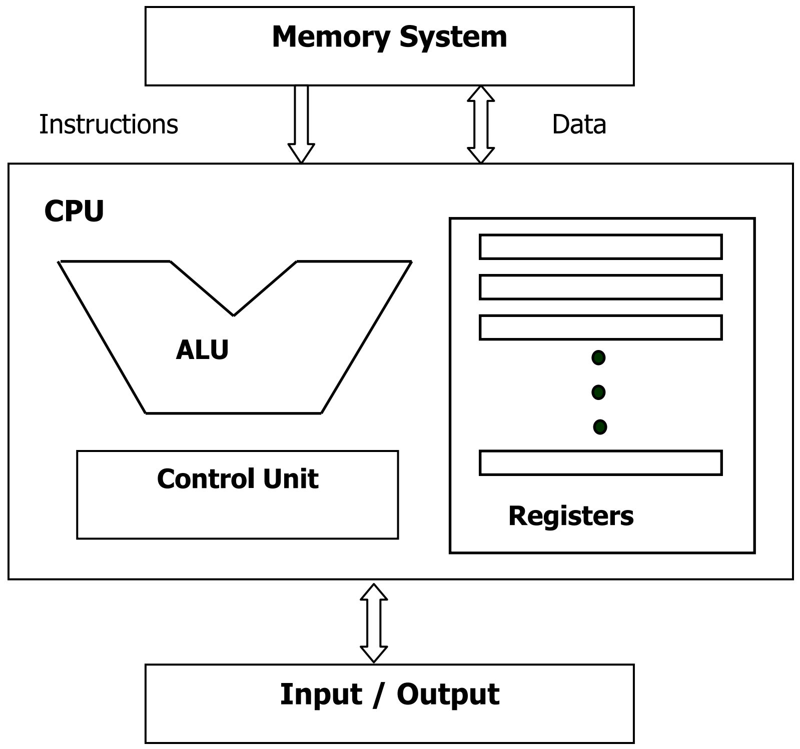
Chapter 5

Processing Unit Design

5.1 CPU Basics

- A typical CPU has three major components:
  - Register Set,
  - Arithmetic Logic Unit, and
  - Control Unit (CU).
  - The register set is usually a combination of general-purpose and special-purpose registers.
- General-purpose registers are used for any purpose.
- Special-purpose registers have specific functions within the CPU.

5.1 CPU Basics



Central Processing Unit Main Components and Interactions with the Memory and I/O

5.1 CPU Basics

- A typical and simple execution cycle in a CPU is as follows:
  - The next instruction to be executed, whose address is obtained from the PC, is fetched from the memory and stored in the IR.
  - Instruction is decoded.
  - Operands are fetched from the memory and stored in CPU registers, if needed.
  - Instruction is executed.
  - Results are transferred from CPU registers to the memory, if needed.
- The execution cycle is repeated as long as there are more instructions to execute.
- A check for pending interrupts is usually included in the cycle.

5.2 Register Set

- Memory Access Registers
  - Two registers are essential in memory write and read operations:
    - *memory data register* (*MDR*) and
    - *memory address register* (*MAR*).
  - The *MDR* and *MAR* are used exclusively by the CPU and are not directly accessible to programmers.
  - In order to perform a write operation into a specified memory location, the MDR and MAR are used as follows:
    - The word to be stored into the memory location is first loaded by the CPU into *MDR*
    - The address of the location into which the word is to be stored is loaded by the CPU into a *MAR*.

5.2 Register Set

- Memory Access Registers
  - Similarly, to perform a memory read operation, the MDR and MAR are used as follows:
    - The address of the location from which the word is to be read is loaded into the MAR.
    - The required word will be loaded by the memory into the MDR ready for use by the CPU.
- Instruction Fetching Registers
  - Two main registers are involved in fetching an instruction for execution:
    - the *program counter* (PC) and
    - the *instruction register* (IR).

5.2 Register Set

- Instruction Fetching Registers
  - The PC is the register that contains the address of the next instruction to be fetched.
  - The fetched instruction is loaded in the IR for execution.
  - After a successful instruction fetch, the PC is updated to point to the next instruction to be executed.
  - In the case of a branch operation, the PC is updated to point to the branch target instruction after the branch is resolved.

5.2 Register Set

- Condition Registers
  - Condition registers, or flags, are used to maintain status information.
  - Some architectures contain a special Program Status Word (PSW) register.
  - The PSW contains bits that are set by the CPU to indicate the current status of an executing program.
  - These indicators are typically for arithmetic operations, interrupts, memory protection information, or processor status.

