known after decoding has been done*.



Register-Register ALU Instruction:

Reg [rd] ← ALUOut;

Register-Immediate ALU Instruction:

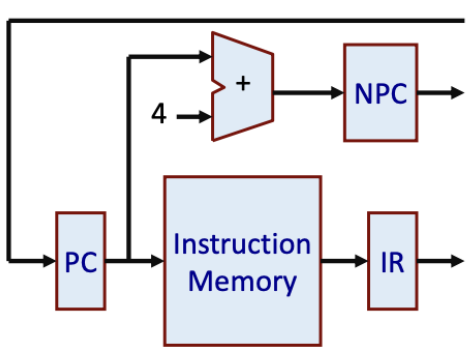
Reg [rt] ← ALUOut;

Load Instruction:

Reg [rt] ← LMD;

Data Path Design of MIPS32

The IF Stage

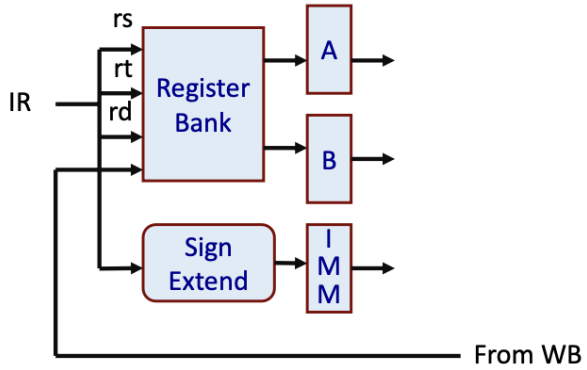


IR ← Mem [PC];

NPC ← PC + 4;

- 32-bit PC
- 32-bit NPC
- 32-bit IR
- 32-bit adder

The ID Stage



A ← Reg [rs];

B ← Reg [rt];

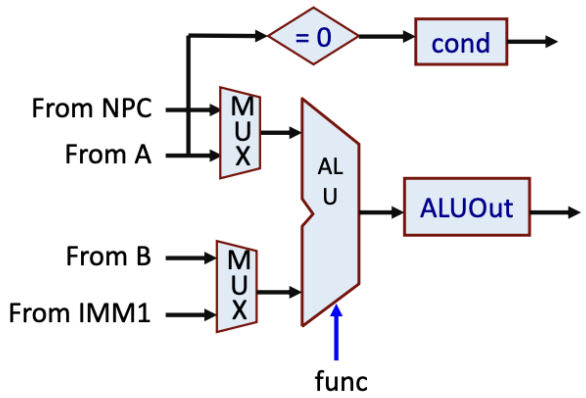
Imm ← $(IR_{15})^{16}$ ## $IR_{15..0}$

Imm1 ← $(IR_{25})^{6}$ ## $IR_{25..0}$ ## 00

The IMM1 calculation is not shown.

- 32-bit A
- 32-bit B
- 32-bit IMM

The EX Stage



- 32-bit ALUOut
- 1-bit cond
- 32-bit 2x1 MUX

Memory Reference:

ALUOut ← A + Imm;

Register-Register ALU Instruction:

ALUOut ← A func B;

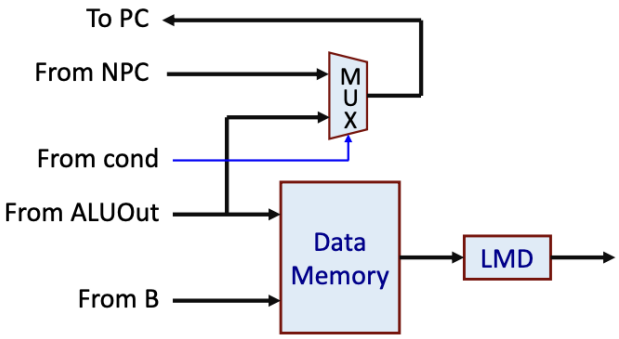
Register-Immediate ALU Instruction:

ALUOut ← A func Imm;

Branch:

ALUOut ← NPC + Imm1;
cond ← (A op 0);

The MEM Stage



- 32-bit LMD
- 32-bit MUX

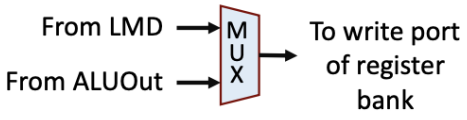
Load instruction:
PC ← NPC;
LMD ← Mem [ALUOut];

Store instruction:
PC ← NPC;
Mem [ALUOut] ← B;

Branch instruction:
if (cond) PC ← ALUOut;
else PC ← NPC;

Other instructions:
PC ← NPC;