5.2 Register Set

- Special Purpose Address Registers
  - Index register
    - The index register holds an address displacement.
    - Index addressing is indicated in the instruction by including the name of the index register in parentheses and using the symbol *X* to indicate the constant to be added.
  - Segment pointers
    - The address issued by the processor should consist of a Segment Number (Base) and a Displacement (or an offset) within the segment.
    - A segment register holds the address of the base of the segment.

5.2 Register Set

- Special Purpose Address Registers
  - Stack Pointer:
    - A stack is a data organization mechanism in which the last data item stored is the first data item retrieved.
    - Two specific operations can be performed on a stack. These are the *Push* and the *Pop* operations.
    - A specific register, called the *stack pointer (SP),* is used to indicate the stack location that can be addressed.
    - In the stack push operation, the SP value is used to indicate the location (called the top of the stack).
    - After storing (pushing) this value, the SP is incremented (In some architectures a SP is decremented (i.e. x86) as the stack grows low in memory).

5.2 Register Set

- 80x86 Registers
  - the Intel basic programming model of the 386, 486 and the *Pentium* consists of three register groups:
    - General-purpose registers,
    - Segment registers, and
    - The instruction pointer (program counter) & the flag register.

5.2 Register Set

- MIPS Registers

Name	Number	Usage	Name	Number	Usage
zero	0	Constant 0	s0	16	Saved temporary (preserved across call)
at	1	Reserved for assembler	s1	17	Saved temporary (preserved across call)
v0	2	Expression evaluation and	s2	18	Saved temporary (preserved across call)
v1	3	results of a function	s3	19	Saved temporary (preserved across call)
a0	4	Argument 1	s4	20	Saved temporary (preserved across call)
a1	5	Argument 2	s5	21	Saved temporary (preserved across call)
a2	6	Argument 3	s6	22	Saved temporary (preserved across call)
a3	7	Argument 4	s7	23	Saved temporary (preserved across call)
t0	8	Temporary (not preserved across call)	t8	24	Temporary (not preserved across call)
t1	9	Temporary (not preserved across call)	t9	25	Temporary (not preserved across call)
t2	10	Temporary (not preserved across call)	k0	26	Reserved for OS kernel
t3	11	Temporary (not preserved across call)	k1	27	Reserved for OS kernel
t4	12	Temporary (not preserved across call)	gp	28	Pointer to global area
t5	13	Temporary (not preserved across call)	sp	29	Stack pointer
t6	14	Temporary (not preserved across call)	fp	30	Frame pointer
t7	15	Temporary (not preserved across call)	ra	31	Return address (used by function call)

5.3 Datapath

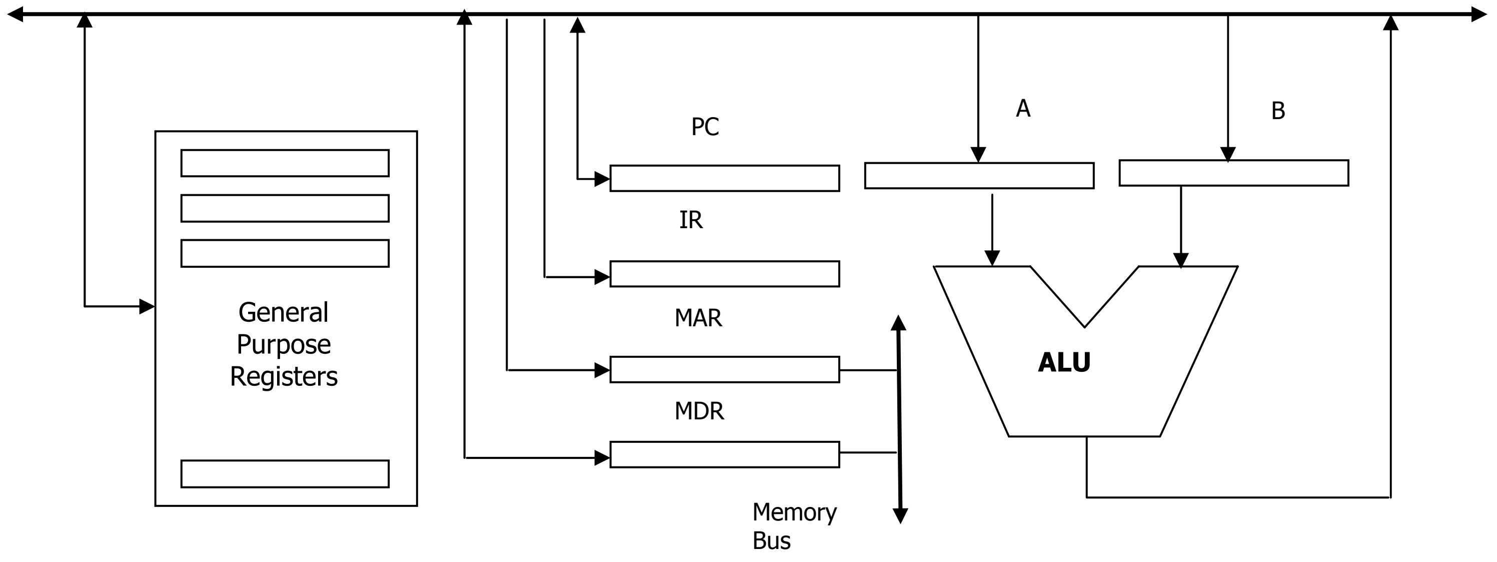
- The CPU can be divided into:
  - Data section
    - The data section, also called the datapath, contains the registers and the ALU.
    - The datapath is capable of performing certain operations on data items.
  - Control section
    - is basically the control unit, which issues control signals to the datapath.
  - Data move from one register to another and between ALU and registers.
  - Internal data movements are performed via local buses, which may carry data, instructions, and addresses.
    - Internal data movement among registers and between the ALU and registers may be carried out using different organizations including one-bus, two-bus or three-bus organizations.

5.3 Datapath

- One-Bus Organization
  - Using one bus, the CPU registers and the ALU use a single bus to move outgoing and incoming data.
  - Since a bus can handle only a single data movement within one clock cycle, two-operand operations will need two cycles to fetch the operands for the ALU.
  - Additional registers may also be needed to buffer data for the ALU.
  - This bus organization is the simplest and least expensive but it limits the amount of data transfer that can be done in the same clock cycle, which will slow down the overall performance.

5.3 Datapath

- One-Bus Organization

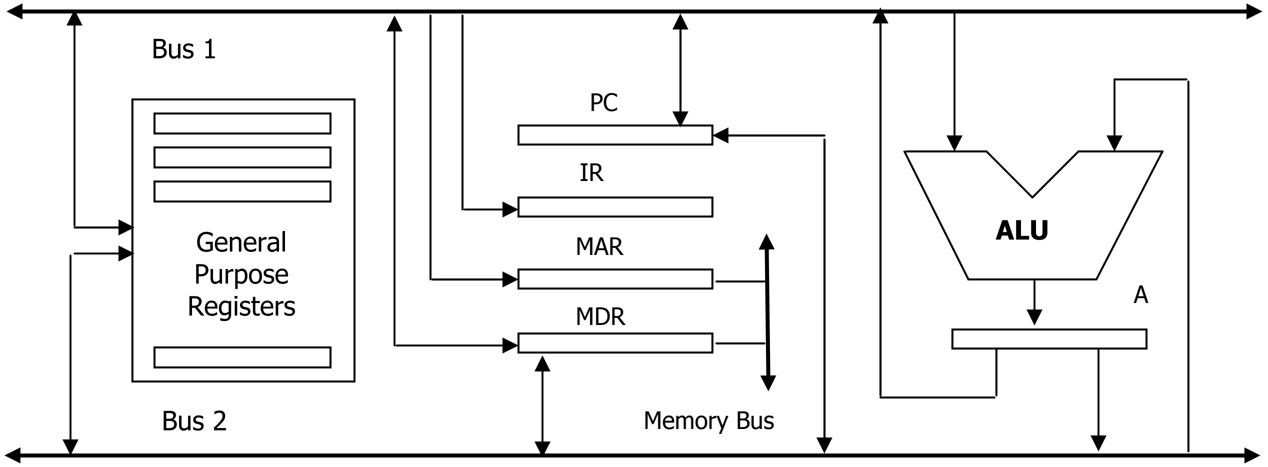


5.3 Datapath

- Two-Bus Organization
  - Using two buses is a faster solution than the one bus organization.
  - General-purpose registers are connected to both busses. Data can be transferred from two different registers to the input point of the ALU at the same time.
  - A two-operand operation can fetch both operands at the same clock cycle.
  - An additional buffer register may be needed to hold the output of the ALU when the two busses are busy carrying the two operands.
  - In some cases, one of the busses may be dedicated for moving data into registers (*in-bus*), while the other is dedicated for transferring data out of the registers (*out-bus*).