The WB Stage

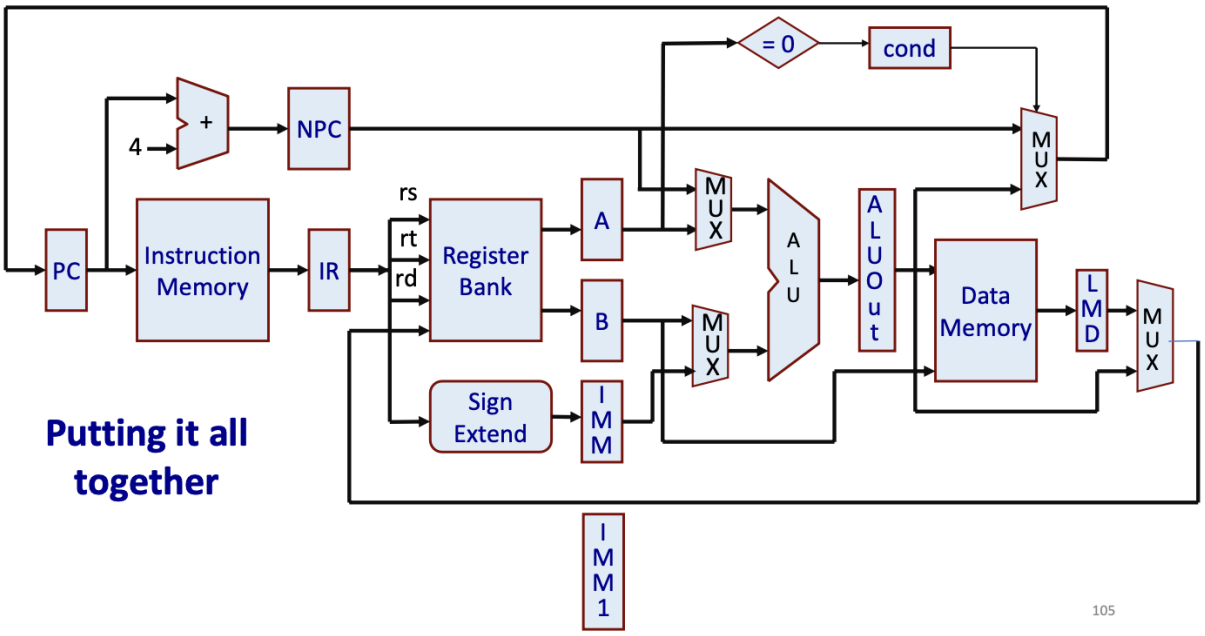


- 32-bit MUX

Register-Register ALU Instruction:
Reg [rd] ← ALUOut;

Register-Immediate ALU Instruction:
Reg [rt] ← ALUOut;

Load Instruction:
Reg [rt] ← LMD;



Putting it all together

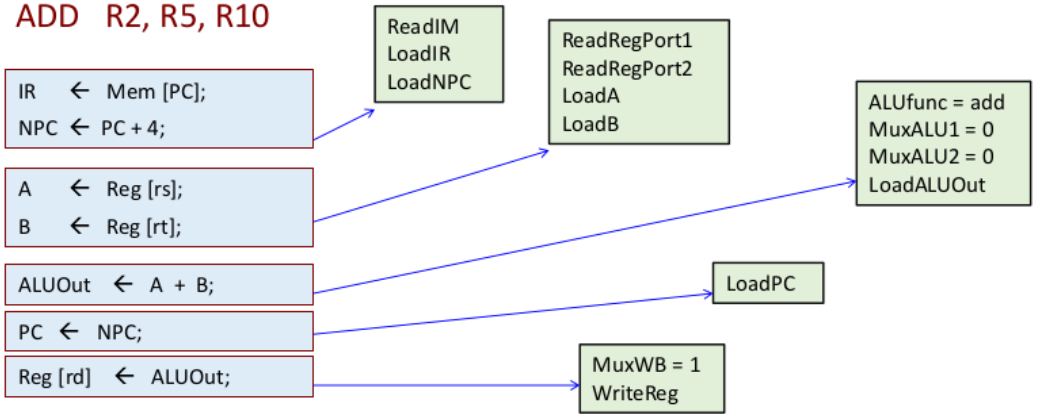
Simplicity of the Control Unit Design

- Due to the regularity in instruction encoding and simplicity of the instruction set, the design of the control unit becomes very easy.
- Control signals in the data path:

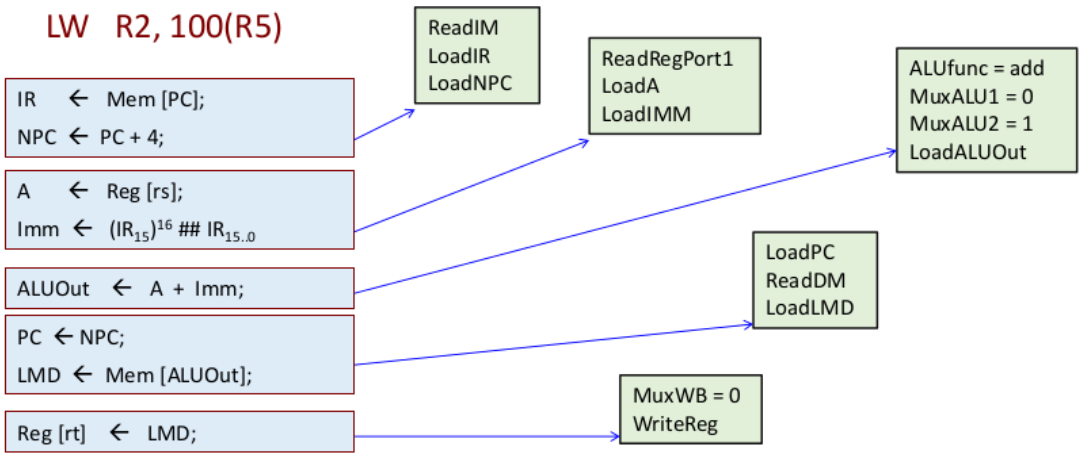
a) LoadPC
b) LoadNPC
c) ReadIM
d) LoadIR
e) ReadRegPort1
f) ReadRegPort2
g) LoadA
h) LoadB
i) LoadIMM
j) MuxALU1
k) MuxALU2
l) ALUfunc
m) LoadALUOut
n) MuxPC
o) ReadDM
p) WriteDM
q) LoadLMD
r) MuxWB
s) WriteReg