5.3 Datapath

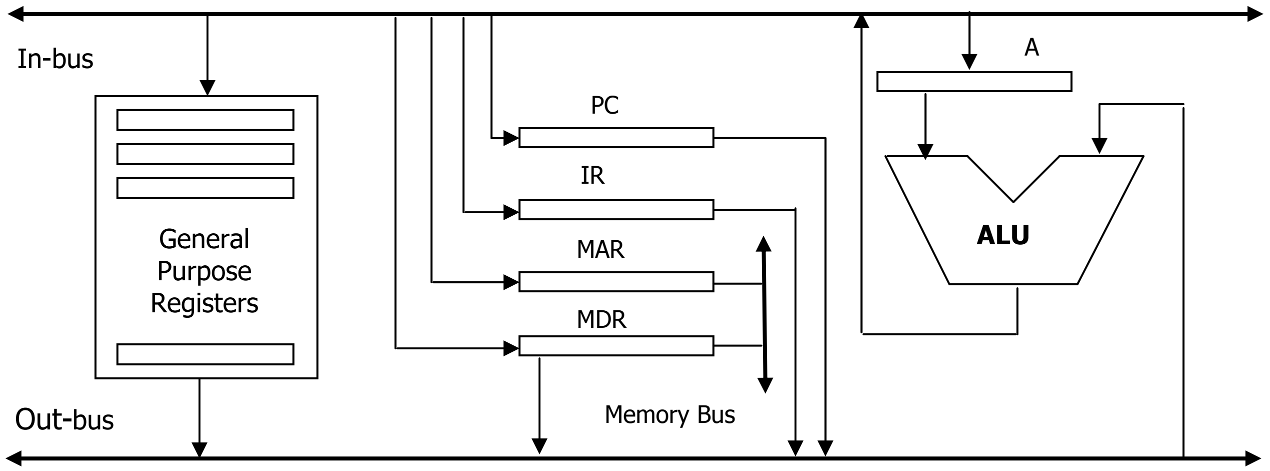
- Two-Bus Organization



Two-Bus Datapath

5.3 Datapath

- Two-Bus Organization



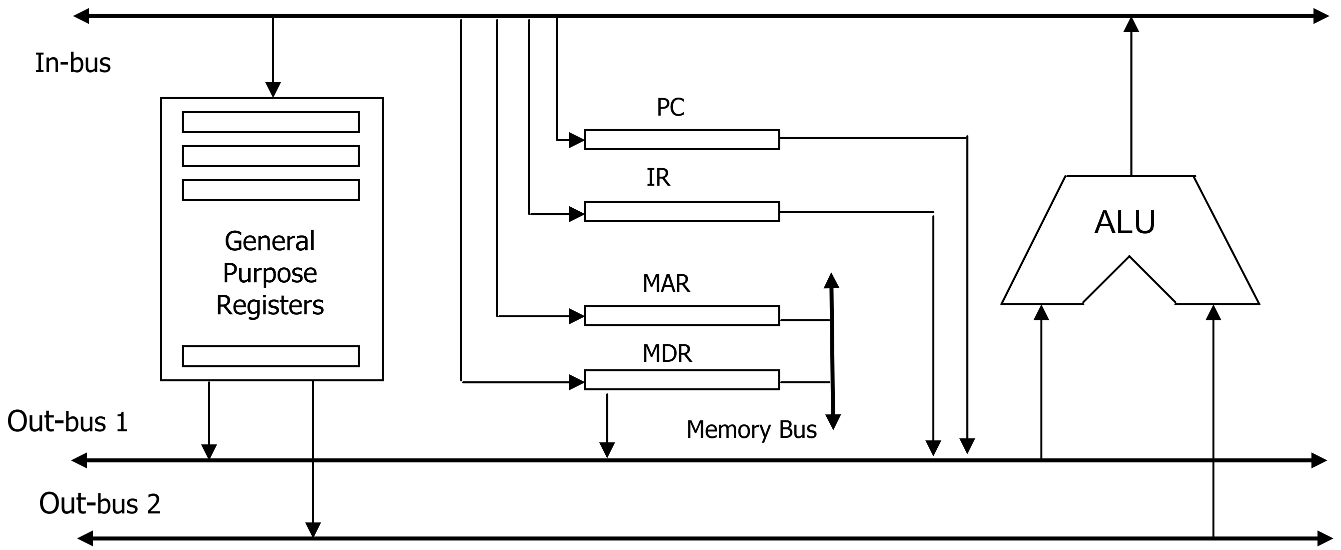
Two-Bus Datapath with in-bus and out-bus

5.3 Datapath

- Three-Bus Organization
  - Two buses may be used as source busses while the third is used as destination.
  - The source busses move data out of registers (*out-bus*), and the destination bus may move data into a register (*in-bus*).
  - Each of the two out-busses is connected to an ALU input point.
  - The output of the ALU is connected directly to the in-bus.
  - The more busses we have, the more data we can move within a single clock cycle.
  - However, increasing the number of busses will also increase the complexity of the hardware.

5.3 Datapath

- Three-Bus Organization

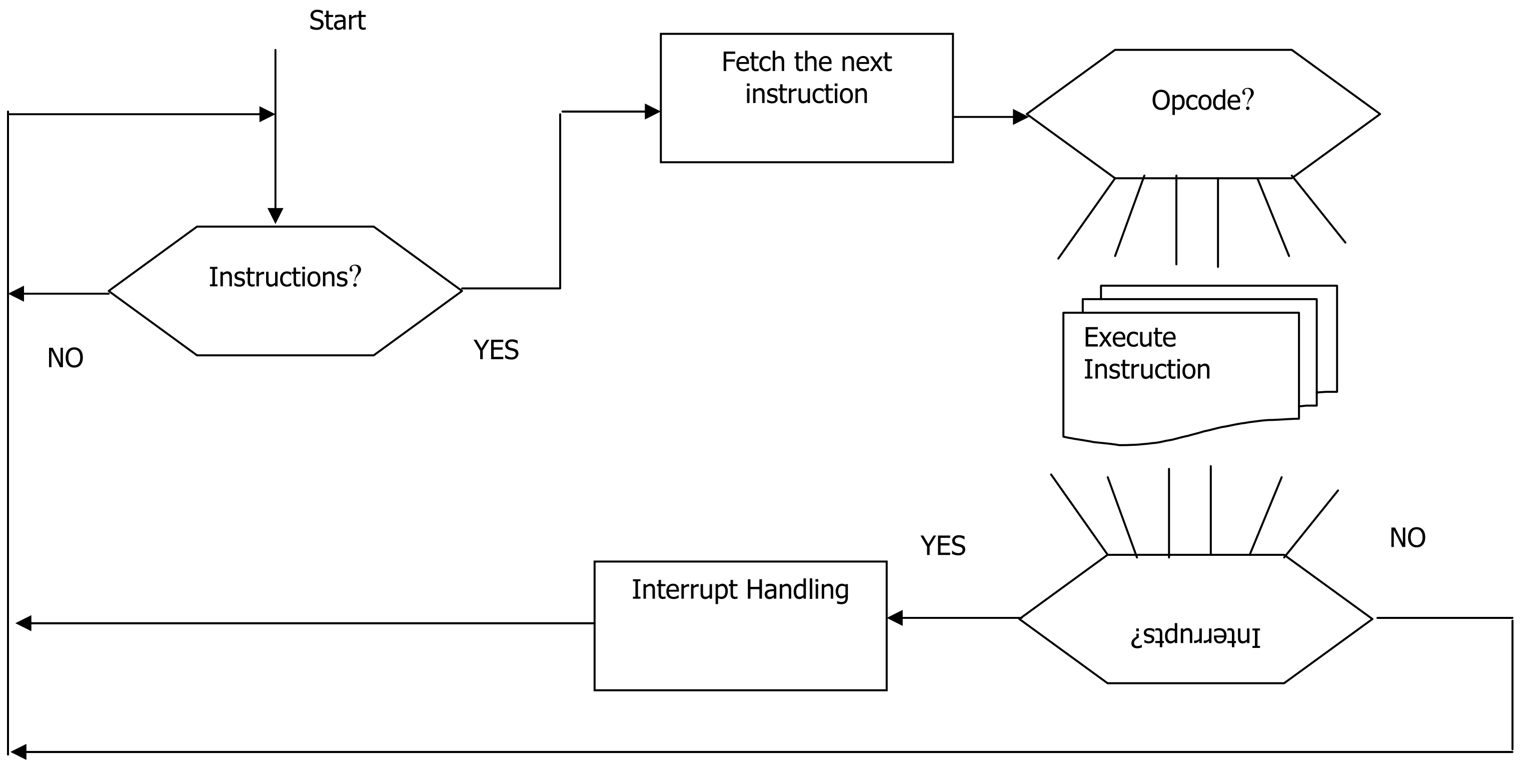


Three-Bus Datapath

5.4 CPU Instruction Cycle

- As long as there are instructions to execute, the next instruction is fetched from main memory.
- The instruction is executed based on the operation specified in the opcode field of the instruction.
- At the completion of the instruction execution, a test is made to determine whether an interrupt has occurred.
- An interrupt handling routine needs to be invoked in case of an interrupt.
- The basic actions during fetching an instruction, executing an instruction, or handling an interrupt are defined by a sequence of micro-operations.
- A group of control signals must be enabled in a prescribed sequence to trigger the execution of a micro-operation.

5.4 CPU Instruction Cycle



CPU Functions

5.4 CPU Instruction Cycle

- **Fetch Instructions**
  - The sequence of events in fetching an instruction can be summarized as follows:
    - The contents of the PC are loaded into the MAR.
    - The value in the PC is incremented. (This operation can be done in parallel with a memory access).
    - As a result of a memory read operation, the instruction is loaded into the MDR.
    - The contents of the MDR are loaded into the IR.