Control Signals for Some Instructions

ADD R2, R5, R10



LW R2, 100(R5)



Processor Memory Interaction

a) Volatile versus Non-volatile:

• A volatile memory system is one where the stored data is lost when the power is switched off.
• Examples: CMOS static memory, CMOS dynamic memory.
• Dynamic memory in addition, requires periodic refreshing.
• A non-volatile memory system is one where the stored data is retained even when the power is switched off.
• Examples: Read-only memory, Magnetic disk, CDROM/DVD, Flash memory, Resistive
memory.

b) Random-access versus Direct/Sequential access:

• A memory is said to be random-access when the read/write time is independent of the memory location being accessed.
• Examples: CMOS memory (RAM and ROM).
• A memory is said to be sequential access when the stored data can only be accessed sequentially in a particular order.
• Examples: Magnetic tape, Punched paper tape.
• A memory is said to be direct or semi-random access when part of the access
is sequential and part is random.
• Example: Magnetic disk.
• We can directly go to a track after which access will be sequential.

c) Read-only versus Random-access:

• Read-only Memory (ROM) is one where data once stored in permanent or semi-permanent.
• Data written (programmed) during manufacture or in the laboratory.
• Examples: ROM, PROM, EPROM, EEPROM.
• Random Access Memory (RAM) is one where data access time is the same independent of the location (address).
Can be read as well as written.
Used in main / cache memory systems.
Example: Static RAM (SRAM) → data once written are retained as long as power is on.
Example: Dynamic RAM (DRAM) → requires periodic refreshing even when power is on
(data stored as charge on tiny capacitors).

Difference between RAM and ROM

Difference	Random Access Memory (RAM)	Read Only Memory (ROM)
Data-Retention	RAM is a volatile memory that could store the data as long as the power is supplied.	ROM is a non-volatile memory that the could retain the data even when the power is turned off.
Read/Write	Read and write operations are supported.	Only read operations are supported.
Use	Used to store the data that has to be currently processed by CPU temporarily.	It is typically used to store firmware or microcode, which is used to initialize and control hardware components of the computer.
Speed	It is a high-speed memory.	It is much slower than the RAM.

CPU Interaction	CPU can easily access data stored in RAM.	CPU cannot easily access data stored in ROM.
Size and Capacity	Large size with higher capacity, concerning ROM.	Small size with less capacity, concerning RAM.
Used as/in	CPU Cache, Primary memory.	Firmware, Micro-controllers.
Accessibility	The data stored is easily accessible.	The data stored is not as easily accessible as in the concerning RAM.
Cost	RAM is more costlier than ROM.	ROM is cheaper than RAM.
Chip Size	A RAM chip can store only a few gigabytes (GB) of data.	A ROM chip can store multiple megabytes (MB) of data.

Function	Used for the temporary storage of data currently being processed by the CPU.	Used to store firmware, BIOS, and other data that needs to be retained.

Advantages of RAM

- **Speed:** RAM is much faster than other types of memory, such as hard disk drives, making it ideal for storing and accessing data that needs to be accessed quickly.
- **Volatility:** RAM is volatile memory, which means that it loses its contents when power is turned off. This property allows RAM to be easily reprogrammed and reused.
- **Flexibility:** RAM can be easily upgraded and expanded, allowing for more memory to be added as needed.

Disadvantages of RAM

- **Limited capacity:** RAM has a limited capacity, which can limit the amount of data that can be stored and accessed at any given time.
- **Volatility:** The volatile nature of RAM means that data must be saved to a more permanent form of storage, such as a hard drive or SSD, to prevent data loss.
- **Cost:** RAM can be relatively expensive, particularly for high-capacity modules, which can make it difficult to scale memory as needed.

Advantages of ROM

- **Non-volatile:** ROM is non-volatile memory, which means that it retains its contents even when power is turned off. This property makes ROM ideal for storing permanent data, such as firmware and system software.
- **Stability:** ROM is stable and reliable, which makes it a good choice for critical systems and applications.
- **Security:** ROM cannot be easily modified, which makes it less susceptible to malicious attacks, such as viruses and malware.

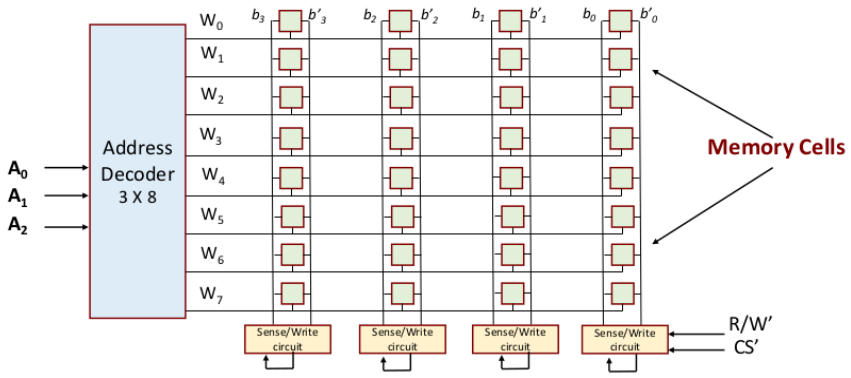
Disadvantages of ROM

- **Limited flexibility:** ROM cannot be easily reprogrammed or updated, which makes it difficult to modify or customize the contents of ROM.
- **Limited capacity:** ROM has a limited capacity, which can limit the amount of data that can be stored and accessed at any given time.
- **Cost:** ROM can be relatively expensive to produce, particularly for custom or specialized applications, which can make it less cost-effective than other types of memory.