5.4 CPU Instruction Cycle

- **Fetch Instructions**

Step	**Micro-operation**
t_0	MAR $\leftarrow$ (PC); A $\leftarrow$ (PC)
t_1	MDR $\leftarrow$ Mem[MAR]; PC $\leftarrow$ (A) + 4
t_2	IR $\leftarrow$ (MDR)

One-bus datapath organization

5.4 CPU Instruction Cycle

- **Fetch Instructions**

Step	**Micro-operation**
t_0	MAR $\leftarrow$ (PC); PC $\leftarrow$ (PC) + 4
t_1	MDR $\leftarrow$ Mem[MAR]
t_2	IR $\leftarrow$ (MDR)

Three-bus datapath organization

5.4 CPU Instruction Cycle

- **Execute Simple Arithmetic Operation**
  - Add *R*1, *R*2, *R*0
    - This instruction adds the contents of source registers *R*1 and *R*2, and stores the results in destination register *R*0. This addition can be executed as follows:
      - The registers *R*0, *R*1, *R*2, are extracted from the IR.
      - The contents of *R*1 and *R*2 are passed to the ALU for addition.
      - The output of the ALU is transferred to *R*0.

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *R*1, *R*2, *R*0
    - One-bus datapath:

Step	**Micro-operation**
t_0	$A \leftarrow (R_1)$
t_1	$B \leftarrow (R_2)$
t_2	$R_0 \leftarrow (A) + (B)$

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *R*1, *R*2, *R*0
    - Two-bus datapath:

Step	**Micro-operation**
t_0	$A \leftarrow (R_1) + (R_2)$
t_1	$R_0 \leftarrow (A)$

    - Two-bus datapath with in-bus and out-bus:

Step	**Micro-operation**
t_0	$A \leftarrow (R_1)$
t_1	$R_0 \leftarrow (A) + (R_2)$

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *R*1, *R*2, *R*0
    - Three-bus datapath:

Step	Micro-operation
t_0	$R_0 \leftarrow (R_1) + (R_2)$

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *X*, *R*0
    - This instruction adds the contents of memory location *X* to register *R*0 and stores the result in *R*0. This addition can be executed as follows:
      - The memory location *X* is extracted from IR and loaded into MAR.
      - As a result of memory read operation, the contents of *X* are loaded into MDR.
      - The contents of MDR are added to the contents of *R*0.

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *X*, *R*0
    - One-bus datapath:

Step	**Micro-operation**
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	$A \leftarrow (R_0)$
t_3	$B \leftarrow (MDR)$
t_4	$R_0 \leftarrow (A) + (B)$

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *X*, *R*0
    - Two-bus datapath:

Step	Micro-operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	$A \leftarrow (R_0) + (MDR)$
t_3	$R_0 \leftarrow (A)$

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *X*, *R*0
    - Two-bus datapath with in-bus and out-bus:

Step	Micro-operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	$A \leftarrow (R_0)$
t_3	$R_0 \leftarrow (A) + (MDR)$

5.4 CPU Instruction Cycle

- Execute Simple Arithmetic Operation
  - Add *X*, *R*0
    - Three-bus datapath:

Step	**Micro-operation**
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	R_0 ← (R_0) + (*MDR*)

5.4 CPU Instruction Cycle

- Interrupt Handling
  - After the execution of an instruction, a test is performed to check for pending interrupts. If there is an interrupt request waiting, the following steps take place:
    - The contents of PC are loaded into MDR (to be saved).
    - The MAR is loaded with address at which the PC contents to be saved.
    - The PC is loaded with the address of the first instruction of the interrupt handling routine.
    - The contents of MDR (old value of the PC) are stored in memory.

5.4 CPU Instruction Cycle

- Interrupt Handling

Step	**Micro-operation**
t_1	MDR ← (PC)
t_2	MAR ← address1 (where to save old PC); PC ← address2 (interrupt handling routine)
t_3	Mem[MAR] ← (MDR)

5.5 Control Unit

- The control unit is the main component that directs the system operations by sending control signals to the datapath.
- These signals control the flow of data within the CPU and between the CPU and external units such as memory and I/O.
- Control busses generally carry signals between the control unit and other computer components in a clock-driven manner.
- The system clock produces a continuous sequence of pulses in a specified duration and frequency.