• Terminologies used to measure speed of the memory system.
a) Memory Access Time: Time between initiation of an operation (Read or Write) and completion of that operation.
b) Latency: Initial delay from the initiation of an operation to the time the first data is available.
c) Bandwidth: Maximum speed of data transfer in bytes per second.
• In modern memory organizations, every read request reads a block of words into some high-speed registers (LATENCY), from where data are

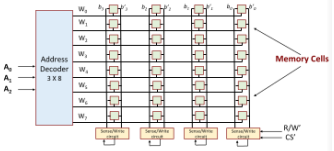
supplied to the processor one by one (ACCESS TIME).

Organization of Cells in an 8x4 Memory Chip



External Connection Requirements

- The 8 x 4 memory requires the following external connections:
  - Address decoder of size: 3 x 8
    - 3 external connections for address.
  - Data output : 4-bit
    - 4 external connections for data.
  - 2 external connections for R/W' and CS'.
  - 2 external connections for power supply and
  - Total of 3 + 4 + 2 + 2 = 11.



17

What About a 256 X 16 Memory?

- Here the total number of external connections are estimated as follows:
  - Address decoder size: 8 x 256
    - 8 external connections for address.
  - Data output : 16-bit
    - 16 external connections for data.
  - 2 external connections for R/W' and CS'.
  - 2 external connections for power supply and ground.
  - Total of 8 + 16 + 2 + 2 = 28.

Difference between Static RAM and Dynamic RAM

SRAM	DRAM

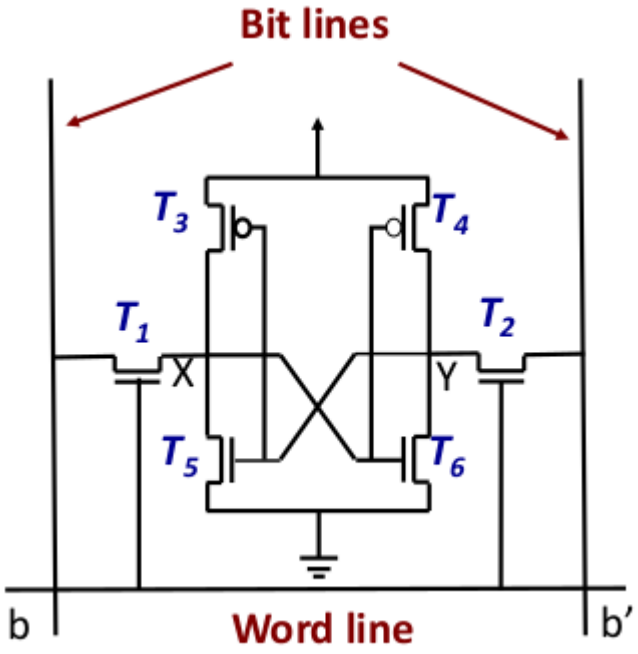
It stores information as long as the power is supplied.	*It stores information as long as the power is supplied or a few milliseconds when the power is switched off.*
Transistors are used to store information in SRAM.	*Capacitors are used to store data in DRAM.*
Capacitors are not used hence no refreshing is required.	*To store information for a longer time, the contents of the capacitor need to be refreshed periodically.*
SRAM is faster compared to DRAM.	*DRAM provides slow access speeds.*
It does not have a refreshing unit.	*It has a refreshing unit.*
These are expensive.	*These are cheaper.*

SRAMs are low-density devices.	DRAMs are high-density devices.
In this, bits are stored in voltage form.	In this, bits are stored in the form of electric energy.
These are used in cache memories.	These are used in main memories.
Consumes less power and generates less heat.	Uses more power and generates more heat.
SRAMs has lower latency	DRAM has more latency than SRAM
SRAMs are more resistant to radiation than DRAM	DRAMs are less resistant to radiation than SRAMs
SRAM has higher data transfer rate	DRAM has lower data transfer rate

SRAM is used in high-speed cache memory	***DRAM is used in lower-speed main memory***
SRAM is used in high performance applications	***DRAM is used in general purpose applications***

Static Random Access Memory (SRAM)

• SRAM consists of circuits which can store the data as long as power is
applied.
• It is a type of semiconductor memory that uses bistable latching circuitry (flip-flop) to store each bit.
• SRAM memory arrays can be arranged in rows and columns of memory cells.
• Called word line and bit line.
•
SRAM technology:
• Can be built using 4 or 6 MOS transistors.
• Modern SRAM chips in the market uses 6-transistor implementations for CMOS
compatibility.
• Widely used in small-scale systems like microcontrollers and embedded systems.
• Also used to implement cache memories in computer systems.
• To be discussed later.



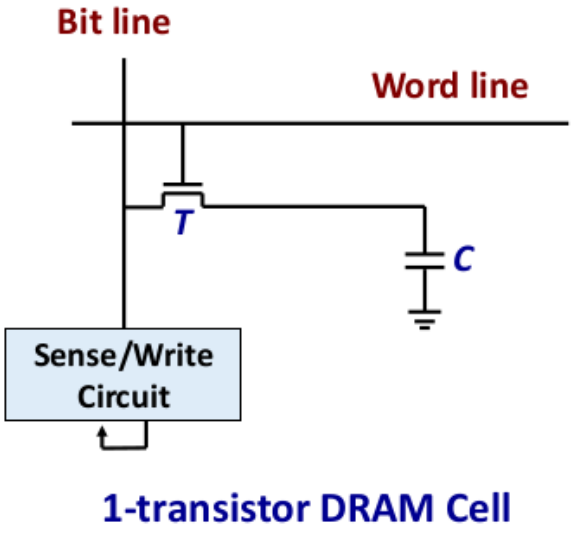
Features of SRAM

• Moderate / High power consumption.
• Current flows in the cells only when the cell is accessed.
• Because of latch operation, power consumption is higher than DRAM.
• Simplicity – refresh circuitry is not needed.
• Volatile :: continuous power supply is required.
• Fast operation.
• Access time is very fast; fast memories (cache) are built using SRAM.
• High cost.
• 6 transistors per cell.
• Limited capacity.
• Not economical to manufacture high-capacity SRAM chips.

Dynamic Random Access Memory (DRAM)

• Dynamic RAM do not retain its state even if
power supply is on.
Bit line
Word line
• Data stored in the form of charge stored on a
capacitor.
• Requires periodic refresh.
• The charge stored cannot be retained over long time
(due to leakage).

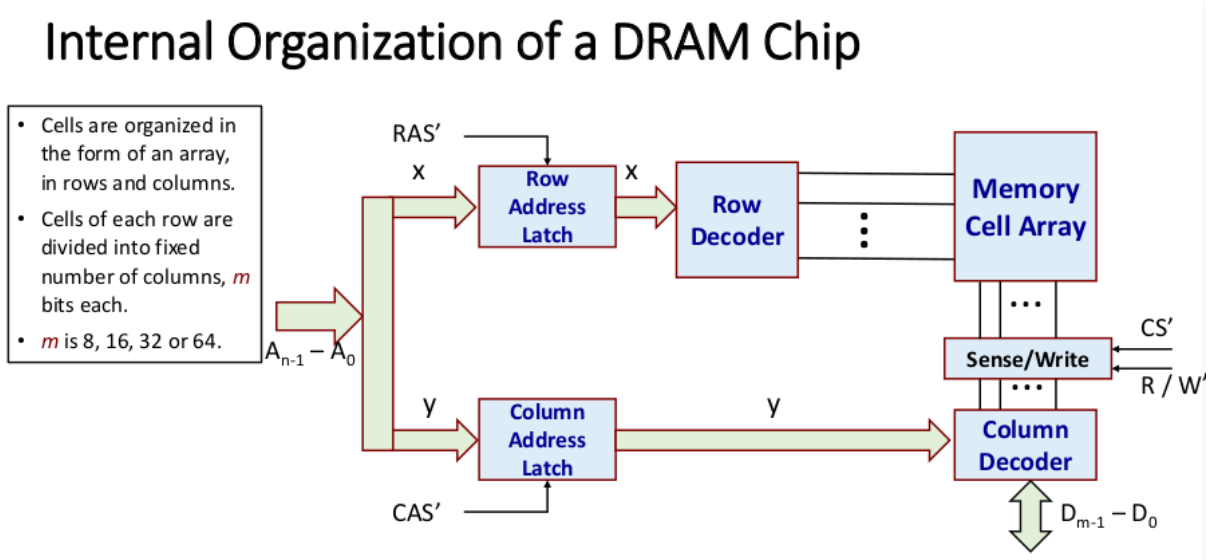
• Less expensive that SRAM.
• Requires less hardware (one transistor and one capacitor per cell).
• Address lines are multiplexed.



1-transistor DRAM Cell

Asynchronous DRAM

• The timing of the memory device is controlled asynchronously.
• The device connected to this memory is responsible for the delay.
• Address lines are divided into two parts and multiplexed.
• Upper half of address:
• Loaded into Row Address Latch using Row Address Strobe (RAS).
• Lower half of address:
• Loaded into Column Address Latch using Column Address Strobe (CAS).



• Suppose that the memory cell array is organized as r x c.

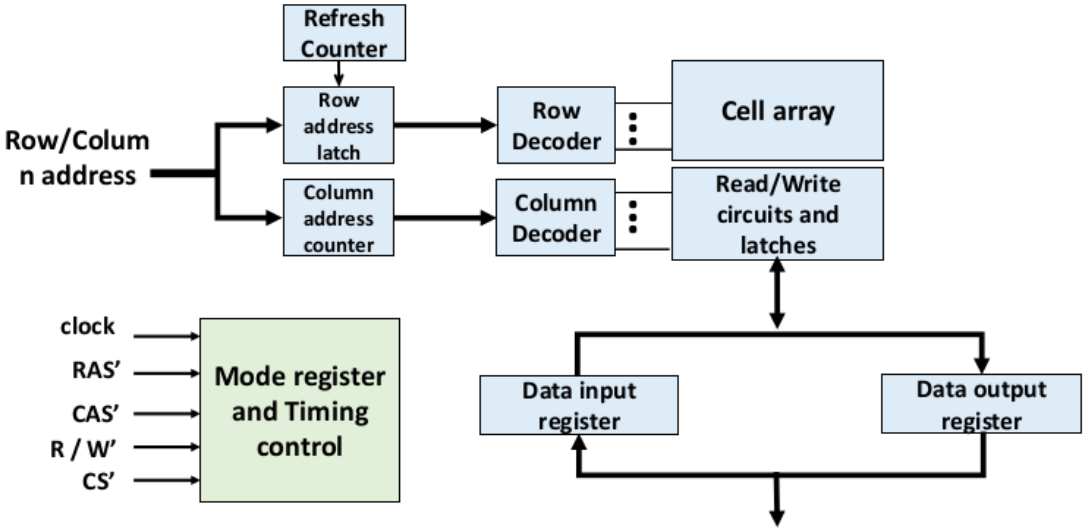
• r rows and c columns.
• An x-bit address is required to select a row r, where x = log 2 r.
• An y-bit address is required to select a column c, where y = log 2 c.
• Total address bits: n = x (high order) + y (low order)