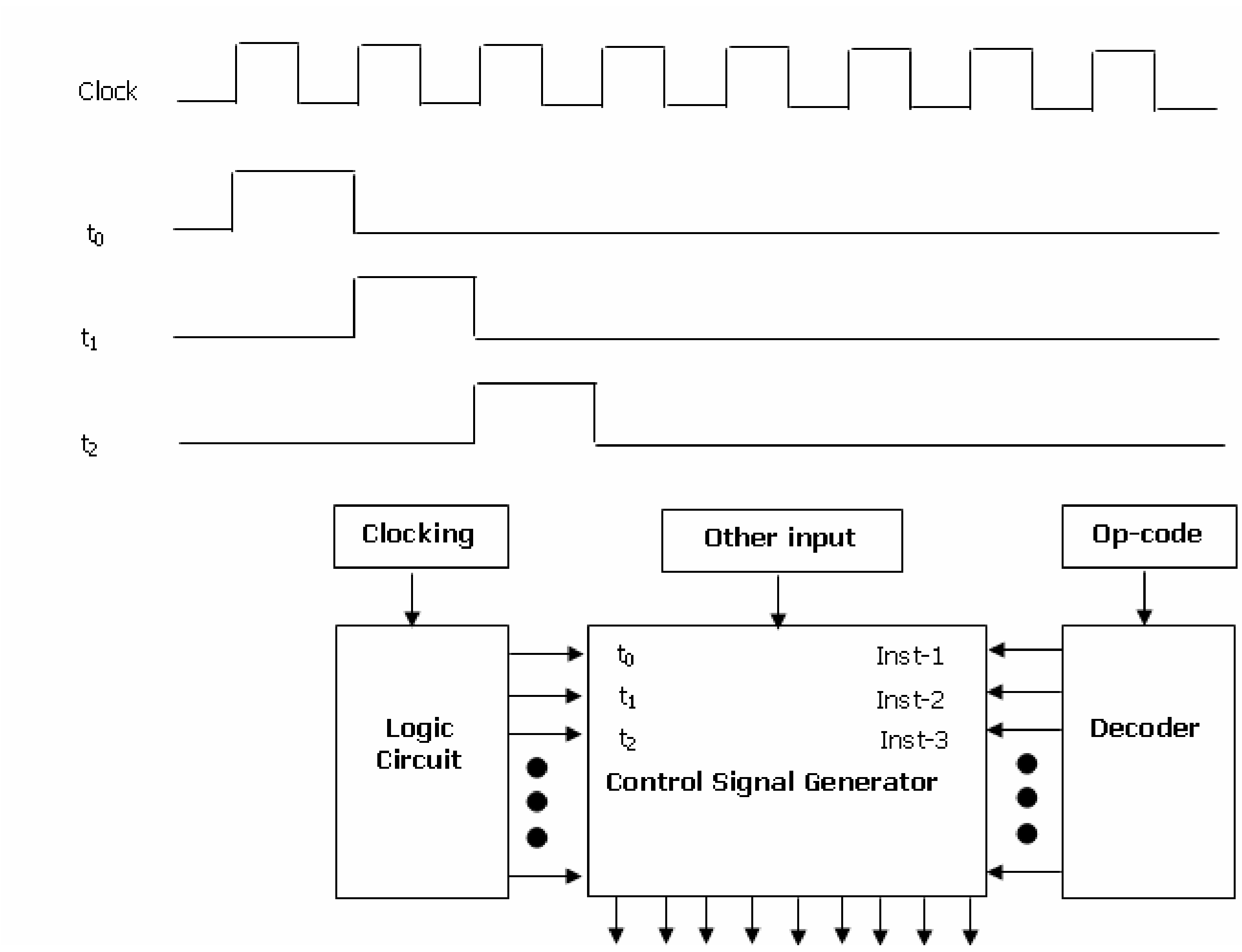
5.5 Control Unit

- A sequence of steps *t*0, t1, *t*2, … (*t*0 < *t*1 < *t*2 <…) are used to execute a certain instruction.
- The op-code field of a fetched instruction is decoded to provide the control signal generator with information about the instruction to be executed.
- Step information generated by a logic circuit module is used with other inputs to generate control signals.
- The signal generator can be specified simply by a set of Boolean equations for its output in terms of its inputs.

5.5 Control Unit

- There are mainly two different types of control units:
  - Microprogrammed
    - The control signals associated with operations are stored in special memory units inaccessible by the programmer as control words.
  - Hardwired
    - Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.