READ or WRITE Operation
• For a read operation, the x-bit row address is applied first.
• It is loaded into Row Address Latch in response to the signal
RAS'.
• The read operation is performed in which all the cells of the selected row are read and refreshed.
• After loading of row address, the column address is selected.
• In response to CAS' the column address is loaded into Column Address Latch.
• Then the column decoder selects a particular column from c columns and an appropriate group of m sense/write circuits are selected.
• For a READ operation, the output values of the selected circuits are transferred to data lines D m-1 to D 0.
• For a WRITE operation, the data available on the data lines D m-1 to D 0 is transferred to the selected circuits.
• This information is stored in the selected cell.
• Both RAS' and CAS' are active low signals. That is they cause latching the addresses when they move from high to low.
• Each row of the cell array must be periodically refreshed to prevent data loss.
• Cost is low but access time is high compared to SRAM.
• Very high packing density (few billion cells per chip).
• Widely used in the main memory

Synchronous DRAM

• SDRAM is the commonly used name for various kinds of dynamic
RAM that are synchronized with clock.
• The structure of this memory is same as asynchronous DRAM.

Internal Organization of a SDRAM Chip



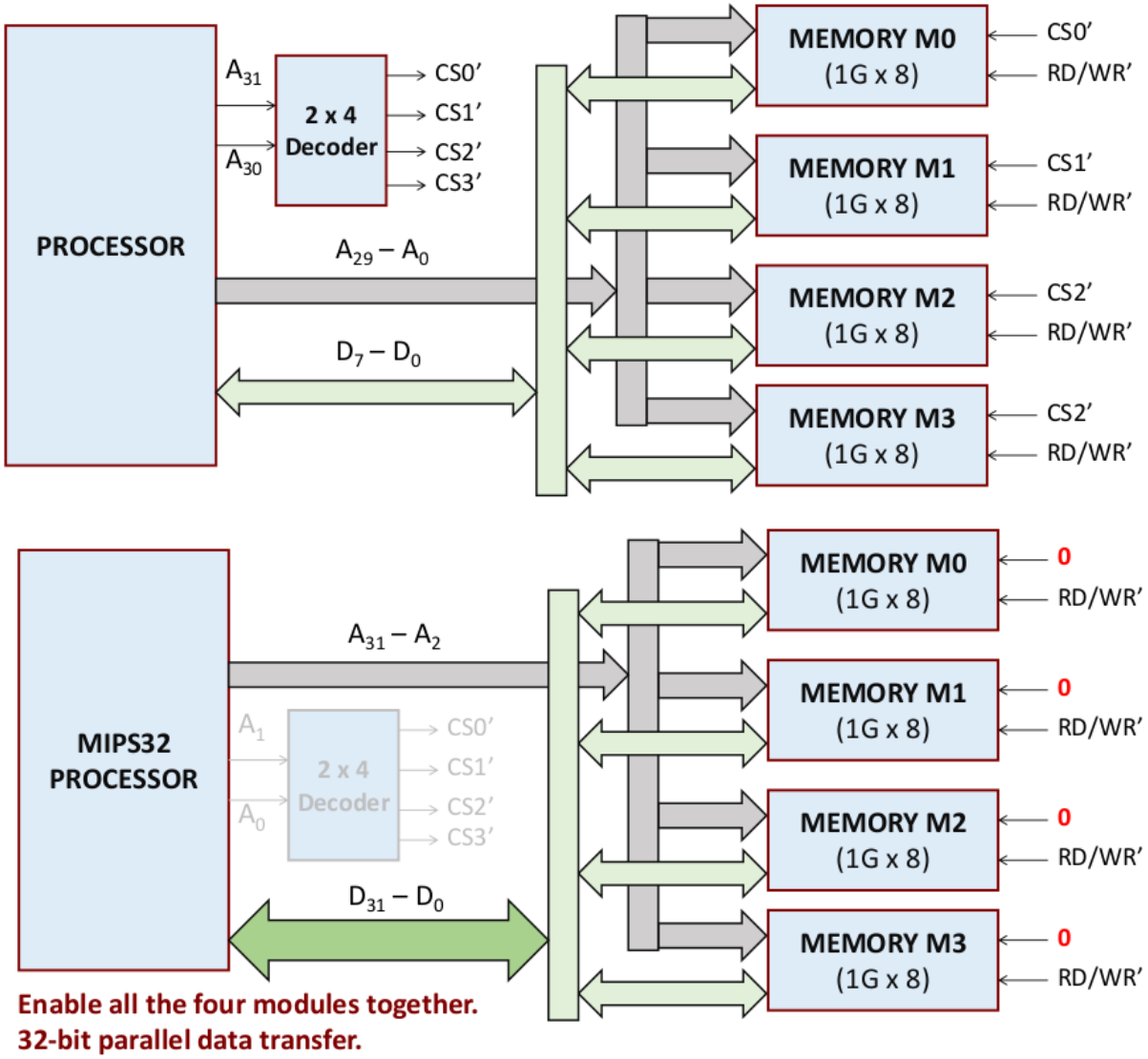
• In SDRAM address and data connections are buffered by registers.
• The output of individual sense amplifier is connected to a latch.
• Mode register is present which can be set to operate the memory chip in different modes.
• To select successive columns it is not required to provide externally generated pulses on CAS line.
• A column counter is used internally to generate the required signals.