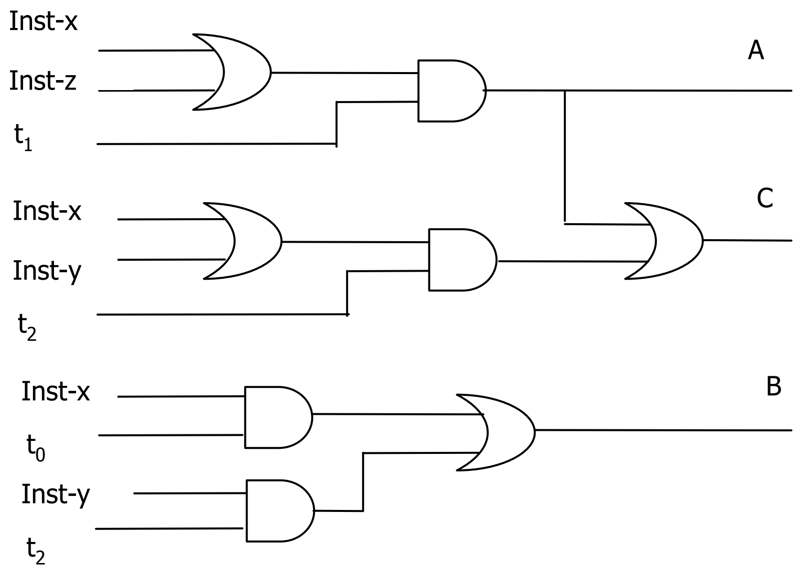
5.5 Control Unit



Timing of control signals

5.5 Control Unit

- Hardwired Implementation
  - In hardwired control, a direct implementation is accomplished using logic circuits.
  - For each control line, one must find the Boolean expression in terms of the input to the control signal generator.



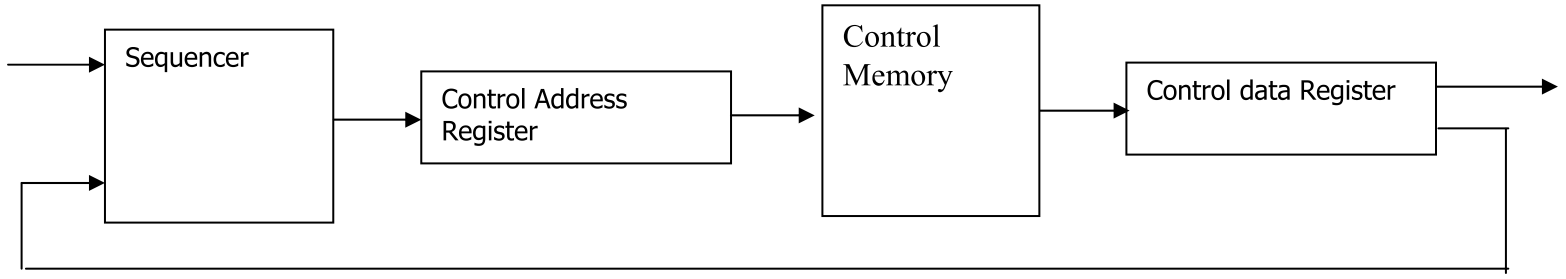
Logic circuits for control lines A, B, and C

5.5 Control Unit

- Microprogrammed Control Unit
  - Microprogramming was motivated by the desire to reduce the complexities involved with hardwired control.
  - Associated with each micro-operation is a set of control lines that must be activated to carry out the corresponding micro-operation.
  - The idea of microprogrammed control is to store the control signals associated with the implementation of a certain instruction as a microprogram in a special memory called a control memory (CM).
  - A microprogram consists of a sequence of microinstructions.
    - A microinstruction is a vector of bits, where each bit is a control signal, condition code, or the address of the next microinstruction.
    - Microinstructions are fetched from CM the same way program instructions are fetched from main memory.

5.5 Control Unit

- Microprogrammed Control Unit



Fetching Microinstructions (control words)

5.5 Control Unit

- Microprogrammed Control Unit
  - Horizontal versus Vertical Microinstructions
    - Individual bits in horizontal microinstructions correspond to individual control lines.
    - Horizontal microinstructions are long and allow maximum parallelism since each bit controls a single control line.
    - In vertical microinstructions, control lines are coded into specific fields within a microinstruction.
    - Decoders are needed to map a field of k bits to 2^k possible combinations of control lines.
    - Because of the encoding, vertical microinstructions are much shorter than horizontal ones.
    - Control lines encoded in the same field cannot be activated simultaneously. Therefore, vertical microinstructions allow only limited parallelism.

5.6 Summary

- The central processing unit (CPU) is the part of a computer that interprets and carries out the instructions contained in the programs we write.
- The CPU’s main components are the register file, ALU, and the control unit.
- The register file contains general-prupose and special registers.
  - General-purpose registers may be used to hold operands and intermediate results.
  - The special registers may be used for memory access, sequencing, status information, or to hold the fetched instruction during decoding and execution.

5.6 Summary

- Arithmetic and logical operations are performed in the ALU.
- Internal to the CPU, data may move from one register to another or between registers and ALU.
- Data may also move between the CPU and external components such as memory and I/O.
- The control unit is the compnent that controls the state of the instruction cycle.
- As long as there are instructions to execute, the next instruction is fetched from main memory.

5.6 Summary

- The instruction is executed based on the operation specified in the opcode field of the instruction.
- The control unit generates signals that control the flow of data within the CPU and between the CPU and external units such as memory and I/O.
- Control unit can be implemented using hardwired or microprogramming techniques.