READ and WRITE Operations
• For READ operation, the row address is applied first, and in response to the column address, the data present in the latches for the selected columns are transferred to the data output register.
• Then the data is available on the data bus.
• For WRITE operation, the row address is applied first, and in response to the column address, the data present in the data bus is made available to the latches through data input register.
• The data is then written to the particular cell.

Memory Interfacing and Addressing

• The data signals of a memory module (RAM) are typically bidirectional.
• Some memory chips may have separate data in and data out lines.
• For memory READ operation:
• Address of memory location is applied to address lines.
• RD/WR' control signal is set to 1, and CS' is set to 0.
• Data is read out through the data lines after memory access time delay.

• For memory WRITE operation:
• Address of memory location is applied to address lines, and the data to be written to data lines.
• RD/WR' control signal is set to 0, and CS' is set to 0.



Secondary Storage Devices:

Magnetic Disk (Hard Disk)

• Magnetic disks constitute a traditional method for non-volatile storage of information using magnetic technology.

• Broadly three types of devices appeared:

1) Floppy disk: made of bendable plastic
2) Magnetic drum: made of solid metal
3) Hard disk: made of metal or glass

• All of these rely on a rotating platter (metal or glass or plastic) coated with a thin magnetic material, and use a moveable read/write head to read and write data from/to the disk.

• Data stored as tiny magnets.

• Since the platters in a hard disk are made of rigid metal or glass, they provide several advantages over floppy disks:

• They can be larger.

• Can have higher density since they can be controlled more Precisely.

• Has a higher data rate because it spins faster.

• No physical contact with read/write head as it spins faster.

• The read/write head floats on a cushion of air (few microns separation).

• Requires dustless environment.

• Results in higher reliability.

• More than one platters can be incorporated in the same unit.

Hard Disk vs Floppy Disk

- The hard disk provides far larger storage capacities and much faster access time than floppy disks.
- The hard disk is cheaper than the floppy disk per megabyte.
- The hard disk is usually more reliable than a floppy disk.
- The hard disk is more durable compared to a floppy disk.

- hard disk store more data compared to a floppy disk.
- The hard disk is universal while floppy disk drive can only be used on obsolete machines.
- Access time is faster for hard disk than a floppy disk. the access time is 25 to msec while in floppy disk more than 100 msec.
- Hard disk holds more data.
- The life span of the hard disk is very long in comparison to floppies.
- The hard disk is faster than other diskettes.
- Nowadays you will not find floppies anywhere. There are some obsolete. Hard disk are used and survived the time bravely and gracefully. Still holding the front.
- Hard disk load and store data more quickly compared to a floppy disk.
- The storage system of the hard disk is available at varying storage capacity options.

Organization of Data on a Hard Disk

• The hard disks consists of a collection of platters (typically, 1 to 5), which are connected together and can spin in unison.
• Each platter has two recording surfaces, and comes in various sizes (1 – 8 inches).
• The stack of platter typically rotates at a speed of 5,400 to 10,000 rpm.
• Each disk surface is divided into concentric circles called tracks.
• The number of tracks per surface can vary from 1000 to 5000.
• Each track is divided into a number of sectors (64 – 200 sectors/track).
• Typical sector size: 512 – 2048 bytes.
• Sector is the smallest unit that can be read or written.
• The disk heads for all the surfaces are connected and move together.
• All the tracks under the heads at a given time on all surfaces is called a cylinder.

Disk Access Time

• There are three components to the access time in hard disk:

a) Seek time:

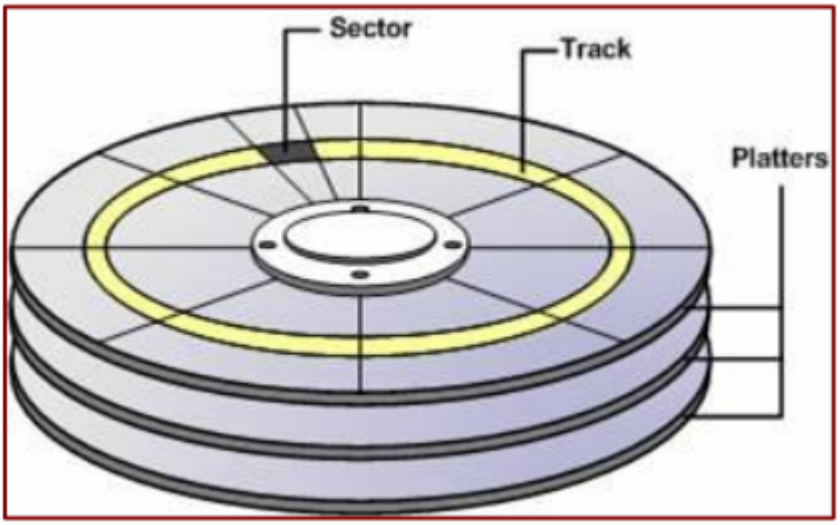
• The time required to move the head to the desired track.
• Average seek times are in the range 8 – 20 msec.
• Actual average can be 25 – 30% less than this number, since accesses to disks are often localized.

b) Rotational delay:

• Once the head is on the correct track, we must wait for the desired sector to rotate under the head.
• The average delay or latency is the time for half the rotation.
• Examples:
➢ For 3600 rpm, average rotational delay = 0.5 rotation / 3600 rpm = 8.30 msec
➢ For 5400 rpm, average rotational delay = 0.5 rotation / 5400 rpm = 5.53 msec
➢ For 7200 rpm, average rotational delay = 0.5 rotation / 7200 rpm = 4.15 msec

c) Transfer time:

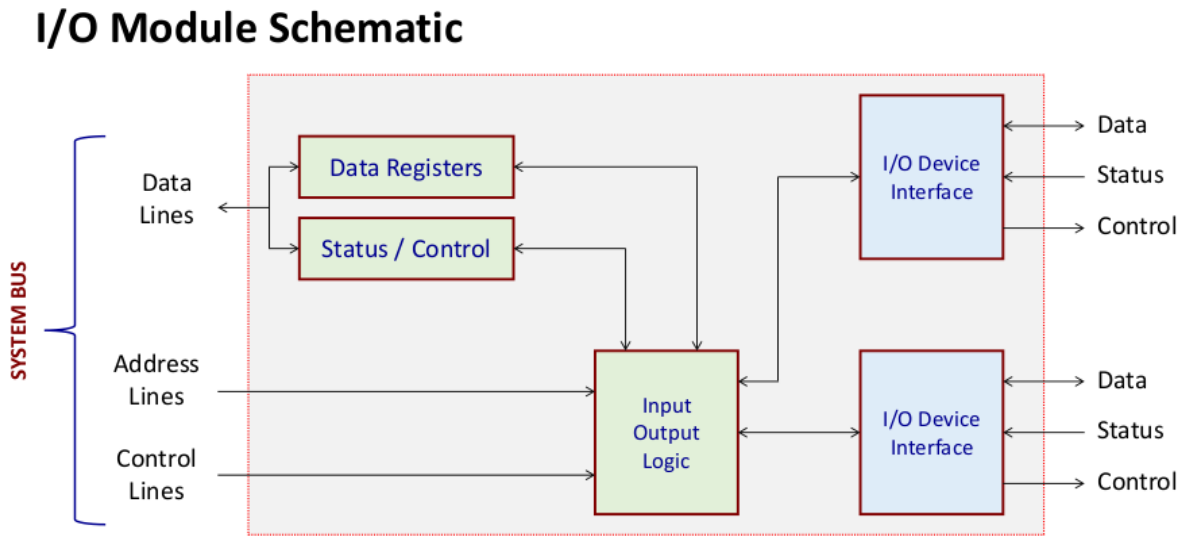
• The total time to transfer a block of data (typically, a sector).
• Transfer rates are typically 15 MB/sec or more.
• Transfer time depends on:
➢ Sector size
➢ Rotation speed of the disk
➢ Recording density on the tracks



Solid State Drives

Some features:
• Non-volatile
• Low power consumption
• Faster than hard disk
• Random access

• Data typically written block-wise (erase followed by write)



Typical Steps During I/O

a) Processor requests the I/O Module for device status.
b) I/O Module returns the status to the processor.
c) If the device is ready, processor requests data transfer.
d) I/O Module gets data from device (say, input device).
e) I/O Module transfers data to the processor.
f) Processor stores the data in memory.

How are I/O devices typically interfaced?

• Through input and output ports.
• Output port:
• Basically a PIPO register that is enabled when a particular output device address is given.
• The register inputs are connected to the data bus, and the register outputs are connected to the output device.
• Input port:
• Basically a parallel tristate bus driver that is enabled when a particular Input device address is given.
• The driver outputs are connected to the data bus, while the inputs are connected to the input device.

A tristate bus driver is a circuit or component used in digital electronics to control the flow of data on a bus, which is a set of wires or lines used to transmit binary information (0s and 1s) between various components within a computer or electronic system. The term "tristate" refers to the three possible output states that the driver can assume:

- High (1): When the driver is in the high state, it allows the bus line to carry a logical high voltage (often represented as a "1" in digital systems). This means that the bus line is actively transmitting a binary "1" signal.
- Low (0): When the driver is in the low state, it allows the bus line to carry a logical low voltage (often represented as a "0" in digital systems). This means that the bus line is actively transmitting a binary "0" signal.
- High Impedance (Hi-Z): When the driver is in the high-impedance state, it effectively disconnects the bus line from the rest of the circuit. In this state, the driver's output is electrically disconnected from the bus line, allowing other components to control the bus without interference.

A PIPO register, which stands for "Parallel-In, Parallel-Out" register, is a type of digital circuit or component commonly used in digital electronics and microprocessor systems. It serves as a temporary storage unit for data, allowing data to be loaded into the register in parallel (all bits at once) and then read out in parallel.

Key characteristics of a PIPO register include:

- Parallel Loading: Data can be loaded into the PIPO register simultaneously on all of its input bits. This is in contrast to a serial-in, parallel-out (SIPO) register, where data is loaded serially, one bit at a time.
- Parallel Output: Data can be read or output from the PIPO register in parallel, meaning all bits are available simultaneously on the output lines.
- Synchronous or Asynchronous Operation: PIPO registers can operate either synchronously or asynchronously. In synchronous operation, the loading and reading of data are synchronized with a clock signal, ensuring that data changes only at specific clock edges. Asynchronous operation, on the other hand, doesn't rely on a clock signal and allows data to be loaded and read without synchronization.